Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki

Rozprawa Doktorska

Zastosowanie głębokiej eksploracji danych, w tym dyskretnego głębokiego uczenia, w indukcji reguł logicznych zawierających złożone warunki elementarne

Cezary Maszczyk

Promotor: prof. dr hab. Marek Sikora

Gliwice 2025

Spis treści

Spis tabel

Spis rysunków

Podziękowania

Serdecznie dziękuję mojemu Promotorowi, prof. dr. hab. Markowi Sikorze, za nieocenione wsparcie, inspirujące dyskusje i cenne wskazówki, które miały kluczowe znaczenie dla powstania niniejszej pracy.

Wyrazy wdzięczności kieruję również do dr. inż. Łukasza Wróbla za jego ogromne wsparcie merytoryczne oraz pomysłowość, które pozwoliły na powstanie tej pracy.

Szczególne podziękowania składam mgr. inż. Dawidowi Macha oraz dr. inż. Michałowi Kozielskiemu za ich pomoc i nieocenione wsparcie w prowadzeniu badań oraz przygotowywaniu publikacji.

Na koniec, z głębi serca dziękuję mojej żonie i rodzinie za cierpliwość i nieustanne wsparcie, które były dla mnie motywacją i opoką przez cały okres pisania pracy.

1 Wstęp

Wraz z dynamicznym rozwojem technologii, Sztuczna Inteligencja (AI) i Uczenie Maszynowe (ML) zyskały ogromną popularność, znajdując zastosowanie w niemal każdej dziedzinie życia, od medycyny przez finanse aż po przemysł. Popularność tych technologii wynika między innymi z ich zdolności do przetwarzania i analizowania dużych zbiorów danych, wykrywania złożonych wzorców oraz automatyzacji skomplikowanych procesów decyzyjnych [93].

Metody uczenia maszynowego i sztucznej inteligencji można podzielić na symboliczne i niesymboliczne - zwane również metodami subsymbolicznymi [30, 90]. Różnią się one działaniem oraz sposobem reprezentacji wyuczonej wiedzy. Symboliczna sztuczna inteligencja operuje na tak zwanych symbolach oraz regułach [166]. Z kolei metody subsymboliczne zamiast tego uczą się skomplikowanych wzorców i zależności wprost z surowych danych. Wiedza wyuczona przez model jest tutaj zwykle reprezentowana w sposób nieintuicyjny i trudny do zrozumienia dla człowieka. Przykładowo dla maszyn wektorów nośnych (SVM) zawarta jest ona w wyznaczonym zestawie przykładów zwanych wektorami nośnymi [42]. W przypadku głębokich sieci neuronowych z kolei, wiedza modelu jest zawarta w wyuczonych wartościach jego parametrów [72].

Każde z podejść oferuje unikalne korzyści i znajduje zastosowanie w różnych kontekstach. Metody subsymboliczne radzą sobie z dużymi, nieustrukturyzowanymi i zaszumionymi zbiorami danych dużo lepiej od symbolicznych [3]. Z racji swej skomplikowanej struktury są one jednak traktowane jako tak zwane czarne skrzynki [3]. Oznacza to, że dokładny sposób, w jaki podejmowana jest dana decyzja, jest trudny lub wręcz niemożliwy do prześledzenia i zrozumienia. Stanowi to duży problem w dziedzinach życia związanych z etyką, takich jak prawo czy medycyna, gdzie istnieje potrzeba ręcznej weryfikacji poprawności działania modelu [194]. W metodach symbolicznych wyuczona przez model wiedza jest reprezentowana w sposób zrozumiały przez człowieka. Możliwe jest także prześledzenie i objaśnienie procesu podejmowania decyzji przez model [57]. Dzięki tym cechom metody symbolicznego uczenia maszy-

nowego znajdują szczególne zastosowanie w procesie tak zwanego odkrywania wiedzy lub też eksploracji danych (z ang. knowledge discovery and data mining, KDD). Polega on na identyfikacji istotnych i potencjalnie użytecznych wzorców w danych [165]. Obecnie symboliczne metody uczenia maszynowego zyskują na popularności w kontekście modnego ostatnio pojęcia wyjaśnialnej sztucznej inteligencji XAI(z ang. Explainable AI - XAI). Określa ono dziedzinę sztucznej inteligencji, skupiającą się na tworzeniu modeli, których decyzje są przejrzyste i zrozumiałe dla człowieka [77]. Celem jest zapewnienie możliwości weryfikacji i kontroli, a co za tym idzie zwiększenie zaufania względem systemów AI i ML [7]. Symboliczne metody uczenia maszynowego, inherentnie interpretowalne, świetnie wpisują się w tę koncepcję [149].

Powyższe metody znajdują zastosowanie głównie w dwóch komplementarnych typach zadań: predykcji oraz eksploracji wiedzy.

W zadaniu predykcji celem jest przewidywanie wartości zmiennej zależnej na podstawie danych wejściowych. Metody subsymboliczne, takie jak sztuczne sieci neuronowe, są tutaj częściej wybierane ze względu na osiąganą przez nie na ogół wyższą dokładność predykcji. Istnieją liczne badania potwierdzające ich wyższość pod tym względem w zadaniach takich jak klasyfikacja, regresja czy rozpoznawanie obrazów [94, 104].

Jednakże w kontekście odkrywania wiedzy i objaśniania decyzji podejmowanych przez systemy AI metody symboliczne, a w szczególności modele regułowe, dominują pod względem przydatności [66]. Dzieje się tak dlatego, że wiedza wyuczona przez modele regułowe jest reprezentowana w zrozumiałej dla człowieka postaci symbolicznej. Umożliwia to prześledzenie i objaśnianie procesu decyzyjnego. To właśnie ta zdolność do introspekcji bywa kluczowa w wielu zastosowaniach, gdzie wymagana jest kontrola nad działaniem systemu [149].

Jednymi z najpopularniejszych metod symbolicznego uczenia maszynowego są systemy regułowe oraz drzewa decyzyjne. Te pierwsze posiadają jednak pewne unikalne zalety względem drugich. Metody regułowe z racji swojej budowy są w stanie prowadzić do bardziej zwięzłych, mniej złożonych, a co za tym idzie, bardziej interpretowalnych opisów danych [66, 144]. Wynika to ze sposobu, w jaki budowane są drzewa decyzyjne, gdzie każda ze ścieżek pokrywa inny zestaw przykładów, nie mogąc przy tym pokrywać żadnych przykładów wspólnych. Ta rozłączność nie jest wymagana w przypadku większości algorytmów regułowych. Dodatkowo metody regułowe, takie jak Ripper, na ogół generalizują lepiej od drzew decyzyjnych, będąc przy tym bardziej odpornymi na problem przetrenowania [38, 66].

1 Podstawowe definicje i opis problemu

Rozważmy zbiór danych oznaczony jako $D(A, \delta)$, zawierający n przykładów uczących. Każdy taki przykład jest opisany m-elementowym zbiorem atrybutów $A = \{a_1, a_2, ..., a_{|A|}\}$ oraz specjalnym atrybutem zwanym atrybutem decyzyjnym δ. Najczęściej wyróżnia się dwa podstawowe rodzaje atrybutów: numeryczne i nominalne. Pierwsza grupa to zmienne ciągłe, które opisują pewne cechy ilościowe, takie jak np. wiek. Atrybuty nominalne, zwane również kategorycznymi, przyjmują dyskretne wartości z pewnej określonej domeny wartości, np. płeć. Jako W zostanie oznaczona średnia liczba wartości w dziedzinach wszystkich atrybutów kategorycznych.

Forma i interpretacja atrybutu decyzyjnego mogą się różnić w zależności od rodzaju problemu, który jest rozwiązywany. W ramach tej pracy uwagę skupiono na trzech typach problemów: klasyfikacji, regresji i analizie przeżycia.

Dla problemu klasyfikacji wartość atrybutu decyzyjnego określa przynależność przykładu do pewnej klasy. W takim wypadku δ jest dyskretną zmienną, identyfikującą klasę $\delta \in \{L_1, L_2, ..., L_{|L|}\}$. Problem klasyfikacji polega na zgadnięciu poprawnej klasy L_j dla zadanego przykładu X_i. W przypadku problemu regresji atrybut decyzyjny jest zmienną ciągłą: $\delta \in \mathbb{R}$. Rozwiązanie problemu polega na przewidzeniu wartości atrybutu decyzyjnego z jak najmniejszym - idealnie zerowym - błędem. W analizie przeżycia atrybut decyzyjny jest zmienną binarną, wskazującą na to, czy dla danego przykładu wystąpiło pewne zdarzenie, czy też nie, np. nawrót choroby u pacjenta. Dla tego rodzaju problemu konieczne jest, by wśród atrybutów wystąpił dodatkowy atrybut numeryczny T_i., określający tzw. czas przeżycia. Jeżeli dla danego przykładu status przeżycia wskazuje na wystąpienie zdarzenia, jest on rozumiany jako czas, który upłynął do jego wystąpienia. W przeciwnym wypadku jest on równy całkowitemu czasowi trwania obserwacji przykładu. Dla problemów klasyfikacji i regresji każdy przykład ze zbioru D reprezentowany jest przez wektor cech $x_i = (a_{i1}, a_{i2}, \ldots, a_{i|A|}, \delta_i)$. W przypadku problemu przeżyciowego wektor ten należy rozszerzyć o dodatkową zmienną T_i.

Reguły decyzyjne dla każdego z wyżej wymienionych typów problemów mogą zostać zapisane w następującej formie:

$$\textbf{JEŻELI } c \textbf{ TO } \delta = f(X)$$

W dalszej części pracy wykorzystywana bywa również popularniejsza i zwięźlejsza wersja zapisu w języku angielskim:

IF c **THEN** $\delta = f(X)$

Gdzie c jest dowolnym warunkiem logicznym (hipotezą), którego prawdziwość można ustalić na podstawie zmiennych należących do A. Ten element reguły nazywany jest jej przesłanką. Jeżeli warunek zawarty w przesłance reguły jest prawdziwy dla danego przykładu, mówimy, że przykład ten pokrywany jest przez regułę. Liczby przykładów pozytywnych i negatywnych pokrywanych przez regułę (dla problemu klasyfikacji) oznaczone są kolejno literami p i n, zgodnie z powszechnie przyjętą konwencją. Literami P i N oznaczone są z kolei liczby wszystkich przykładów pozytywnych i negatywnych w całym zbiorze D. Przesłanka reguły przyjmuje najczęściej postać koniunkcji lub alternatywy warunków. W ramach poniższej pracy warunki typu $a =$ wartość (dla atrybutów nominalnych) oraz $a >$ wartość lub $a <$ wartość dla atrybutów numerycznych nazywane będą warunkami prostymi. Są one wykorzystywane w większości popularnych metod uczenia regułowego [36, 37, 74, 86]. Oprócz nich można zdefiniować szereg innych warunków, które mogą zostać wykorzystane do budowania reguł decyzyjnych. Dokładniejszy ich opis wraz z przykładami wykorzystania zaczerpniętymi z literatury znajduje się w sekcji 3. Warunki logiczne w ramach tej pracy oznaczone są literą c a ich zbiór literą C.

Część reguły występująca po **TO** nazywana jest konkluzją. Konkluzja określa część decyzyjną reguły i może posiadać różną postać w zależności od typu adresowanego problemu. W przypadku klasyfikacji określa ona zwykle klasę decyzyjną przykładu [119]. Dla problemu regresji konkluzja zawiera funkcję określającą numeryczną wartość atrybutu decyzyjnego. Wartość ta wyznaczana bywa w różny sposób, jako średnia [110] lub mediana [74] pokrytych przykładów, czy też poprzez minimalizację pewnej funkcji straty [48]. W przypadku analizy przeżycia konkluzja zawiera pewien estymator funkcji przeżycia dla przykładów pokrywanych przez regułę. Najczęściej wykorzystywany jest w tym celu estymator Kaplana-Meiera [16, 103, 162].

2 Motywacja i cel pracy

Większość tradycyjnych metod symbolicznego uczenia maszynowego konstruuje stosunkowo proste opisy danych w postaci reguł lub drzew decyzyjnych. W przypadku modeli regułowych są one oparte najczęściej na logice zdaniowej (a nie w bardziej ekspresywnej logice predykatów) [159]. Metody te wykorzystują zwykle do opisu

danych wcześniej zdefiniowane warunki proste (atrybut > wartość lub atrybut < wartość, lub atrybut = wartość) [27, 36, 37, 39]. Mogą one jednak być nieefektywne w sytuacjach, gdy granice decyzyjne w danych mają bardziej złożony, nieliniowy kształt [26]. W takich przypadkach drzewa decyzyjne mogą stać się bardzo złożone i głębokie, próbując aproksymować te granice za pomocą wielu równoległych względem osi cięć [198]. Podobna sytuacja występuje w przypadku wielu algorytmów regułowych, gdzie indukowane zbiory reguł mogą stać się z tego powodu przesadnie skomplikowane [161]. Uzyskiwanie możliwie jak najprostszych, a przez to łatwych w interpretacji dla człowieka modeli jest w przypadku symbolicznych metod uczenia maszynowego oraz odkrywania wiedzy sprawą niezwykle istotną. Jak wykazują liczne badania, to właśnie od złożoności modelu w dużej mierze zależna jest jego interpretowalność [27, 68, 142].

Zastosowanie bardziej złożonych warunków logicznych w symbolicznych metodach uczenia maszynowego może prowadzić do uzyskiwania prostszych, mniej rozbudowanych, a przy tym łatwiejszych w zrozumieniu dla człowieka modeli. Dodatkowo potencjalnie poszerza ich możliwości wykrywania i opisywania nowych wzorców w danych, które nie mogą być opisane za pomocą warunków prostych.

W niniejszej pracy uwagę skupiono na algorytmach regułowych. Generowane przez nie opisy uznawane są za jedne z najłatwiejszych do interpretacji przez człowieka, a mimo to pozwalają uzyskiwać relatywnie wysoką jakość predykcji. Niemal wszystkie istniejące metody pozwalają na indukcję zbiorów reguł w jednej z dwóch postaci: koniunkcyjnej (CNF) lub dysjunkcyjnej (DNF) postaci normalnej [66]. Choć, jak zostało to udowodnione [85], każde wyrażenie logiczne można przedstawić na obydwa sposoby. Istnieją jednak takie, dla których jedna z nich jest znacząco krótsza od drugiej [69, 176]. W ostatnich latach w literaturze pojawiło się pojęcie głębokiego dyskretnego uczenia (DDL), wprowadzone przez Petera Flach i Johannesa Fürnkranz [8]. Odnosi się ono do uczenia zbiorów reguł o bardziej złożonej postaci, gdzie przesłanki reguł nie muszą być prostymi koniunkcjami lub alternatywami warunków. Zamiast tego mogą składać się z zagnieżdżonych, hierarchicznych wyrażeń logicznych. Zbiory tego typu reguł potrafią w bardziej zwięzły i skuteczny sposób opisywać niektóre relacje w danych, umykające klasycznym metodom regułowym.

Wspominana idea dyskretnego głębokiego uczenia stanowi tutaj punkt wyjścia dla badań opisanych w niniejszej pracy. Autor jednak rozszerza jej rozumienie, skupiając się nie tylko na indukcji bardziej skomplikowanych i zagnieżdżonych formuł logicznych, lecz także na wykorzystaniu bardziej złożonych warunków wewnątrz przesłanek reguł.

W pracy tej nie zdefiniowano tezy głównej ani tez pomocniczych. Skoncentrowano się zamiast tego na określonych celach (celu głównym i celach szczegółowych), które można zdefiniować w następujący sposób:

Celem niniejszej rozprawy było opracowanie nowych metod głębokiego dyskretnego uczenia, zastosowanych do budowy regułowych modeli danych. Wyznaczone modele powinny skutecznie rozwiązywać problemy klasyfikacji, regresji oraz analizy przeżycia. W szczególności w pracy zaadresowano następujące cele szczegółowe:

1. *Zaproponowanie metod pozwalających na wyznaczanie reguł, w których przesłankach pojawiają się złożone warunki elementarne.*
2. *Zaproponowanie metody pozwalającej na indukcję złożonych wyrażeń logicznych, zawierających zagnieżdżone operatory koniunkcji i alternatywy.*
3. *Eksperymentalna weryfikacja efektywności zaproponowanych metod zarówno w zakresie zdolności predykcyjnych, jak i opisowych, wyznaczanych zbiorów reguł.*

Autor realizuje wyżej opisany cel główny, proponując trzy metody uczenia opartego na regułach, opisane w rozdziale 6. Wszystkie z nich adresują dodatkowo pierwszy spośród celi szczegółowych pracy. Metoda ComplexRules skupia się na generowaniu reguł z bardziej złożonymi formami warunków wewnątrz przesłanek. Dzięki temu jest ona w stanie odkrywać i opisywać bardziej skomplikowane zależności i w pewnych przypadkach tworzyć bardziej zwięzłe opisy danych. Druga z metod o nazwie MofNRules, rozszerza ją, dodając do niej warunki M-z-N (patrz rozdział 4 sekcja 3 podsekcja 4). Warunki tego typu pozwalają w niektórych przypadkach zredukować wielkość wynikowego zbioru reguł, opisując złożone zależności w danych w łatwy do interpretacji sposób. Ostatnia z metod DeepRules umożliwia generowanie reguł zawierających wszystkie rodzaje warunków wprowadzonych w ramach poprzednich dwóch. Dodatkowo uzyskane z jej pomocą reguły mogą zawierać zagnieżdżone formuły logiczne, wpisując się tym samym w trzeci spośród celów szczegółowych pracy. Wszystko to, w przeciwieństwie do metody MofNRules, dokonywane jest w sposób bardziej wydajny obliczeniowo, bez konieczności dwukrotnego generowania zbiorów reguł. W odróżnieniu od metod opartych na gradientowym uczeniu, wszystkie proponowane algorytmy mają charakter heurystyczny i bazują na strategii sekwencyjnego pokrywania, nie wymagając stosowania różniczkowalnych funkcji straty.

Warto zauważyć, że opisywane prace nawiązują do koncepcji konstruktywej indukcji zaproponowanej po raz pierwszy przez Michalskiego [117]. Mimo swej popularności

w latach 90. technika ta jest stosunkowo rzadko wykorzystywana we współczesnych algorytmach uczenia maszynowego. Stało się tak częściowo ze względu na ograniczenia ówczesnej mocy obliczeniowej systemów komputerowych. Można tutaj dostrzec pewną analogię do historii rozwoju sztucznych sieci neuronowych, które po okresie relatywnego zastoju (tzw. „zimy AI" w latach 90.) przeżywają obecnie renesans za sprawą rozwoju technik uczenia głębokiego [104] oraz dzięki dynamicznemu rozwojowi sprzętu obliczeniowego [97].

3 Układ pracy

Niniejsza praca składa się z dziesięciu rozdziałów. W rozdziale 2 wymieniono i opisano różne rodzaje reguł decyzyjnych, wraz z przykładami. Dla każdego z nich podano algorytmy, które je wykorzystują.

Kolejny rozdział 3 został poświęcony przeglądowi różnych form warunków elementarnych stosowanych w przesłankach reguł decyzyjnych. Omówiono w nim wpływ użycia poszczególnych z nich na zdolność reprezentacji wiedzy. Uwaga zostanie skupiona na warunkach występujących w regułach ostrych, wykorzystujących logikę klasyczną.

Rozdział 4 zawiera krótki przegląd istniejących w literaturze podejść i metodologii indukcji reguł decyzyjnych. Omówiono w nim także, w jaki sposób dokonywana jest predykcja z użyciem modeli regułowych.

W rozdziale 5 zostało wprowadzone pojęcie głębokiego dyskretnego uczenia. Omówiona została jego geneza oraz w jaki sposób jest ono rozumiane w niniejszej pracy. Podano także przykłady niewielu istniejących już metod stosujących te koncepcje uczenia.

Następny rozdział 6 zawiera obszerną analizę trzech opracowanych przez autora metod indukcji reguł, nawiązującą do wprowadzonego wcześniej pojęcia głębokiego dyskretnego uczenia. Działanie każdej z nich zostało dokładnie opisane oraz omówione na przykładach. Rozdział ten zawiera również analizę złożoności każdej z metod.

Rozdział 7 został poświęcony badaniom eksperymentalnym zaproponowanych wcześniej algorytmów indukcji reguł. Zawiera on opis metodologii, użytych wartości parametrów oraz obszerną analizę wyników. Realizuje on jednocześnie trzeci cel szczegółowy określony w tejże pracy.

W rozdziale 8 omówione zostały wybrane studia przypadków, na przykładzie których przedstawiono i porównano działanie opracowanych przez autora algoryt-

mów. Dotyczą one zarówno zbiorów syntetycznych o znanych występujących w nich zależnościach, jak i rzeczywistych danych.

W rozdziale 9 opisano wdrożenie wyników opisanych prac badawczych w produkcie podmiotu zatrudniającego jej autora. Ten rozdział zawiera również uzasadnienie, w jaki sposób uatrakcyjniło to ów produkt i wyróżniło go na tle konkurencji.

Ostatni rozdział, 10, zamyka całą pracę. Zawiera on kluczowe wnioski oraz nakreśla możliwe kierunki dalszych badań.

2 Wybrane typy reguł decyzyjnych

W literaturze przedmiotu możemy spotkać się z różnymi typami reguł decyzyjnych. Każdy z nich charakteryzuje się unikalnymi właściwościami, determinującymi ich przydatność w określonych kontekstach. Niniejszy rozdział ma na celu przedstawienie przeglądu kluczowych rodzajów reprezentacji reguł decyzyjnych, ze szczególnym uwzględnieniem ich formalnych podstaw, właściwości oraz różnic pomiędzy nimi.

1 Reguły ostre

Reguły ostre (z ang. Crisp Rules) stanowią klasyczną formę reprezentacji wiedzy, opartą na logice dwuwartościowej. Ich przesłanki zbudowane są z warunków logicznych, które mogą być spełnione lub nie. Zbiory reguł ostrych są powszechnie reprezentowane w dysjunkcyjnej postaci normalnej (DNF) lub koniunkcyjnej postaci normalnej (CNF). W zbiorze reguł w postaci DNF każda reguła składa się z koniunkcji warunków, a poszczególne reguły połączone są operatorem logicznej alternatywy. W postaci CNF natomiast przesłanki reguł są alternatywą warunków, a same reguły łączone są operatorem koniunkcji. W literaturze funkcjonują także pojęcia reguły charakterystycznej (z ang. characteristic rule) i dyskryminacyjnej (z ang. discriminative rule). Pierwsze z nich oznacza regułę, której przesłanka jest koniunkcją warunków, drugie dotyczy takiej, gdzie przyjmuje ona formę alternatywy warunków [5].

Przykład dwóch równoważnych zbiorów reguł klasyfikacyjnych, dla obu postaci CNF i DNF, znajduje się w tabeli 2.1.

#	DNF	CNF
r1:	IF $c_1 \wedge c_2$ THEN $\delta = 1$	IF $c_1 \vee c_3$ THEN $\delta = 1$
r2:	IF $c_3 \wedge c_4$ THEN $\delta = 1$	IF $c_1 \vee c_4$ THEN $\delta = 1$
r3:		IF $c_2 \vee c_3$ THEN $\delta = 1$
r4:		IF $c_2 \vee c_4$ THEN $\delta = 1$

Tabela 2.1: Przykładowe zbiór reguł ostrych opisujących wyrażenie $(c_1 \wedge c_2) \vee (c_3 \wedge c_4)$

W przypadku prezentowanego zbioru reguł w postaci DNF przykład jest klasyfikowany jako pozytywny, wtedy gdy zostanie pokryty przez którąkolwiek z reguł. Dla zbioru w postaci CNF stanie się tak wtedy, gdy zostanie on pokryty przez wszystkie reguły.

Większość popularnych algorytmów indukcji reguł generuje zbiory reguł dyskryminacyjnych w postaci DNF. Klasycznymi przykładami są tutaj: AQ17 [20], Ripper [37], CN2 [36] czy też Foil [143]. Do bardziej współczesnych metod indukcji reguł w postaci DNF zaliczyć można oparty o zachłanne przeszukiwanie algorytm RuleKit [74], metodę EXPLORE wykorzystującą wyczerpujące przeszukiwanie [147], generujący lokalnie optymalne reguły algorytm LORD [86] czy też stosującą optymalizację z użyciem spadku wzdłuż gradientu Relational Rule Network (R2N) [98].

Istnieje stosunkowo niewiele metod indukcji zbiorów reguł w postaci CNF. Mooney (1995) proponował modyfikację algorytmu Foil [143] PFOIL-CNF, umożliwiającą uzyskiwanie tego typu reguł [123]. Dries et al. opisywali metodę k-CNF [52], wykorzystującą w tym celu algorytm eliminacji kandydatów (z ang. Candidate Eliminarion Algorithm [122]). Beck et al. proponowali rozwiązanie wykorzystujące wariant algorytmu LORD nazwany $\overline{LORD}$ [11]. Proponowana przez Dash et al. metoda Boolean Decision Rules via Column Generation również umożliwiała generowanie zbiorów reguł w postaci CNF [46].

W niniejszej pracy uwaga została skupiona na regułach ostrych (Crisp). Wszelkie proponowane w niej metody i algorytmy dotyczą jedynie tej wybranej kategorii reguł decyzyjnych.

2 Reguły rozmyte

Reguły tego typu wykorzystują logikę rozmytą do radzenia sobie z niepewnością i dwuznacznością w danych. Są one szczególnie przydatne w systemach, w których granice pomiędzy pewnymi klasami lub kategoriami nie są dobrze zdefiniowane [58].

Zadeh, w swej pracy wprowadzającej teorie zbiorów rozmytych, jako przykład podaje klasę wysokiego mężczyzny [195]. Trudne jest jednoznaczne określenie progu wzrostu, powyżej którego można uznać daną osobę za wysoką. Mimo to, porównując dwie osoby, łatwo określić, której z nich bliżej jest do bycia wysokim mężczyzną.

W regułach ostrych, gdzie jest wykorzystywana klasyczna logika dwuwartościowa, przesłanka reguły może być jedynie prawdziwa lub fałszywa. W przypadku reguł rozmytych przesłanka, a za tym również konkluzja reguły, może być prawdziwa jedynie w pewnym stopniu. Reguły rozmyte zamiast warunków reprezentujących zbiory ostre, wykorzystują tak zwane zbiory rozmyte. Zbiór rozmyty A, zawierający elementy $x \in U$, jest zdefiniowany poprzez funkcję przynależności $m(x)$, która każdemu elementowi x przypisuje liczbę rzeczywistą z zakresu 0–1. Wartość 0 oznacza tutaj całkowity brak (całkowity fałsz) a 1 całkowitą przynależność (całkowitą prawdę) [115].

Przykładowa reguła rozmyta znajduje się poniżej.

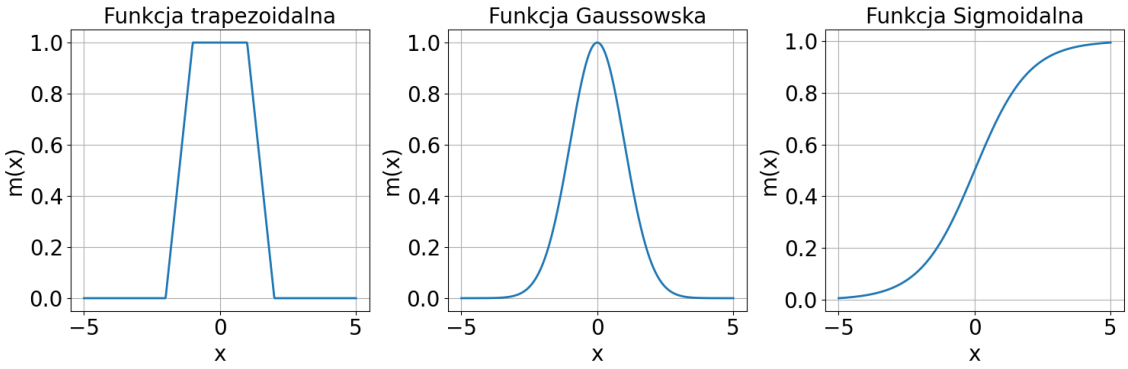
JEŻELI prędkość = *wysoka* **TO** *ryzyko* = *wysokie*

Zakładamy tutaj, że zarówno prędkość, jak i ryzyko są atrybutami numerycznymi. W przypadku reguły ostrej, do określenia tego, czy prędkość oraz ryzyko są wysokie, użyte zostałyby pewne wartości progowe. Wszystkie wartości powyżej tych progów uznane byłyby za wartości wysokie. Takie podejście może nie być odpowiednie w tym wypadku, jako że ryzyko może rosnąć w sposób ciągły wraz ze wzrostem prędkości. W przypadku powyższej reguły rozmytej dla każdego przykładu określana jest prawdziwość przesłanki z wykorzystaniem jej funkcji przynależności. Następnie na podstawie jej wartości określana jest prawdziwość konkluzji. [53].

Istnieje wiele rodzajów funkcji przynależności, wykorzystywanych w regułach rozmytych. Popularnie wykorzystywanymi funkcjami są tutaj funkcja trapezoidalna, Gaussowska czy też sigmoidalna (rysunek 2.1) [138].

W regułach rozmytych, z racji wykorzystania logiki rozmytej, nie stosuje się logicznych operatorów koniunkcji i alternatywy do łączenia warunków w przesłance. Zamiast tego wykorzystywane są operatory logiki rozmytej. Operator koniunkcji logicznej zastępuje się funkcją zwaną T-normą, alternatywa zaś S-normą (zwaną również T-konormą). Przykładem funkcji T-normy jest funkcja minimum, a S-normy maksimum [199]. W warunkach skrajnych (dla wartości 0 i 1) są one tożsame z operatorami alternatywy i koniunkcji logicznej.

W literaturze można spotkać wiele metod indukcji reguł decyzyjnych wykorzystujących logikę rozmytą. Przykładowo metoda Wang-Mendel'a dokonuje podziału



Rysunek 2.1: Przykładowe funkcje przynależności zbiorów rozmytych

danych treningowych na rozmyte obszary. Następnie na podstawie relacji między nimi generuje zbiory reguł [179]. Odmienne podejście, z wykorzystaniem metod klasteryzacji proponował w swej metodzie Bezdek [17]. Do indukcji reguły rozmytych wykorzystywano również algorytmy genetyczne [41]. Nomura et al. proponowali także rozwiązanie wykorzystujące metodę spadku wzdłuż gradientu [132]. Opisywana przez Hühn i Hüllermeier metoda FURIA bazuje z kolei na algorytmie RIPPER, dostosowując go do indukcji reguł rozmytych. W przeciwieństwie do RIPPER, gdzie generowane są listy reguł o ustalonym porządku, tutaj wynikowe zbiory reguł są nieuporządkowane, a ich kolejność nie ma znaczenia. Po utworzeniu początkowego zbioru reguł FURIA zamienia ostre przedziały w regułach na zbiory rozmyte, zachłannie optymalizując kryterium jakości [87].

3 Reguły asocjacyjne

Reguły asocjacyjne stanowią specjalny rodzaj reguł decyzyjnych. Służą identyfikowaniu pewnych współwystępujących ze sobą elementów w zbiorze danych. Mają one postać implikacji $X \rightarrow Y$, gdzie X i Y są zbiorami rozłącznymi. [1]. Wskazują one na to, że jeżeli elementy zbioru X pojawią się w transakcji, to prawdopodobnie pojawią się w niej również elementy zbioru Y. Przykład reguły asocjacyjnej znajduje się poniżej.

JEŻELI $\{kawa, cukier\}$ **TO** $\{mleko\}$

Reguła ta wskazuje, że jeżeli klient kupuje kawę i cukier, to istnieje duże prawdopodobieństwo, że kupi również mleko.

Generowanie reguł asocjacyjnych odbywa się zwykle na podstawie wyszukanych w danych zbiorów częstych. Zbiorem częstym nazywamy zbiór elementów, pojawiający się minimalnie w pewnej określonej liczbie transakcji [1]. Liczba ta określana jest poprzez minimalne wsparcie zbioru.

Większość metod indukcji reguł asocjacyjnych generuje je z wykorzystaniem klasycznej logiki dwuwartościowej. Popularną metodą jest tutaj zaproponowany przez Rakesh i Ramakrishnan algorytm Apriori [2]. Metoda ta rozpoczyna od znalezienia wszystkich jednoelementowych zbiorów, występujących w co najmniej N procentach transakcji, gdzie N jest parametrem algorytmu. Następnie generuje on z elementów pary, a następnie z par trójki itd. aż do osiągnięcia maksymalnego rozmiaru zadanego parametrem. Alternatywą dla Apriori jest algorytm FP-Growth, zaproponowany przez Han et al. [79]. Jest on wydajnym algorytmem wyszukiwania zbiorów częstych, mogącym służyć do indukcji reguł asocjacyjnych. Wykorzystuje on do reprezentacji danych tak zwane FP-drzewa, pozwalające w szybki i wydajny sposób wyszukiwać zbiory częste. Innym algorytmem znajdującym zastosowanie w indukcji reguł asocjacyjnych jest metoda LCM [174]. Pozwala ona wyszukiwać zbiory częste z liniową złożonością, wykorzystując do kompresji danych tablice zamiast drzew. Kolejną wydajną metodą indukcji reguł asocjacyjnych jest algorytm OPUS [181]. Wykorzystuje ona drzewa do reprezentacji przestrzeni poszukiwań, zawężając ją poprzez iteracyjne odcinanie poszczególnych gałęzi, niezawierających zadowalająco dobrych rozwiązań.

Metody indukcji rozmytych reguł asocjacyjnych są obecne w literaturze, choć mniej popularne. Logika rozmyta jest wykorzystywana tutaj głównie w celu lepszej obsługi atrybutów numerycznych oraz uniknięcia ostrych granic w warunkach reguł. Wśród istniejących metod można wymienić między innymi algorytm F-APACS [32]. Wykorzystuje on terminy lingwistyczne, oparte na zbiorach rozmytych, do opisywania regularności wykrytych w danych. W przeciwieństwie do wielu innych metod, dzięki zastosowaniu analizy różnic korygowanych, nie ma tutaj potrzeby określania progów dla wsparcia i ufności poszukiwanych zbiorów. Pierrard at al. proponowali w swojej pracy modyfikację algorytmu Close, operującą na zbiorach rozmytych [135]. Metoda wykorzystywała zmodyfikowany operator konsekwencji. Pozwala to na redukcję przestrzeni przeszukiwania w porównaniu do klasycznych algorytmów, takich jak Apriori. Inną metodą jest Fuzzy ARM umożliwiająca wydajne generowanie rozmytych reguł asocjacyjnych dla dużych zbiorów danych [108]. Zbiór danych jest dzielony w niej na partycje, które następnie są kolejno przetwarzane w celu wyszukiwania zbiorów częstych. Do ich wydajnego śledzenia i przechowywania wykorzystywana jest spe-

cjalna struktura danych zwana tidlistą. Metoda wykorzystuje algorytm fuzzy c-means (FCM) w początkowej fazie do „rozmycia" danych wejściowych. Proponowana przez Dhopte i Tarapore metoda CARM wykorzystuje odmienne podejście. W pierwszym kroku generuje ona pulę reguł z wykorzystaniem algorytmów genetycznych, następnie dokonuje rozmycia w celu uniknięcia ostrych granic w warunkach dotyczących atrybutów numerycznych. Następnie z wykorzystaniem reguł rozmytych tworzone są reguły asocjacyjne poprzez zastosowanie zmodyfikowanego algorytmu Apriori. [169]

4 Reguły akcji

Reguły akcji są specjalną postacią reguł, opisującą, jakie działania należy podjąć, aby obiekt przyporządkowany do jednej ustalonej klasy zmienił przyporządkowanie do innej klasy [146]. Akcje rozumiane są tutaj jako zmiany wartości atrybutów, gdzie zmiana stanu oznacza zmianę wartości atrybutu decyzyjnego. Przykładowa reguła akcji znajduje się poniżej.

JEŻELI (palenie = {tak}) → (palenie = {nie}) ∧ (aktywność = {nie}) → (aktywność = {tak})
TO (zagrożenie zdrowia = {wysokie}) → (zagrożenie zdrowia = {niskie})

Jedna z pierwszych metod indukcji reguł akcji została opisana przez Rasia i Wieczorkowską [146]. Dzieliła ona atrybuty występujące w zbiorze danych na atrybuty zmienne i stałe. W pierwszym kroku generowany był zbiór reguł klasyfikacyjnych z wykorzystaniem drzew decyzyjnych. Następnie, porównując wartości atrybutów zmiennych oraz klasy decyzyjnej, tworzone były na ich podstawie reguły akcji. Metoda DEAR generowała reguły akcji na podstawie par reguł asocjacyjnych, wyekstrahowanych z bazy danych [172]. Algorytm SCARI wykorzystywał z kolei w tym celu strategie sekwencyjnego pokrywania [163]. Nie wymagał on wcześniejszego kroku generowania reguł klasyfikacyjnych, lecz budował on reguły akcji od razu, korzystając z metody zachłannego przeszukiwania. Wspomniane wcześniej metody skupiają się na generowaniu reguł akcji z ostrymi warunkami w przesłance. W literaturze metody indukcji reguł akcji, wykorzystujących logikę rozmytą, są praktycznie nieobecne. Wymienić należy tutaj na pewno GA-FARM, który wykorzystuje zbiory rozmyte jako alternatywę dla popularnie stosowanej dyskretyzacji wartości atrybutów numerycznych [60]. Najbardziej efektywne kosztowo reguły akcji wyszukiwane są z użyciem algorytmów genetycznych. Wynikowe reguły zawierają tak zwane rozmyte akcje, gdzie zmiana wartości atrybutu numerycznego następuje w pewnym kierunku i z pewną

granularnością. Wspomniana praca jest jednak zgodnie z opinią jej autorów, oraz najlepszą wiedzą autora tejże pracy, pierwszą, która wprowadza pojęcie rozmytych reguł akcji.

5 Reguły prototypów

W ramach niniejszego przeglądu należy wspomnieć także o szczególnym rodzaju reguł decyzyjnych, zwanych regułami prototypów. Reguły te oceniają podobieństwo danego przykładu do pewnego określonego zestawu wzorców, zwanego prototypami. Zbiór prototypów P tworzony jest na bazie wybranych przykładów uczących. Stanowią one jednak ich uogólnienie i nie muszą specyfikować wartości wszystkich atrybutów. Wybór prototypów sprowadza się więc do redukcji zbioru danych do kilku reprezentatywnych przykładów, które następnie są optymalizowane pod kątem funkcji odległości i istotnych cech, tak aby zapewnić jak najdokładniejszą i najbardziej zrozumiałą klasyfikację [55]. Reguły prototypów pojawiają się w literaturze głównie w kontekście klasyfikacji, gdzie każdy z prototypów przypisany jest do pewnej klasy decyzyjnej. Podczas klasyfikacji przykładu najpierw wyszukiwany jest najbliższy mu pod kątem pewnej miary prototyp. Klasa, do której przynależy prototyp, brana jest następnie jako wartość predykcji. Do oceny tego, jak bardzo dany przykład różni się od prototypu, służy specjalna funkcja $D(x, p)$ gdzie $p \in P$ zwana miarą odmienności (z ang. dissimilarity function). Najczęściej jest ona funkcją dystansu [55]. Funkcja ta wykazuje pewną analogię do funkcji przynależności zbiorów rozmytych.

Poniżej przedstawiony został przykład reguł prototypów dla popularnego w uczeniu maszynowym zbioru danych opisującego kwiaty irysa [175]. Zbiór zawiera przykłady należące do trzech możliwych klas (gatunków kwiatu): Irys Setosa, Irys Versicolor i Irys Virginica. Każdy przykład opisywany jest przez trzy atrybuty numeryczne: szerokość i długość płatka oraz szerokość i długość działki kielicha. Jako miara odmienności została wykorzystana odległość Euklidesowa. Przykładowe prototypy dla poszczególnych klas przedstawione zostały poniżej.

- **Irys Setosa**: p_1 = {długość płatka = 1.4, szerokość płatka = 0.2)
- **Irys Versicolor**: p_2 = {długość płatka = 4.5, szerokość płatka = 1.3}
- **Irys Virginica**: p_3 = {długość płatka = 5.8, szerokość płatka = 2.0}

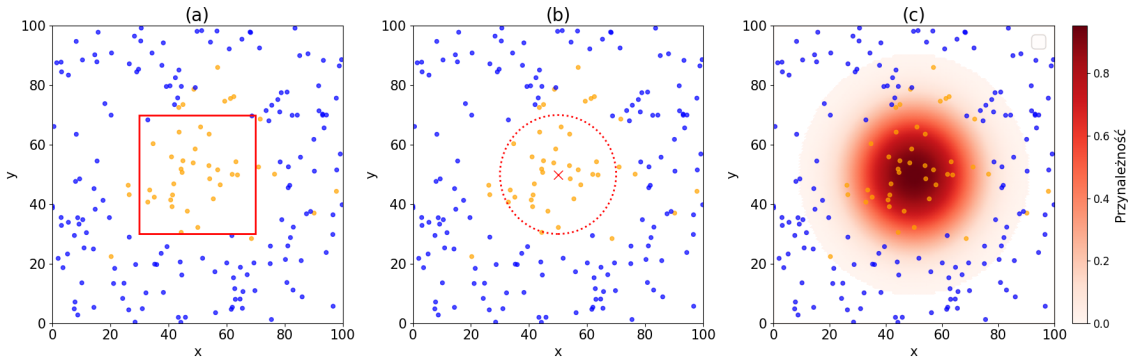
Dla tak wybranego zestawu prototypów utworzyć można następujące reguły:

JEŻELI $argmin(D(x,p_1),\ D(x,p_2),\ D(x,p_3)) = D(x,p_1)$ **TO** $gatunek =$ **Irys Setosa**
JEŻELI $argmin(D(x,p_1),\ D(x,p_2),\ D(x,p_3)) = D(x,p_2)$ **TO** $gatunek =$ **Irys Versicolor**
JEŻELI $argmin(D(x,p_1),\ D(x,p_2),\ D(x,p_3)) = D(x,p_3)$ **TO** $gatunek =$ **Irys Virginica**

W literaturze podejścia wykorzystujące reguły prototypów pojawiają się rzadziej niż te, dotyczące reguł ostrych i rozmytych. Duch i Grudziński [56] proponowali w swej pracy prostą metodę indukcji reguł prototypów. Na początku jako zbiór prototypów wybierała ona wszystkie przykłady zbioru uczącego. Następnie ze zbioru są iteracyjnie usuwane kolejne przykłady, badając przy tym ewentualny spadek dokładności klasyfikacji (accuracy). Parametry algorytmu pozwalają regulować relacje pomiędzy liczbą prototypów a jakością predykcji, zatrzymując proces w odpowiednim momencie [55]. Blachnik i Duch w swojej późniejszej pracy proponowali metodę indukcji reguł prototypów z wykorzystaniem maszyn wektorów nośnych (SVM) [19]. Zbiór prototypów wybierany jest sposród wyznaczonych przez SVM wektorów nośnych. Do jego wyboru wykorzystywany jest algorytm k najbliższych sąsiadów [62] lub innych na nim oparty. Dla każdego prototypu jest wyznaczona i przypisywana waga, w celu odtworzenia oryginalnych granic decyzyjnych wykorzystywanych przez SVM. Podobne podejście opisane zostało wcześniej przez Li et al. [106]. Proponowana przez nich metoda również budowała zbiór prototypów w oparciu o wektory nośne, wyznaczone z wytrenowanych SVM. Zbiór ten był następnie redukowany z użyciem opisanej przez Wilsona metody Drop2 [185]. Dla każdego z potencjalnych prototypów algorytm sprawdza, czy jego usunięcie nie pogorszy klasyfikacji pozostałych przykładów.

3 Typy warunków elementarnych występujących w regułach decyzyjnych

Przesłanki reguł decyzyjnych mogą składać się z dowolnych rodzajów warunków logicznych (reguły ostre) czy też zbiorów rozmytych (reguły rozmyte). Ich wybór determinuje nie tylko zdolność do wykrywania specyficznych relacji, ale także wpływa na interpretowalność i złożoność generowanych reguł. Przykładowo, warunki ostre, reprezentowane przez hiperprostokąty w przestrzeni cech, oferują prostotę i jasność, lecz mogą być niewystarczające w przypadku danych o skomplikowanych granicach decyzyjnych. Z kolei zbiory rozmyte, choć bardziej elastyczne, wprowadzają dodatkową warstwę abstrakcji, zmniejszając interpretowalność opisu. W przypadku reguł prototypów, warunki w przesłankach zastąpione są tak zwanymi prototypami. Dla każdego klasyfikowanego przykładu wybierany jest prototyp najbliższy mu pod kątem pewnej miary odległości. Możliwe jest tutaj osiągnięcie nieliniowych granic decyzyjnych dla pojedynczej reguły. Interpretacja reguł może jednak nie być oczywista, szczególnie w przypadku użycia bardziej skomplikowanych funkcji odległości.



Rysunek 3.1: Reprezentacja graficzna warunków dla reguł logiki klasycznej (a), rozmytej (c) oraz reguł prototypów (b)

Rysunek 3.1 prezentuje graficzną reprezentację przykładowych warunków dla różnych rodzajów reguł decyzyjnych. Lewy wykres dotyczy reguł ostrych, prawy rozmytych, a środkowy reguł prototypów. Rysunek ten obrazuje różnice w sposobie, w jaki opisują one dane. Osie x i y odpowiadają dwóm atrybutom numerycznym zbioru danych. Kolorem niebieskim oznaczone zostały przykłady negatywne, a pomarańczowym pozytywne. Wykres (a) odpowiada tutaj regule:

$$\textbf{JEŻELI } x \in [20, 70] \ \wedge \ y \in [30, 70] \textbf{ TO } \text{klasa} = \{\text{pozytywna}\}$$

Obszar pokrywanych przez nich przykładów reprezentowany jest przez czerwony prostokąt. Środkowy wykres (b) wizualizuje przykładową regułę prototypu, gdzie prototypem jest punkt $p = (x = 50, y = 50)$. Okrąg wyrysowany przerywaną linią określa tutaj zbiór przykładów, dla których miara odmienności $D(x, p)$ (w tym wypadku odległość euklidesowa) od prototypu jest mniejsza niż 20. Ostatni wykres (c) wizualizuje przykładową regułę rozmytą, wykorzystującą gausowską funkcję przynależności. Jasność koloru czerwonego określa tutaj wartość funkcji przynależności — im ciemniejszy, tym ona większa. Przedstawiony na wykresie (c) rozmyty okrąg można by także przedstawić w trzech wymiarach, gdzie oś z reprezentowałaby przynależność. W takiej przestrzeni przybrałby on postać trójwymiarowego dzwona funkcji gausowskiej.

W niniejszym rozdziale dokonany zostanie przegląd różnorodnych warunków stosowanych w regułach decyzyjnych, z uwzględnieniem ich wpływu na zdolność reprezentacji wiedzy. Uwaga zostanie skupiona na warunkach występujących w regułach ostrych, bazujących na logice klasycznej.

1 Warunki proste

W rozdziale 1 zostało wprowadzone pojęcie warunków prostych. Przyjmują one postać $a \odot v$, gdzie v jest dyskretną wartością atrybutu a, a $\odot$ operatorem relacji. Dla atrybutów numerycznych dostępnymi operatorami relacji są: $<$, $\leqslant$, $>$, $\geqslant$. Dla atrybutów nominalnych jest to zawsze operator równości. Warunek taki jest prawdziwy wtedy, gdy ów atrybut przyjmuje określoną przez warunek wartość. Warunki proste dla danych numerycznych definiują pewne graniczne wartości, poniżej lub powyżej których uznawany jest on za spełniony. W rezultacie definiują one hiperpłaszczyzny prostopadłe do osi odpowiadającej atrybutowi a. W literaturze obecna jest także forma warunku prostego dla atrybutów numerycznych, określająca jednocześnie

górną i dolną wartość brzegową, definiując tym samym przedział wartości w postaci $a \in [v_1, v_2]$, gdzie v_1 i v_2 należą do $\mathbb{R}$ oraz $v_1 < v_2$. Warunek taki można również zapisać w postaci koniunkcji logicznej dwóch warunków: $a \geqslant v_1$ i $a \leqslant v_2$. W ramach niniejszej pracy ten rodzaj warunków prostych nazwany zostanie warunkiem przedziału.

Warunki proste, z racji intuicyjnej interpretacji i prostoty, są najpopularniejszym rodzajem warunków wykorzystywanych w przesłankach reguł ostrych. Reguły, zawierające takie warunki, generowane są przez większość istniejących metod indukcji reguł, takich jak RIPPER [37], CN2 [36], C4.5 [152], LORD [86], RuleKit [74], Boolean Decision Rules via Column Generation [46], R2N [98] i wiele innych.

2 Negacje

W literaturze znaleźć można również metody używające negacji wyżej wspomnianych warunków prostych. W przypadku atrybutów numerycznych warunek taki przyjmuje postać $a \neq v_i$. Może on zostać zapisany w postaci alternatywy warunków prostych $\bigvee_{v \in (U \setminus \{v_i\})} a = v$, gdzie U oznacza uniwersum wartości atrybutu a. Alternatywa taka składa się z $|U|-1$ literałów. Warto zauważyć w tym miejscu, że zapis z wykorzystaniem negacji pozwala zapisać taką zależność w znacznie bardziej zwięzły i czytelny sposób, szczególnie dla większych rozmiarów U. Dla atrybutów numerycznych negacje warunków prostych mają sens jedynie w przypadku warunków przedziałów. Dla warunków z jedną granicą oznaczają one jedynie zmianę operatora $\odot$, nie zmieniając samej interpretacji warunku. Zanegowany warunek przedziału $a \notin \langle v_1, v_2 \rangle$ traktowany może być jako alternatywa dwóch warunków $a < v_1$ i $a > v_2$. Zastosowanie negacji pozwala tutaj na efektywne opisanie relacji za pomocą jednego prostego warunku, podczas gdy bez niej konieczne byłoby użycie dwóch warunków prostych.

Wśród najbardziej popularnych metod indukcji reguł, zawierających tego typu warunki, wymienić można między innymi algorytm C2N oparty na sekwencyjnym pokrywaniu [36]. Są one również wspierane w wersji drugiej algorytmu RuleKit [76]. Warunki tego typu są wykorzystywane także w metodzie Olex, proponowanej przez Rullo et al [150]. Generowane przez nią reguły zawierały zawsze jeden warunek niezanegowany i jeden lub więcej warunków zanegowanych. Negacje warunków prostych były także jednym z wielu rodzajów warunków stosowanych przez algorytm AQ [117].

3 Wewnętrzne alternatywy

Warunki tego typu są alternatywą dwóch lub większej liczby warunków prostych lub złożonych. Ten rodzaj warunków omawiany może być jedynie w kontekście reguł, gdzie przesłanka ma postać koniunkcji (zbiory reguł w postaci DNF). Przykład reguły z takim warunkiem znajduje się poniżej (warunek ten został podkreślony).

JEŻELI pleć $= \{kobieta\} \wedge \underline{(\text{wiek} \in [35, 50) \vee \text{wiek} > 65)}$ **TO** ryzyko $= \{wysokie\}$

Specyficznym przypadkiem warunku wewnętrznej alternatywy jest warunek, gdzie wszystkie warunki składowe są warunkami prostymi, opartymi na tym samym atrybucie. Dla atrybutu nominalnego warunek taki zapisać można z użyciem równania 3.1, gdzie $v_1, \ldots v_N$ to nominalne wartości atrybutu a_i:

$$a_i = v_1 \vee a_i = v_2 \vee \ldots \vee a \ = v_N \tag{3.1}$$

Wyrażenie takie można uprościć do postaci $a_i \in \{v_1, v_2, \ldots, v_N\}$. Na potrzeby niniejszej pracy warunki tego typu określane będą jako warunki zbioru dyskretnego.

Większość istniejących metod generuje reguły, w których przesłanki są koniunkcją [36,37,74,143], rzadziej alternatywą [92,123] warunków elementarnych. Stosunkowo niewiele badań dotyczyło indukcji reguł mogących zawierać tego typu wyrażenia logiczne w swych przesłankach. Jedną z pierwszych metod, która budowała podobne opisy danych, był algorytm STAGGER [153], wykorzystujący zachłanne przeszukiwanie z użyciem wiązki. Nowszym podejściem jest algorytm CORD, proponowany przez Beck et al. [11]. Jest on modyfikacją i rozszerzeniem algorytmu LORD, generującego lokalnie optymalne reguły [86].

Zdaniem autora istnieje tutaj pewna luka badawcza, którą usiłuje wypełnić poprzez swoje badania. W ramach nich zostały opracowane trzy metody umożliwiające generowanie reguł zawierających warunki wewnętrznych alternatyw w przesłankach (patrz rozdział 6). Dwie pierwsze spośród nich: ComplexConditions i MofNRules zostały już opisane w publikacji do czasopisma Machine Learning and Knowledge Extraction, zatytułowanej „Classification, Regression, and Survival Rule Induction with Complex and M-of-N Elementary Conditions". W przypadku ostatniej z metod o nazwie DeepRules została ona po raz pierwszy zaprezentowana w ramach niniejszej pracy.

4 Warunki M-z-N (z ang. M-of-N)

Warunki typu M-z-N składają się z N warunków logicznych. W zależności od wybranej interpretacji są one prawdziwe, gdy dokładnie M, co najmniej M lub dokładnie M spośród N warunków składowych zostanie spełnione. W pierwszym wypadku mówimy o tak zwanych warunkach „przynajmniej M-z-N", w drugim o warunkach „dokładnie M-z-N", a w ostatnim o warunkach „co najwyżej M-z-N". Przykład warunku M-z-N znajduje się poniżej:

$$2\text{-}z\text{-}3(wiek > 35,\ plec = \{male\},\ zarobki = \{wysokie\}).$$

Tego typu hipotezę przedstawić można w postaci wyrażenia logicznego w dysjunkcyjnej postaci normalnej. Dla wariantu „przynajmniej M-z-N" oraz „dokładnie M-z-N" zawiera ona $\binom{N}{M}$ koniunkcji, gdzie każda z nich składa się z M literałów. Oznaczamy zbiór warunków wchodzących w skład warunku M-z-N jako $C' = \{c_1, c_2, \ldots, c_N\}$. Z wykorzystaniem tej notacji możemy zapisać warunek „co najmniej 2 z 3" w następujący sposób:

$$(c_1 \wedge c_2) \vee (c_1 \wedge c_3) \vee (c_2 \wedge c_3) \tag{3.2}$$

Jak można zauważyć, składniki występujące w wyrażeniu koniunkcji odpowiadają wszystkim M-elementowym kombinacjom zbioru C'.

Dla warunku „dokładnie M-z-N" dla $M = 2$ i $N = 3$ wyrażenie te przyjmuje formę:

$$(c_1 \wedge c_2 \wedge \neg c_3) \vee (c_1 \wedge c_3 \wedge \neg c_2) \vee (c_2 \wedge c_3 \wedge \neg c_1) \tag{3.3}$$

W tym wypadku również każda z koniunkcji odpowiada N-elementowej kombinacji zbioru C'. Zawiera ona jednak dodatkowo negację wszystkich pozostałych warunków.

Sytuacja ma się zgoła odmiennie w przypadku warunków co najwyżej M-z-N. O ile warunek taki wciąż rozpisać można w postaci alternatywy koniunkcji, to potrzebnych jest ich aż $\sum_{i=1}^{M} \binom{N}{i}$. Dla naszego przykładu uzyskamy więc następujący zapis:

$$\begin{aligned}(c_1 \wedge \neg c_2 \wedge \neg c_3) \vee (c_2 \wedge \neg c_1 \wedge \neg c_3) \vee (c_3 \wedge \neg c_1 \wedge \neg c_2) \vee \\ (c_1 \wedge c_2 \wedge \neg c_3) \vee (c_1 \wedge c_3 \wedge \neg c_2) \vee (c_2 \wedge c_3 \wedge \neg c_1)\end{aligned}$$

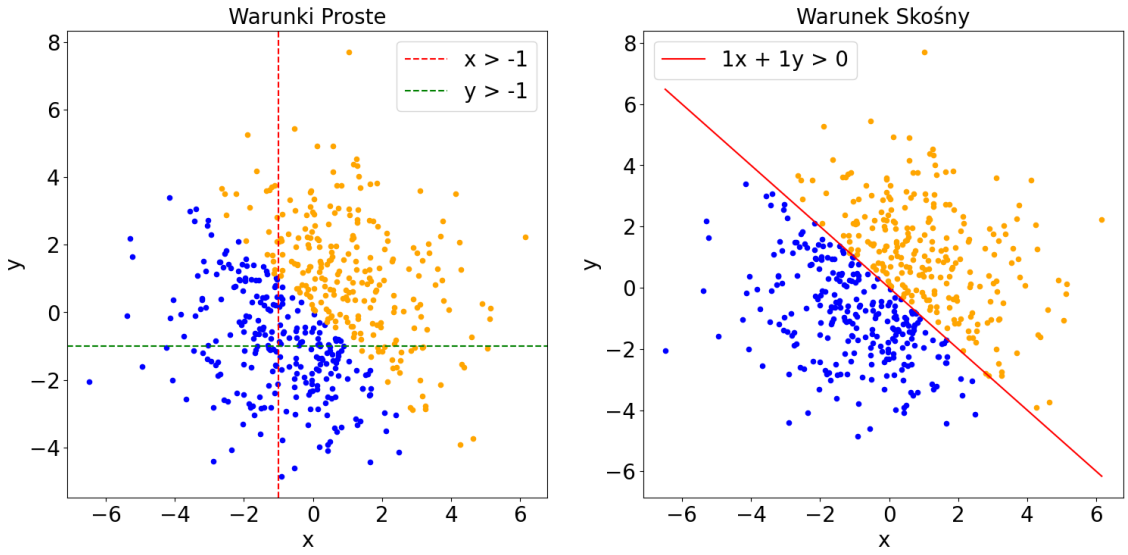
Jak można zauważyć na powyższym przykładzie, warunki M-z-N pozwalają w bardziej zwięzły sposób opisać złożone wyrażenia DNF, będąc przy tym łatwiejszymi w zrozumieniu i interpretacji.

Warunki tego typu były wykorzystywane w wielu metodach indukcji reguł na przestrzeni lat. Stosował je Bledorn i Michalski w swoim algorytmie AQ17-DCI [20] Pojawiały się także w metodzie EFC (Vouk et al.) [178]. Wariant dokładnie M-z-N stosowany był również m .in. w pracy Zheng et al. [197]. Warunki przynajmniej M-z-N wykorzystywane były z kolei w algorytmie ID2-of-3 [126]. Były one budowane poprzez inkrementacyjną rozbudowę o kolejne warunki składowe z wykorzystaniem strategii wspinaczki. W literaturze obecne są także metody ekstrakcji reguł z warunkami M-z-N z wytrenowanych sieci neuronowych [156], [170]. Fürnkranz i inni opisywali modyfikację algorytmu LORD, umożliwiającą uczenie się reguł zawierających tego typu warunki, z wykorzystaniem zachłannego przeszukiwania [9].

Większość istniejących w literaturze metod, z wyjątkiem AQ17-DCI i EFC, pozwala generować tylko i wyłącznie reguły, których przesłanka jest pojedynczym warunkiem M-z-N. Proponowana przez autora metoda indukcji reguł MofNRules [113], opisywana w rozdziale 6, jest jedną z nielicznych, pozwalającą łączyć warunki M-z-N z innymi rodzajami deskryptorów.

5 Warunki skośne

Warunki proste w postaci $a_i < x_i$ lub $a_i > x_i$ można przedstawić jako hiperprostokąty w wielowymiarowej przestrzeni cech. Mogą one dzielić przestrzeń cech jedynie w sposób prostopadły w stosunku do osi. Warunki skośne nie posiadają tego ograniczenia. Dzielą one przestrzeń z użyciem funkcji liniowych jednego lub więcej atrybutów o dowolnym nachyleniu. Fakt, że funkcje tego typu nie muszą definiować granic równoległych w stosunku do osi, pozwala często na opisanie relacji występujących w danych z mniejszym błędem i z pomocą mniejszej ilości warunków. Przykład takiego przypadku zaprezentowany jest na rysunku 3.2. Prezentuje on przykładowy zbiór dwuklasowy, gdzie kolorem pomarańczowym została oznaczona klasa pozytywna. Zbiór taki trudno opisać z użyciem warunków prostych (lewy wykres). Do osiągnięcia wysokiej dokładności byłaby potrzebna duża ich liczba. Znacznie łatwiej dokonać tego z użyciem pojedynczego warunku skośnego, który klasyfikuje dane z zerowym błędem (prawy wykres). Prezentowany przykład jest oczywiście przykładem skrajnym, lecz dobrze pokazuje, jak wykorzystanie różnych rodzajów warunków może wpływać na jakość wynikowych reguł oraz ich złożoność.



Rysunek 3.2: Przykładowy zbiór danych opisany z użyciem warunków prostych i jednego warunku skośnego

Warunek skośny bazujący na dwóch atrybutach zapisywany jest zwykle w następującej postaci:

$$A\ a_i + Ba_j \geqslant 0 \quad \text{gdzie } i \ \neq j$$

Gdzie A i B są pewnymi stałymi. Warunki skośne mogą bazować również na większej liczbie atrybutów niż dwa, definiując w ten sposób hiperpłaszczyzny w przestrzeni danych.

Metoda ADReD [145] umożliwia generowanie zbiorów reguł z warunkami skośnymi bezpośrednio z danych. Dzieli ona przykłady klasy pozytywnej i negatywnej na klastry. Dla każdego z nich była wyznaczana możliwie najmniejsza liczba par hiperpłaszczyzn, otaczająca klaster. Wyznaczone były one w taki sposób, by określana przez nie grupa przykładów zawierała jak najmniej przykładów klasy przeciwnej. W przeciwieństwie do wcześniej wymienianej metody, algorytm CHIRA [161] generuje reguły skośne na podstawie istniejących zbiorów reguł o cięciach hiperprostokątnych. Dokonuje tego w dwóch krokach: łączenia i agregacji reguł. Pierwszy z nich łączy warunki dwóch reguł w jeden hiperprostokąt. Drugi tworzy na jego podstawie nową regułę z warunkiem skośnym. Podejście takie pozwala nie tylko zmniejszyć liczbę reguł w wynikowym zbiorze reguł, ale także uzyskać bogatsze opisy danych. Zaproponowana przez Michalaka i współpracowników metoda bazowała na liniowej analizie dyskryminacyjnej (ang. linear discriminant analysis, LDA) do wyznaczania

hiperpłaszczyzn dzielących klasy decyzyjne [116]. Do generowania zbiorów reguł z warunkami skośnymi wykorzystywano także z sukcesem maszyny wektorów nośnych [6], [111], [19].

Inna grupa metod indukcji reguł z warunkami skośnymi skupia się na ich ekstrachowaniu z wytrenowanych sieci neuronowych. REANN [95] wykorzystuje w tym celu trzywarstwową sieć neuronową. Metoda automatycznie dobiera liczbę neuronów warstwy ukrytej, zaczynając od małej ich liczby i stopniowo zwiększając ją aż do uzyskania zadowalającej skuteczności modelu. Warunki skośne uzyskiwane są na podstawie wytrenowanych wag warstwy ukrytej. Inne podejście zostało zastosowane przez Saad et al. w metodzie HYPINV [151]. Reguły nie są budowane tutaj na podstawie struktury sieci i jej wag. Zamiast tego jest ona wykorzystywana jako czarna skrzynka i swego rodzaju „nauczyciel". Algorytm generuje zbiór reguł z warunkami skośnymi tak, by jak najlepiej aproksymować decyzje podejmowane przez wytrenowaną sieć.

Warto wspomnieć także o metodach indukcji drzew decyzyjnych z warunkami skośnymi, jako że w prosty sposób można z nich uzyskać zbiory reguł. Jedną z pierwszych tego typu metod był algorytm CART-LC [27]. Wykorzystywał on strategię wspinaczki do wyznaczania skośnych cięć dla każdego z węzłów drzewa. Metoda ta jednak była całkowicie deterministyczna i nie miała żadnego mechanizmu chroniącego przed utknięciem w lokalnym minimum. Problem ten adresowali Sreerama et al., opisując w swej pracy algorytm OC1 stosujący mechanizm randomizacji [127]. Wickramarachchia et al. proponowali także metodę HHCART, która szukała cięć skośnych z użyciem wektorów własnych [183]. Dla każdej z klas był wyznaczany dominujący wektor własny. Następnie przestrzeń danych była obracana tak, by wektor stał się równoległy do osi. Dzięki takiemu przekształceniu wyznaczone cięcia równoległe do osi odpowiadały cięciom skośnym w oryginalnej przestrzeni danych.

6 Warunki relacji atrybutów

Warunki takie określają pewną relację pomiędzy dwoma lub większą liczbą atrybutów w zbiorze danych. Spełnione są dla przykładów, dla których relacja pomiędzy tymi atrybutami jest prawdziwa. Dla dwóch atrybutów warunek taki ma następującą postać:

$$a_i \odot a_j \quad gdzie\ i\ \neq j \tag{3.4}$$

Symbol $\odot$ oznacza tutaj operator relacji. Najczęściej wykorzystywanymi relacjami są:

- Równości i nierówności ($a_i = a_j$ lub $a_i \neq a_j$),
- Większości i mniejszości ($a_i > a_j$) lub ($a_i < a_j$),
- Większy lub równy oraz mniejszy, lub równy ($a_i \geqslant a_j$) lub ($a_i \leqslant a_j$).

Z przypadku gdy a_i i a_j są numeryczne, warunek relacji atrybutów można traktować jako specjalny przypadek warunku skośnego dla $A = 1$ i $B = 1$ (patrz poprzednia sekcja). Mimo istnienia wielu metod wykorzystujących warunki skośne, stosunkowo niewiele wykorzystywało warunki relacji atrybutów. Przykładami takich metod są: AQ17-DCI (Bloedorn et al.) [20] oraz BACON (Bradshaw et al.) [24]. Obie opisywane są dokładniej w sekcji 4.

4 Istniejące podejścia do indukcji reguł

1 Sekwencyjnego pokrywanie

Pierwsze algorytmy indukcji reguł bazowały na pojęciu sekwencyjnego pokrywania wprowadzonego w swych pracach przez Michalskiego [119]. Metodyka ta pojawia się w literaturze także pod nazwą dziel i rządź z ang. separate-and-conquer. Polega ona na uczeniu się reguły, usunięciu ze zbioru treningowego przykładów przez nią pokrytych (dziel), a następnie rekurencyjnym generowaniu kolejnych reguł, pokrywających niepokryte dotąd przykłady (rządź). Proces taki odbywa się tak długo, aż zabraknie przykładów w zbiorze treningowym lub aż spełnione zostanie specjalnie określone kryterium stopu [64].

Algorytmy indukcji reguł działające według paradygmatu sekwencyjnego pokrywania podzielić można na dwie grupy, ze względu na sposób, w jaki uczone są poszczególne reguły [66]. Pierwsza grupa to tak zwane podejścia „od ogółu do szczegółu" (z ang. top-down). Zaczynają one od reguły możliwie najbardziej ogólnej, pokrywającej jak najwięcej przykładów, specjalizując ją następnie aż do osiągnięcia zadanego kryterium stopu. Druga grupa to podejścia „od szczegółu do ogółu" (z ang. bottom-up). W ich przypadku początkowa postać reguły jest jak najbardziej specyficzna, pokrywając możliwie małą liczbę przykładów. W zależności od metod jest to reguła pusta bądź opisująca jeden wybrany przykład. Następnie w trakcie procesu indukcji jest ona generalizowana aż do osiągnięcia określonego przez algorytm kryterium stopu [66].

Każde z tych dwóch podejść posiada pewne unikalne zalety i wady. Metody „od ogółu do szczegółu" zwykle prowadzą do uzyskania bardziej ogólnych reguł, pokrywających większą liczbę przykładów. Dokonywane przez nie ogólnienie mogą jednak być w niektórych sytuacjach zbyt duże. Podejścia te jednak lepiej radzą sobie z zakłóceniami w danych treningowych od tych należących do drugiej grupy.

Metody „od szczegółu do ogółu" lepiej radzą sobie z kolei w sytuacjach, gdzie zbiór treningowy jest niewielki, niezbalansowany oraz tam, gdzie jest on przetwarzany w sposób inkrementacyjny, przykład po przykładzie [66, 131].

Większość popularnych obecnie metod indukcji reguł stosujących sekwencyjne pokrywanie generuje reguły „od ogółu do szczegółu". Podejście „od szczegółu do ogółu" jest obecnie rzadziej spotykane, choć nie nieobecne. [66, 125]. Gilles proponował z swej pracy metodę SIA [177]. Generowała ona reguły, wybierając losowe przykłady i następnie generalizując je z wykorzystaniem algorytmów genetycznych. Na koniec procesu zbiór reguł poddawany był fltracji w celu ograniczenia jego wielkości. Wartym wspomnienia jest również algorytm BRACID [131]. Jego autorzy dowodzą, że strategia „od szczegółu do ogółu" lepiej radzi sobie w przypadku danych niezbalansowanych, gdzie niektóre z klas są mniej liczne od pozostałych. BRACID rozpoczyna się od reguły opisującej pojedynczy wybrany przykład. Początkowo każdy przykład uczący traktowany jest jako potencjalny kandydat do utworzenia reguły. Reguły są generalizowane maksymalizując kryterium jakości, aż do momentu, gdy nie sposób dokonać kolejnej ich generalizacji bez pogorszenia zdolności klasyfikacyjnych całego zbioru. W przypadku gdy dla kilku przykładów osiągnięta zostanie ta sama postać końcowa reguły, zbiór wynikowy zawierać będzie tylko jedną z nich. Muggleton et al. [125] w swojej metodzie ProGolem wykorzystywali koncepcje tak zwanego *lgg*, czyli najmniej ogólnego uogólnienia (z ang. least general generalisation) zaproponowaną przez Plotkin [137]. ProGolem wybiera reguły startowe na podstawie *lgg* losowo wybranych par przykładów pozytywnych. Następnie reguły te są zachłannie generalizowane, aż do momentu, gdy żadna kolejna generalizacja nie poprawia jakości reguły. Pozytywne przykłady pokryte przez regułę są następnie usuwane ze zbioru treningowego, a proces kontynuowany aż do pokrycia wszystkich przykładów.

Również algorytm AQ [118] będący pierwszym algorytmem indukcji reguł wykorzystującym sekwencyjne pokrywanie, realizował strategie „od szczegółu do ogółu". Algorytm ten budował zbiór reguł decyzyjnych poprzez wybranie pojedynczego, niepokrytego przykładu z klasy pozytywnej, tak zwanego ziarna. Następnie algorytm generował wszystkie możliwe reguły w formacie DNF, pokrywające zadane ziarno oraz niepokrywające żadnych przykładów negatywnych. Kolejno z wygenerowanego zestawu reguł była wybierana ta, która najlepiej spełniała zadane przez użytkownika kryterium jakości. Metoda ta, w swej pierwotnej formie, generowała reguły decyzyjne, mające pokrywać jak największą ilość przykładów pozytywnych, nie pokrywając przy

tym żadnych przykładów negatywnych. Jeżeli istniało kilka takich możliwych opisów, wybierany był ten najlepiej spełniający zadane kryterium jakości [117].

W późniejszych latach powstało wiele kolejnych wersji algorytmu AQ [21, 117, 118, 189]. Rozszerzały one jego możliwości algorytmu i eliminowały pewne jego ograniczenia. W wersji 15 pozwalał on już na indukcję reguł na podstawie danych zawierających przykłady niespójne, a więc takie, gdzie dla tych samych wartości atrybutów klasa decyzyjna różniła się [117]. Dodatkowo algorytm pozwalał na wprowadzenie pewnego startowego zestawu reguł, który następnie był rozszerzany przez algorytm.

Jednym z najbardziej popularnych algorytmów, opartym na sekwencyjnym pokrywaniu, jest Ripper (Repeated Incremental Pruning to Produce Error Reduction) [37]. Metoda ta rozpoczyna proces indukcji reguł od posortowania występujących w zbiorze treningowym klas decyzyjnych od najmniej do najbardziej licznej. Dla kolejnych tak posortowanych klas algorytm wyznacza reguły. Dzięki takiemu działaniu metoda ta dobrze sprawdza się dla zbiorów niezbalansowanych i zapobiega problemowi przetrenowania. Kolejnym algorytmem indukcji reguł, opartym o sekwencyjne pokrywanie, jest algorytm CN2 [36]. Budował on reguły podobnie jak Ripper w dwóch krokach: wzrostu i przycinania. W każdym z nich do oceny nowo utworzonych reguł była wykorzystywana inna miara jakości. Algorytm wyszukiwał najlepsze reguły w sposób zachłanny, za każdym razem dodając lub usuwając warunek maksymalizujący wartość miary.

Kolejnym algorytmem indukcji reguł wykorzystującym sekwencyjne pokrywanie jest RuleKit [74] [162]. Podobnie jak wcześniej wspomniany algorytm CN2 wykorzystuje on heurystyki, aby ograniczyć przestrzeń poszukiwań podczas wzrostu i przycinania reguły. W przeciwieństwie do wcześniej wspomnianych algorytmów, pozwala on jednak generować nie tylko zbiory reguł klasyfikacyjnych, ale także znajduje zastosowanie dla problemów regresji i przeżycia. Dodatkowo udostępnia on możliwość generowania zbiorów reguł z wykorzystaniem wprowadzonej przez użytkownika wiedzy eksperckiej. Wiedza taka może mieć nie tylko postać gotowych reguł, ale może także zawierać zbiór preferowanych lub zabronionych atrybutów i warunków, które zdaniem eksperta powinny, czy też nie, być zawarte w wynikowym zbiorze reguł. Algorytm udostępnia również szereg parametrów, pozwalających na sterowanie tym, w jaki sposób ma zostać wykorzystana wiedza ekspercka. Algorytm może nie tylko uzupełnić ją o automatycznie wygenerowane reguły, ale też zbudować całkowicie nowy zbiór reguł na podstawie wprowadzonych przez użytkownika informacji. Algorytm RuleKit rozpoczyna indukcję zbioru reguł od pustego zbioru

i iteracyjnie rozszerza go o kolejne reguły, aż do momentu pokrycia całego zbioru treningowego lub do momentu, gdy liczba pokrytych przykładów spadnie poniżej pewnego zadanego parametrem poziomu.

2 Konstruktywna indukcja

Wraz z kolejnymi wersjami algorytmu AQ Michalski wprowadził również pojęcie konstruktywnej indukcji. Technika ta polega na przekształcaniu przestrzeni testowanych hipotez w celu osiągnięcia jak najlepszych opisów danych [117]. Przekształcenie to polega zwykle na wprowadzeniu do zbioru danych nowych atrybutów. Docelowo zmiana ta ma pozwolić wybranemu algorytmowi uczenia maszynowego na uzyskanie lepszych wyników i bardziej efektywny proces uczenia. Z użyciem ograniczonego zestawu hipotez testowych przez algorytm rozwiązanie niektórych problemów może nie być możliwe do uzyskania. Konstruktywna indukcja stara się rozwiązać ten problem, wprowadzając do zbioru danych nowe atrybuty, mogące reprezentować bardziej złożone zależności między oryginalnymi cechami.

Michalski i Wnek [188] zaproponowali taksonomie metod konstruktywnej indukcji, dzieląc ją na cztery grupy. Są to kolejno:

- konstruktywna indukcja sterowana danymi (z ang. Hypothesis-Driven Constructive Induction HCI),
- konstruktywna indukcja sterowana hipotezami (z ang. Data-Driven Constructive Induction DCI),
- konstruktywna indukcja sterowana wiedzą (z ang. Knowledge-Driven Constructive Induction KCI),
- wielostrategiczna konstruktywna indukcja (z ang. Multistrategy Constructive Induction Systems MCI),

2.1 Konstruktywna indukcja sterowana danymi

Konstruktywna indukcja sterowana danymi modyfikuje przestrzeń atrybutów na podstawie zależności występujących bezpośrednio w zbiorze danych. Różne algorytmy realizowały tę strategię w odmienny sposób. Algorytm BACON dodawał nowe

atrybuty poprzez wyszukiwanie relacji między oryginalnymi atrybutami w zbiorze danych [24]. Dokonywał tego, testując na nich różne hipotezy. Aby ograniczyć przestrzeń takich poszukiwań, autorzy zastosowali szereg heurystyk, bazujących na współzależności cech. Algorytm ABACUS wyznaczał dla zbioru danych zestaw opisujących go równań, w sposób podobny do algorytmu BACON [61]. Następnie dla każdego równania był wyznaczany warunek logiczny określający, dla jakich przykładów zbioru treningowego znajduje ono zastosowanie. Inne podejście zostało zastosowane w algorytmie STAGGER. Nowe atrybuty dodawane są tam jako koniunkcje lub alternatywy warunków prostych typu $a = v$, gdzie v jest pewną dyskretną wartością a. Do znalezienia takich formuł logiczny algorytm wykorzystuje zachłanne przeszukiwanie z wykorzystaniem wiązki [153]. Metoda EFC wykorzystywała popularne metody wyjaśnialności, takie jak IME czy SHAP, w celu wyznaczenia grup często występujących ze sobą wartości atrybutów [178]. Nowe atrybuty były dodawane jedynie na podstawie uzyskanych we wcześniejszym kroku grup. Pozwalało to w efektywny sposób na ograniczenie przestrzeni poszukiwań, zmniejszając złożoność obliczeniową samego algorytmu. Do tworzenia nowych atrybutów na podstawie danych wykorzystywano również algorytmy genetyczne. Muñoz et al. używali ich do budowy nowych cech, trenując na nich następnie wybrany algorytm uczenia maszynowego. Wyniki uzyskiwane przez model były wykorzystywane do oceny danej populacji [130]. Dzięki zasadzie działania algorytmów genetycznych metoda samoistnie utrzymała stałą liczbę nowo utworzonych atrybutów, zapobiegając eksplozji cech. PLS2 z kolei tworzył nowe atrybuty, reprezentujące funkcje liniowe oryginalnych atrybutów. Algorytmy genetyczne były wykorzystywane do uczenia parametrów tychże funkcji [105].

2.2 Konstruktywna indukcja sterowana hipotezami

W konstruktywnej indukcji, sterowanej hipotezami, nowe atrybuty dodawane są w sposób iteracyjny. W każdej iteracji wyznaczane są hipotezy, w których następnie są wyszukiwane różnego rodzaju wzorce. Wykryte wzorce dodawane są jako atrybuty do zbioru danych i przekazywane ponownie do trenowania modelu w kolejnej iteracji [188]. W tego typu indukcji istotnym problemem jest wzrost wymiarowości danych wraz z kolejnymi iteracjami. Problem ten był adresowany na wiele sposobów przez różnych badaczy. Muggleton w metodzie Duce stosował zestaw sześciu operacji logicznych, przekształcających wygenerowane w danej iteracji reguły, tworząc z nich nowe [124]. Tak przekształcone reguły wykorzystywane były następnie jako

potencjalne dodatkowe atrybuty. Po każdej iteracji propozycje nowych cech były prezentowane użytkownikowi, który pełniąc rolę wyroczni, wybierał, które z nich powinny zostać użyte. Inne ciekawe podejście do problemu zostało wykorzystane w metodzie CITRE (Matheus et al.). Budowała ona nowe atrybuty poprzez trenowanie drzew decyzyjnych [114]. Jeżeli takie drzewo posiadało więcej niż jeden liść o tej samej wartości decyzji, dla każdego z nich wyznaczano opisujące go wyrażenie logiczne. Wyrażenia te oparte były na operatorze logicznej koniunkcji i negacji. Algorytm ogranicza pulę atrybutów, uzyskaną po każdej iteracji do pewnej określonej liczby, wykorzystując do ich oceny współczynnik wzmocnienia informacji. W metodzie ID2-of-3 [126] nowo tworzone atrybuty reprezentowały warunki przynajmniej M-z-N. Algorytm budował takie hipotezy w sposób iteracyjny z wykorzystaniem strategii wspinaczki. Na początku wybierany był pewien warunek startowy, do którego dodawany był kolejny, który maksymalizował miarę jakość warunku. Mimo nieco mylącej nazwy metody, pozwalała ona na budowanie opisów dla dowolnego M i N . Inne podejście do generowania warunków M-z-N zaproponowane zostało przez Wneka i Michalskiego [187]. Analizowali oni hipotezy reguły wygenerowanych algorytmem AQ15, poszukując w nich takich, dla których spełniona była relacja XOR. Następnie takie hipotezy były łączone w warunki M-z-N i wprowadzane jako nowe atrybuty.

2.3 Konstruktywna indukcja sterowana wiedzą

W konstruktywnej indukcji sterowanej wiedzą, do modyfikacji przestrzeni atrybutów używana jest wiedza domenowa eksperta . Może ona być podstawą do budowania nowych cech i hipotez lub też służyć do ich weryfikacji i oceny. Callan i Utgoff proponowali metodę CINDI, która automatycznie, na podstawie wprowadzonej wiedzy domenowej, generowała nowe atrybuty [31]. Ekspert mógł wprowadzić do algorytmu nie tylko opisujące go hipotezy, lecz także wprowadzać specyficzne dla danego problemu operatory, wykorzystywane później w trakcie konstruowania nowych atrybutów. Algorytm MIRO (Drastal i Czako) z kolei wykorzystywał wprowadzony przez użytkownika opis wiedzy do zbudowania tak zwanej „przestrzeni abstrakcji" (z ang. abstraction space) [51]. Była ona tworzona poprzez proces dedukcji reguł, składających się z alternatyw warunków typu atrybut wartość. Przestrzeń abstrakcji z rozszerzonym zestawem atrybutów była wykorzystywana następnie do indukcji wynikowego opisu przekazanych na algorytm danych. Inne specyficzne dla domeny podejście prezentuje algorytm COPPER. Wykorzystuje on zasady popularnie stoso-

wanej w fizyce analizy wymiarowej, w celu wyznaczenia modułów bezwymiarowych na podstawie wprowadzonej przez użytkownika wiedzy [96]. Algorytm tworzy nowe cechy poprzez łączenie wyznaczonych wcześniej modułów bezwymiarowych. Inne podejście zostało zaproponowane przez Elio et al. w metodzie LAIR. Wykorzystuje ona uczenie inkrementacyjne w celu stopniowego wprowadzania nowych atrybutów. Przestrzeń atrybutów rozszerzana jest dynamicznie wraz z przetwarzaniem kolejnych przykładów uczących, z wykorzystaniem wiedzy domenowej wprowadzonej przez eksperta [59].

2.4 Wielostrategiczna konstruktywna indukcja

Wszelkie metody konstruktywnej indukcji, niedające się sklasyfikować do wcześniej wspominanych grup lub łączące kilka z nich, określane są mianem wielostrategicznej konstruktywnej indukcji. Przykładem takiego rozwiązania jest AQ17-MCI [22]. Była to najbardziej zaawansowana pod kątem konstruktywnej indukcji forma algorytmu AQ. Pozwalała ona nie tylko generować nowe atrybuty na podstawie samych danych (DCI), ale także budować je w kolejnych iteracjach na podstawie testowych hipotez (HCI). Pod względem samego mechanizmu indukcji algorytm działał podobnie do wcześniejszych jego wersji. Inną metodą wielostrategicznej konstruktywnej indukcji jest INDUCE-1 (Larson i Michalski) [101]. Algorytm ten buduje na podstawie przekazanego zbioru reguł klasyfikacyjnych (KCI) nowy zbiór, maksymalizując przy tym zadane przez użytkownika kryterium jakości. Do budowy drugiego zbioru wykorzystywane są bardziej złożone rodzaje warunków, zbudowane na podstawie warunków zawartych w regułach wejściowych (HCI). Metoda CHIRA może zostać potraktowana w kontekście wielostrategicznej konstruktywnej indukcji. Przyjmuje ona na wejściu pewien wejściowy zbiór reguł. Opis taki można potraktować jako wiedzę wprowadzoną do algorytmu (KCI). Metoda ta agreguje reguły, łącząc je parami. Podczas scalania buduje ona nowe reprezentacje danych z użyciem warunków skośnych, budowanych na podstawie warunków reguł wejściowych (HCI). Inne podejście zostało z kolei zastosowane w algorytmie GALA 2.0 [83]. Funkcjonował on jako preprocesor danych, rozszerzający je o nowe cechy. W ten sposób mógł on zostać zintegrowany z dowolnym innym algorytmem uczenia maszynowego, co stanowiło jego unikalną cechę. GALA dzielił zbiór danych na grupy, następnie generując dla każdej z grup jedną cechę boolowską. Proces dzielenia i generowania cech kontynuowany był aż do momentu osiągnięcia zbiorów homogenicznych lub kryterium stopu. W trakcie

działania nowe atrybuty były oceniane, a ich pula modyfikowana i przycinana na podstawie heurystycznej miary jakości.

3 Metody zespołowe

Warto podkreślić, że wspomniane dotychczas metody indukcji reguł, stosujące strategie sekwencyjnego pokrywania, stanowią pewną ugruntowaną klasykę uczenia regułowego. Są one jednak w dalszym ciągu popularnie używane i aktywnie rozwijane. Wiele nowocześniejszych metod indukcji reguł odchodzi jednak od tradycyjnego sekwencyjnego pokrywania na rzecz innych paradygmatów. W niniejszej sekcji zostaną omówione wybrane podejścia, bazujące na metodach zespołowych.

Uczenie zespołowe polega na łączeniu wielu modeli bazowych w celu stworzenia silniejszego i dokładniejszego predyktora [121]. Dwa z najbardziej znanych podejść w tej dziedzinie to bagging i boosting [107, 120].

Bagging to technika, która generuje wiele modeli bazowych, z których każdy jest uczony na losowym podzbiorze danych treningowych, a następnie agreguje ich predykcje np. poprzez uśrednianie lub mechanizm głosowania. Podejście to pomaga zmniejszyć wariancję poszczególnych modeli, czyniąc zespół bardziej odpornym na przetrenowanie [25]. Popularnym algorytmem uczenia maszynowego opartym na baggingu jest las losowy, łączący wiele drzew decyzyjnych.

Boosting z kolei to podejście, które konstruuje zespół poprzez iteracyjne dodawanie nowych modeli bazowych. Każdy kolejny koncentruje się podczas uczenia na przykładach błędnie przewidywanych przez poprzednie modele [63]. Popularnym algorytmem opartym na boostingu jest XGBoost [33]. Okazał się szczególnie skuteczny w szerokim zakresie zastosowań, zarówno w zadaniach klasyfikacji, jak i regresji [18, 84, 136].

Jedną z metod indukcji ważonych zbiorów reguł, wykorzystującą idee boostingu, jest SLIPPER [40]. Wykorzystuje on uogólnienie Adaboost, pozwalając regułom „wstrzymać się" od głosu dla niektórych przykładów, co zwiększa zrozumiałość wyników. Do generowania reguł jest wykorzystywana prosta procedura oparta o zachłanne poszukiwanie. Weiss i Rutgers proponowali metodę (Lightweight Rule Induction - LRI) wykorzystującą idee sekwencyjnego pokrywania [182]. Metoda ta generuje równą liczbę reguł dla każdej z klas, gdzie przykład klasyfikowany jest jako należący do klasy z największą liczbą pokrywających go reguł. Metoda ta adaptacyjnie waży przykłady treningowe w celu minimalizacji błędu na zbiorze treningowym, zachowując przy tym ustalone ograniczenia w rozmiarze i liczbie reguł. Algorytm

ENDER, zgodnie z ideą boostingu, iteracyjnie indukuje nowe reguły, koncentrując się na przykładach, które były najtrudniejsze do sklasyfikowania przez dotychczasowy zespół [48]. Wykorzystuje on przy tym i minimalizuje funkcje straty. Zastosowania różnych funkcji straty oraz tak zwanych miar nieczystości pozwalało kontrolować kompromis pomiędzy dokładnością (dyskryminacją) a pokryciem (kompletnością) wynikowego zbioru reguł. Metoda ta znajduje zastosowanie zarówno dla problemu klasyfikacji jak i regresji. Proponowana przez Stefanowskiego metoda dokonywała wielokrotnego próbkowania zbioru danych, poprzez losowanie ze zwracaniem [168]. Każda z próbek miała identyczny rozmiar jak oryginalny zbiór treningowy. Dla każdej próbki uruchamiany był następnie algorytm indukcji reguł MODLEM [167], oparty na sekwencyjnym pokrywaniu.

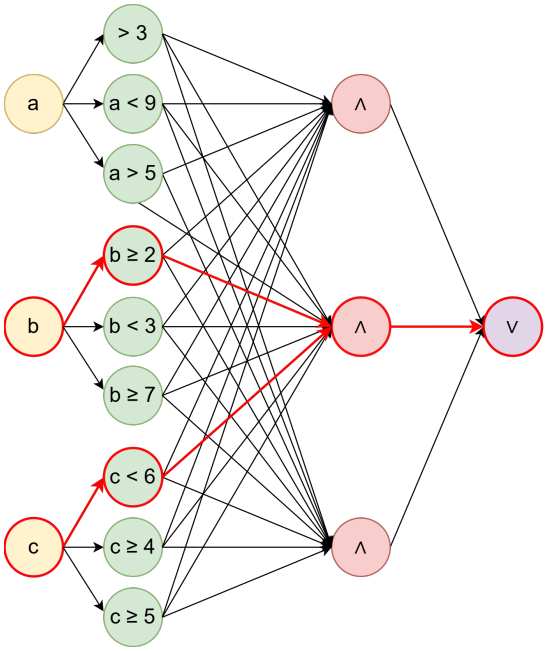
4 Metody stosujące sztuczne sieci neuronowe

Inną grupą algorytmów indukcji reguł są te, oparte o sztuczne sieci neuronowe. Tego typu podejścia można podzielić na dwie grupy. Pierwsza skupia się na ekstrakcji reguł z już wytrenowanej sieci, zwykle na podstawie jej wag. Druga grupa uczy się reguł w trakcie uczenia samej sieci. W pierwszym podejściu wynikowy zbiór reguł, zbudowany na podstawie sieci neuronowej, może funkcjonować inaczej niż ona sama. W drugim wariancie z kolei działanie obydwu modeli jest identyczne w kontekście predykcji.

Duch et al. [54] proponowali w swojej pracy metodę ekstrakcji reguł logiki ostrej i rozmytej na podstawie specjalnej dedykowanej struktury sieci neuronowej o nazwie MLP2LN . Sieć ta składała się z trzech warstw. Pierwsza uczyła się warunków logicznych (prostych oraz ich negacji), druga z kolei budowała z nich reguły koniunkcyjne. Ostatnia warstwa agregowała decyzje poszczególnych reguł, pełniąc rolę wyjścia modelu.

Przykładowa sieć o takiej strukturze przedstawiona została na rysunku 4.1. Kolorem żółtym zaznaczono wejścia sieci odpowiadające atrybutom numerycznym. Rysunek przedstawia sieć z wszystkimi możliwymi połączeniami. Każdemu z nich odpowiada pewna binarna waga ustalana w trakcie procesu treningu. W praktyce więc w procesie predykcji biorą udział jedynie połączenia z wagami równymi 1. Kolorem czerwonym zaznaczona została przykładowa reguła wyekstrahowana z sieci. Zakładając, że każde z połączeń zaznaczonych na czerwono posiada wagę 1, reguła ta ma postać:

$$\textbf{JEŻELI } b \geqslant 2 \wedge c < 6 \textbf{ TO } \text{klasa} = \{1\}$$



Rysunek 4.1: Struktura przykładowej sieci MLP2LN

Sieć była trenowana tradycyjnie metodą propagacji wstecznej błędu, z użyciem metody spadku wzdłuż gradientu. Dodatkowo autorzy proponowali modyfikację metody o nazwie C-MLP2LN, umożliwiającą inkrementacyjną rozbudowę sieci w trakcie treningu. Dzięki temu możliwe było skrócenie czasu treningu oraz ograniczenie jej rozmiaru. Reguły były uzyskiwane z sieci poprzez stopniowe zwiększanie nachylenia zbocza sigmoidalnych funkcji aktywacji neuronów oraz wykorzystanie specjalnej regularyzacji. Inną metodą ekstrakcji reguł na podstawie sieci neuronowej proponuje w swej pracy Tsukimoto [173]. Nie wprowadza ona specjalnej architektury sieci i może być stosowana dla każdej, gdzie funkcje aktywacji są monotonicznie rosnące. Metoda aproksymuje każdy neuron sieci funkcją boolowską, minimalizując przy tym błąd modelu, a rozpoczynając od warstwy wejściowej i kończąc na wyjściowej. Wynikowy zbiór reguł ma postać DNF. Metoda, w przeciwieństwie do wcześniej opisywanej, nie wymaga także modyfikacji samej procedury uczenia sieci. Craven i Shavlik opisywali alternatywną metodę, w której wynikowy zbiór reguł był uczony w oparciu o predykcje wytrenowanej sieci. Zbiory reguł w postaci DNF były tutaj generowane na podstawie zbioru treningowego, gdzie kolumna decyzyjna była zamieniana z wyjściem sieci neuronowej [44]. Nieco podobne podejście, z zastosowaniem algorytmu RuleKit [74], było opisywane również przez autora niniejszej pracy w artykule zatytułowanym „Rule-based approximation of black-box classifiers

for tabular data to generate global and local explanations", zaprezentowanym na międzynarodowej konferencji FedCSIS w 2022 roku [112]. Zostało ono zastosowane do wyjaśniania nie tylko dla sieci neuronowych, lecz także innych popularnych modeli uczenia maszynowego.

Katzir i inni proponowali z kolei dedykowaną architekturę sieci neuronowej dla danych tabelarycznych, która umożliwiała uczenie się formuł logicznych w postaci DNF [94]. Składa się ona z czterech warstw, z tego jedynie pierwsza i ostatnia są warstwami uczącymi. Autorzy wykorzystują różniczkowalne aproksymacje operatorów logicznych koniunkcji i alternatywy jako funkcje aktywacji neuronów. Metoda czerpie inspiracje z metod takich jak lasy losowe, wprowadzając moduł wyboru cech (z ang. feature selection) oraz mechanizm lokalizacji (z ang. localization). Wybór cech polega na wyuczeniu się maski binarnej, określającej warunki wchodzące w skład reguły. Lokalizacja jest realizowana poprzez trenowanie poszczególnych neuronów sieci na wybranym losowo podzbiorze przykładów. Zapobiega to przetrenowaniu modelu oraz wielokrotnemu wyszukiwaniu tych samych zależności w danych. Proponowana przez Muselli metoda SNN [128] pozwala na bezpośrednie uczenie się reguł DNF, bez konieczności ich ekstrakcji. Struktura sieci mapuje się bezpośrednio na reguły logiczne. Metoda zamienia dane treningowe na wektor binarny poprzez dyskretyzację atrybutów numerycznych i kodowanie nominalnych. Sieć nie uczy się w tym wypadku z użyciem metod spadku wzdłuż gradientu, lecz z wykorzystaniem specjalnej techniki *Shadow Clustering* [129]. Pozwala ona rekonstruować funkcje boolowskie na podstawie fragmentu ich tablic prawdy. Wykorzystywana w tej metodzie sieć nie posiada trenowanych wag, zamiast tego algorytm uczy się wprost połączeń pomiędzy poszczególnymi neuronami. Proponowana przez Qiao et al. sieć DR-Net składała z jednej warstwy ukrytej, uczącej się koniunkcji binarnych wejść [140]. Metoda, podobnie jak wcześniejsza, wymagała początkowego kroku dyskretyzacji i kodowania danych wejściowych w celu ich binaryzacji. Autorzy proponowali różniczkowalne funkcje aproksymujące operacje logicznej koniunkcji (pierwsza warstwa) i alternatywy (warstwa wyjściowa). Metoda wykorzystywała technikę straight-through estimator (STE) [14], polegającą na zastosowaniu różnych funkcji aktywacji w trakcie propagacji w przód i wstecz. W przypadku propagacji w przód były wykorzystywane logiczne funkcje alternatywy i koniunkcji. W trakcie propagacji wstecznej były używane ich różniczkowalne aproksymacje.

5 Predykcja z użyciem modeli regułowych

We wcześniejszej części pracy skupiono uwagę na różnorodnych mechanizmach i strategiach indukcji reguł decyzyjnych. Pominięty jednak został aspekt zastosowania ich jako modeli predykcyjnych. W niniejszej sekcji zostaną opisane najpopularniejsze metody dokonywania predykcji z użyciem modeli regułowych, stosowane w literaturze. Bywają one często zależne od samej specyfiki działania poszczególnych algorytmów.

Metody takie jak FOIL, Ripper czy CN2 budują listę reguł w określonej kolejności [36, 37, 143]. Dla przykładu algorytm Ripper rozpoczyna indukcję od najmniej licznej z klas. W przypadku list reguł predykcja dla każdego przykładu opiera się na sekwencyjnym rozpatrywaniu reguł w pewnej określonej kolejności. Wartość predykcji jest ustalana na podstawie pierwszej z nich, która pokryła dany przykład [36, 37].

W przeciwieństwie do list reguł, dla nieporządkowych zbiorów reguł jest możliwe wystąpienie tak zwanych konfliktów. Oznaczają one sytuacje, gdy pojedynczy przykład jest pokrywany przez reguły różniące się konkluzją. W tym wypadku konieczny jest pewien mechanizm, wybierający ostateczną wartość predykcji. Najczęściej stosowanym podejściem jest różnego rodzaju agregowanie decyzji poszczególnych reguł. W przypadku klasyfikacji jest tutaj popularnie stosowana procedura głosowania. Każda reguła pokrywająca przykład „oddaje głos" na klasę, na którą wskazuje jej konkluzja. Jako wynik głosowania, wybierana jest klasa z największą liczbą głosów.

W najprostszym przypadku głos każdej z reguł jest równoważny. Ma to miejsce np. w przypadku algorytmu LRI [182]. Często jednak do głosów poszczególnych reguł przypisywane są pewne wagi, zwiększając lub zmniejszając ich wpływ na ostateczny wynik głosowania. W literaturze obecnych jest wiele sposobów wyznaczania wag głosowania dla reguł. Przykładowo, dla algorytmu AQ-15 są one ustalane w oparciu o procent przykładów pozytywnych pokrywanych przez regułę [117].

W przypadku metody RuleKit jest stosowane ważone głosowanie, gdzie wagi reguł obliczane są z wykorzystaniem pewnej miary jej jakości. Może to być dowolna funkcja bazująca na pokryciu reguły, ustalona przez użytkownika jako parametr. W przypadku problemów regresji i przeżycia zastosowana została modyfikacja procesu głosowania. Dla pierwszego z nich, wartość predykcji ustalana jest jako średnia ważona wartości, na jakie wskazują konkluzje reguł pokrywających przykład. Podobna technika jest używana w metodzie ENDER [168]. Dla problemu analizy przeżycia, wagi głosowania są obliczane w RuleKit zawsze z użyciem testu logrank [109]. Wynikiem głosowania jest uśredniony estymator Kaplana-Meiera.

Inną popularną techniką rozwiązywania konfliktów jest wybór pojedynczej, najlepszej pod pewnym kątem reguły spośród wszystkich pokrywających dany przykład. Wartość predykcji ustalana jest następnie z użyciem tej jednej reguły. Taka strategia jest stosowana między innymi w algorytmie LORD [86]. Do wyboru najlepszej reguły wykorzystywana jest tam heurystyczna miara jakości. W przypadku, gdy dla kilku reguł jest ona jednakowa, wybiera się tę z większym p. Gdy i to nie pozwala na rozstrzygnięcie konfliktu, za lepszą uznaje się regułę wskazującą na mniej liczną klasę [86]. Podobne podejście stosuje się w przypadku algorytmu BRACID, gdzie predykcji dokonuje reguła najbliższa przykładowi pod kątem miary HVDM reguła [131]. Konflikty są tu z kolei rozwiązywane na podstawie wsparcia reguły.

Dla wcześniej wspomnianych metod wykorzystujących sieci neuronowe, sposób dokonywania predykcji bywa odmienny. Przykładowo dla algorytmu Net-DNF, predykcja jest dokonywana bezpośrednio na podstawie aktywacji wyjściowej warstwy sieci. W przypadku klasyfikacji binarnej warstwa ta posiada sigmoidalną funkcję aktywacji [94]. W metodzie proponowanej przez Tsukimoto neurony sieci aproksymowane są z pomocą tzw. ciągłych funkcji logicznych. Podczas predykcji przykładowi przypisuje się klasę, na którą wskazuje neuron wyjściowy, którego funkcja aproksymująca osiągnęła najwyższą wartość [173]. Ciekawe rozwiązanie zastosowano w przypadku metody SwitchingNeuralNetworks. Wartość predykcji ustala się tutaj w procesie ważonego głosowania. Waga reguł jest wyznaczana dynamicznie na podstawie tego, czy wszystkie warunki w przesłance reguły pokrywają przykład, czy też nie. Dla pierwszego przypadku wynosi ona 1, w drugim 0.1.

W przypadku modeli regułowych może wystąpić sytuacja, w której żadna z reguł w zbiorze lub liście nie pokrywa danego przykładu. Najczęstszym sposobem radzenia sobie z taką sytuacją jest przypisywanie mu pewnej domyślnej decyzji [36, 37, 74, 86].

Na koniec warto zaznaczyć, że w przypadku niektórych prac autorzy nie opisują dokładnie procesu, w jakim wykonywana jest predykcja.

6 Podsumowanie

Mimo licznych prac dowodzących, iż wykorzystywanie warunków złożonych może pozytywnie wpływać na wielkość i jakość uzyskiwanych zbiorów reguł [22] [126] [10], większość popularnie stosowanych metod indukcji reguł stosuje jedynie warunki proste lub ich negacje [143], [37], [36], [74]. Wiele spośród wspomnianych rozwiązań, takich jak np. algorytm AQ, nie nadaje się do użycia na większych zbiorach danych

ze względu na zbyt dużą złożoność obliczeniową. Większość metod nie posiada także wydajnej, łatwej w uruchomieniu i publicznie dostępnej implementacji, co utrudnia ich wykorzystanie. Dodatkowo żadne z wcześniej wspomnianych rozwiązań nie umożliwiało indukcji reguł z warunkami złożonymi zarówno dla problemu klasyfikacji, jak i regresji oraz przeżycia. Niniejsza praca i opisane w niej metody starają się wypełnić tę lukę badawczą i prezentują odświeżone podejście do nieco zapomnianej już, zdaniem autora, koncepcji konstruktywnej indukcji.

5 Głębokie dyskretne uczenie

Wprowadzenie koncepcji tak zwanego głębokiego uczenia (z ang. deep learning), wraz z początkiem dwudziestego pierwszego wieku, w znaczący sposób wpłynęło na dalszy rozwój systemów uczenia maszynowego. Klasyczne metody, w tym metody regułowe, zostały w dużej mierze wyparte na rzecz tego nowego paradygmatu [154]. Głębokie uczenie pozwoliło osiągnąć nieosiągalne wcześniej rezultaty w dziedzinach takich jak przetwarzanie języka naturalnego czy rozpoznawanie obrazów [104]. Nie oznacza to jednak, że klasyczne metody uczenia zostały całkowicie przez nie zastąpione. Prostsze modele, takie jak np. lasy losowe, wciąż są popularnie stosowane między innymi w kontekście danych tabelarycznych, osiągając nierzadko lepsze wyniki [94]. Pomimo istnienia wielu architektur głębokich sieci neuronowych dedykowanych do konkretnych danych lub problemów, wciąż brakuje jednej powszechnie akceptowanej i stosowanej architektury dla danych tabelarycznych [94]. Jej poszukiwania są wciąż otwartym problemem, skupiającym uwagę wielu badaczy. Modele głębokiego uczenia są zwykle traktowane jako czarne skrzynki (z ang. black-box models). Oznacza to, że wyjaśnienie i zrozumienie, dlaczego model funkcjonuje w określony sposób, jest trudne lub wręcz niemożliwe [194]. Interpretowalność modelów uczenia maszynowego bywa niezwykle istotna w niektórych dziedzinach takich jak np. prawo, medycyna czy finanse, gdzie konieczna jest możliwość weryfikacji procesu podejmowania przez nie decyzji [194]. W takich przypadkach modele takie jak drzewa decyzyjne lub zbiory reguł, uznawane powszechnie za jedne z najbardziej interpretowalnych, wciąż znajdują zastosowanie [67].

Termin głębokie uczenie, bywa często wręcz utożsamiany z głębokimi sieciami neuronowymi o dużej liczbie warstw. Oba te pojęcia, mimo że pokrewne i mocno ze sobą związane, nie są jednak tożsame [3]. Głębokie uczenie postrzegać można również szerzej, jako pewnego rodzaju paradygmat uczenia maszynowego [45]. Opiera się on na hipotezie, że głębokie (hierarchiczne) modele mogą reprezentować niektóre funkcje w wydajniejszy sposób niż modele płytkie [13]. Hipoteza ta znalazła potwierdzenie w wielu badaniach [102, 134]. Delalleau i Bengio wykazali w swych teoretycznych

rozważaniach, że głębokie sieci typu SPN (z ang. sum-product network), pozwalają na zamodelowanie tej samej funkcji z wykładniczo mniejszą liczbą neuronów niż sieć tego samego typu o płaskiej architekturze z jedną warstwą ukrytą [47]. Co ciekawe, podobna zależność została również potwierdzona dla układów logicznych składających się z bramek AND, OR i negacji. Håstad wykazuje w swej pracy, że istnieją funkcje, których zamodelowanie wymaga wielomianowej liczby bramek dla układu składającego się z k warstw. Po ograniczeniu liczby warstw do $k-1$, liczba bramek rośnie jednak wykładniczo [81].

W uogólnieniu modele głębokiego uczenia cechuje warstwowa, hierarchiczna natura, gdzie kolejne warstwy w sposób automatyczny wyodrębniają cechy wyższego poziomu z danych. Te nowe cechy stanowią bardziej abstrakcyjną i złożoną reprezentację danych, niż ta uzyskana bezpośrednio z surowych atrybutów [3]. Tak rozumianą ideę głębokiego uczenia można zastosować również do innych rodzajów modeli uczenia maszynowego, nie tylko sieci neuronowych.

Beck i Fürnkranz jako pierwsi wprowadzili w swych pracach pojęcie głębokiego uczenia regułowego (z ang. deep rule learning), nazywanego przez nich także głębokim dyskretnym uczeniem [8]. Definiują je oni jako uczenie strukturalnych zbiorów reguł, których predykcje są łączone w celu tworzenia pomocniczych zmiennych (tzw. konceptów), będących następnie wykorzystywanych jako cechy wewnątrz innych reguł [8]. Jak wykazują oni w swej pracy, jest to otwarty temat badawczy i obecnie żadna z istniejących metod uczenia regułowego nie jest zdolna do tworzenia tego typu opisów. Podobnie jak Håstad dowodzą oni, że tego typu zagnieżdżone reguły ostre mogą prowadzić do mniej licznych i złożonych zbiorów reguł. W swoich kolejnych pracach proponują oni dwie możliwe ścieżki badawcze. Pierwsza z nich zakłada wykorzystanie głębokich sieci neuronowych, druga z kolei polega na użyciu bardziej tradycyjnych metod indukcji reguł opartych o przeszukiwanie [12].

Wszystkie wspomniane w sekcji 4 rozdziału 4 metody indukcji reguł oparte o sieci neuronowe, mimo wykorzystania technik głębokiego uczenia, generują proste, niehierarchiczne zbiory reguł. Nie spełniają one tym samym założeń głębokiego dyskretnego uczenia proponowanych przez Beck i Fürnkranz. Indukcja reguł ostrych z wykorzystaniem głębokich sieci neuronowych niesie ze sobą szereg trudności i problemów. Trening modeli głębokich odbywa się zwykle z wykorzystaniem techniki propagacji wstecznej (z ang. Backpropagation). [196]. Polega ona na iteracyjnym dostosowywaniu wag modelu, na podstawie obliczonych wartości gradientu funkcji straty [196]. Technika ta wymaga jednak, aby zarówno funkcje straty, jak i aktywacji poszczególnych

neuronów były różniczkowalne. W przypadku reguł ostrych, gdzie mamy do czynienia z dyskretną logiką dwuwartościową, założenie to jest niemożliwe do spełnienia wprost. Problem jest najczęściej rozwiązywany poprzez wykorzystanie różniczkowalnych funkcji aktywacji aproksymujących funkcje logiczne w trakcie treningu. Wynikowe zbiory reguły uzyskiwane są poprzez dyskretyzację wartości wag modelu, czy to w trakcie treningu (np. MLP2LN [54] lub DR-Net [140]), czy też po jego zakończeniu (NN2Rules [99], H . Tsukimoto 2000 [173]). Dyskretyzacja wag sieci neuronowej wiąże się jednak nieodłącznie z utratą informacji, a za tym możliwym pogorszeniem wyników modelu [141]. Innym podejściem jest wykorzystanie dedykowanej metody uczenia sieci nieopartej o propagację wsteczną błędu. Przykładem tego typu rozwiązania jest Switching Neural Networks [128], gdzie do treningu wykorzystano specjalną technikę *Shadow Clustering* [129]. Wyniki zaprezentowane przez jej autorów wykazują jednak znaczące zwiększenie liczby reguł i ich długości w stosunku do zbiorów wygenerowanych bardziej klasyczną metodą C4.5 [152]. Autorzy metody wskazują także na znacząco wydłużony czas treningu względem algorytmu C4.5 [129]. Problem złożoności generowanych zbiorów reguł jest często pomijany w przypadku metod bazujących na sieciach neuronowych. Aspekt ten jest jednak niezwykle istotny w aspekcie eksploracji danych. Preferowaną cechą jest tam interpretowalność uzyskiwanych opisów, która w przypadku zbiorów reguł zależy głównie od ich złożoności [68]. Jedną z niewielu prac dotyczących tej tematyki prezentujących wyniki dotyczące złożoności uzyskiwanych zbiorów reguł, jest praca Qiao at al. opisująca metodę DR-Net [139]. Autorzy porównują własną metodę z algorytmami takimi jak Ripper i CART na pięciu tabelarycznych zbiorach klasyfikacyjnych. Wyniki pokazują tutaj znaczącą przewagę metody DR-Net nad pozostałymi [139]. Eksperymenty przeprowadzone przez autorów tej pracy na innych popularnych benchmarkowych zbiorach danych nie potwierdziły jednak tych wniosków. Wyniki tychże eksperymentów przedstawione zostały w tabeli 5.1. Algorytm DR-Net został porównany z algorytmem RuleKit [74] opartym na strategii sekwencyjnego pokrywania. Do prób użyto implementacji[1] udostępnionej przez autorów. Eksperymenty przeprowadzone zostały w trybie 5-krotnej walidacji krzyżowej z wykorzystaniem domyślnych hiperparametrów dla obydwu modeli. Metoda DR-Net generowała zbiory o znacząco większej liczbie reguł. Reguły zawierały również większą liczbę warunków niż te uzyskane algorytmem RuleKit. Co ciekawe, także jakość klasyfikacji okazała się gorsza dla wszystkich testowanych

[1]Link do oficjalnej implementacji metody DR-Net `https://github.com/Joeyonng/decision-rules-network`, [dostęp 16.09.2025].

Zbiór danych	Metoda	BAcc	Liczba reguł	Średnia liczba warunków w regule
monk-2	DR-Net	0.500	50.0	15.0
	RuleKit	0.598	21.8	3.4
mushroom	DR-Net	0.990	49.8	13.7
	RuleKit	1.000	4.0	3.9
iris	DR-Net	0.333	50.0	34.0
	RuleKit	0.940	7.6	2.2
cleveland	DR-Net	0.200	50.0	63.0
	RuleKit	0.314	39.2	6.7
hepatitis	DR-Net	0.500	50.0	65.0
	RuleKit	0.655	7.4	3.0

Tabela 5.1: Wyniki eksperymentów przeprowadzonych dla metod DR-Net oraz RuleKit

zbiorów dla metody DR-Net. Niewykluczone, że staranny dobór hiperparametrów modelu pozwoliłby poprawić uzyskane wyniki. Niemniej jednak nie potwierdzają one entuzjastycznych wniosków autorów metody.

Wiele istniejących metod indukcji reguł z wykorzystaniem sieci neuronowych działa jedynie na pewnym określonym rodzaju danych. Przykładowo zarówno Switching Neural Networks jak DR-Net operują jedynie na atrybutach nominalnych. Atrybuty numeryczne mogą zostać obsłużone jedynie poprzez uprzednie wykorzystanie dyskretyzacji. Metoda Net-DNF umożliwiała z kolei operowanie wyłącznie na danych numerycznych. Autorzy jednak pomijają problem interpretowalności modelu. Sama metoda, choć inspirowana metodami regułowymi, nie pozwala uzyskać interpretowalnych opisów danych wprost ze struktury sieci [94]. Innym algorytmem indukcji reguł korzystającym z sieci neuronowych, operującym na danych numerycznych, jest R2R proponowany przez Kusters et al [98]. Reguły budowane z wykorzystaniem tej metody zawierają jednak w przesłankach wyłącznie warunki skośne. Autorzy wspominają o możliwości wprowadzania do reguł innych rodzajów warunków, również operujących na atrybutach nominalnych, poprzez dodanie dodatkowych atrybutów do zbioru treningowego. Tego typu zabieg może znacząco zwiększyć rozmiar sieci neuronowej, wpływając na czas jej treningu. Kusters et al. porównują w swej pracy metodę R2R z metodą DR-Net oraz algorytmem Ripper, pod kątem zarówno jakości predykcji, jak i złożoności reguł. Przedstawione przez nich wyniki dowodzą, że zbiory reguł uzyskiwane z użyciem R2R zawierają na ogół mniej reguł z mniejszą liczbą warunków niż pozostałe metody [98]. Wyniki zostały zaprezentowane jedynie na pięciu publicz-

nie dostępnych zbiorach rzeczywistych oraz pięciu zbiorach syntetycznych. Należy zaznaczyć, że mniejsza liczba reguł i warunków nie musi w tym wypadku iść w parze z poprawą interpretowalności. Zastosowane w R2R warunki skośne, szczególnie oparte na wielu atrybutach, bywają na ogół trudniejsze w zrozumieniu przez człowieka, choć nierzadko bardziej opisowe [161].

Istnieje wiele prac badawczych wskazujących na problem wysokiej redundancji w sieciach neuronowych [34, 80, 91]. Modele tego typu posiadają często ogromną ilość trenowanych parametrów, które zwykle reprezentowane są jako liczby zmiennoprzecinkowe z wysoką precyzją [141]. Kwantyzacja, a więc ograniczenie precyzji, z jaką przechowywane są wagi modelu, może w znaczący sposób zmniejszyć ilość zużytej pamięci oraz czas obliczeń związanych z procesem predykcji. Binaryzacja jest w tym wypadku najbardziej skrajną formą kwantyzacji, która jednak umożliwia zastąpienie najbardziej kosztownych obliczeniowo operacji mnożenia macierzy, szybkimi operacjami XNOR na poszczególnych bitach [141]. Sieci neuronowe, których wagi posiadają tylko dwie możliwe wartości, nazywamy sieciami binarnymi [141]. Odpowiednikiem sieci binarnych dla wag z trzema możliwymi wartościami są sieci ternarne [4]. Sieci binarne i ternarne mogą być stosowane do modelowania funkcji logicznych [35, 50].

Kwantyzacja wartości wag modelu wiąże się jednak nierozłącznie z utratą informacji oraz, co za tym idzie, potencjalnym pogorszeniem jakości predykcji. Pomimo tak skrajnej formy kwantyzacji jak binaryzacja, sieci binarne wciąż mogą osiągać bardzo dobre wyniki predykcyjne. Courbariaux et al. w swej pracy opisującej metodę treningu sieci binarnej BinaryConnect, wskazują, że tego typu modele mogą osiągać wyniki bardzo zbliżone do sieci z wagami o pełnej precyzji [43]. Motywacją większości prac w tym temacie jest jednak optymalizacja pamięciowa i obliczeniowa, a nie chęć uzyskania interpretowanych opisów danych. Sieci binarne, pomimo prostszej budowy, nie można traktować zawsze jako interpretowalnych modeli. Wciąż mogą one wykorzystywać skomplikowane i trudne w zrozumieniu przez człowieka przekształcenia i funkcje aktywacji [141].

Drugą ścieżką badawczą w głębokim dyskretnym uczeniu, proponowaną przez Beck i Fürnkranz, jest opracowanie dedykowanego algorytmu opartego o istniejące metody indukcji reguł ostrych w postaci DNF i CNF. Jak wykazują oni w swej pracy, metody uczenia regułowego realizujące takie podejście są w literaturze praktycznie nieobecne [8]. Autorzy proponują tutaj metodę nawiązującą do wspomnianej wcześniej konstruktywnej indukcji sterowanej hipotezami. Autorzy bazują w niej

na opracowanym przez siebie algorytmie LORD [86]. Indukuje on tak zwane lokalnie optymalne reguły w formacie DNF, gdzie dla każdego przykładu w zbiorze treningowym generowana jest osobna reguła, która go opisuje. Autorzy wprowadzają następnie modyfikację LORD o nazwie $\overline{LORD}$ umożliwiającą generowanie reguł w postaci CNF. Autorzy wykorzystują w niej prawa de Morgana, przekształcając problem uczenia się formuł CNF w problem uczenia się formuł DNF na tak zwanej zanegowanej przestrzeni pokrycia. Każdą formułę logiczną w postaci DNF można przekształcić w równoważną formułę logiczną w postaci CNF. Można tego dokonać poprzez zastosowanie praw de Morgana.

$$(a \wedge b) \vee (c \wedge d) = \neg[(\neg a \vee \neg b) \wedge (\neg c \vee \neg d)] \tag{5.1}$$

Jak można zauważyć na podstawie Równania 5.1, uzyskana w ten sposób formuła CNF składa się z zanegowanej koniunkcji, a poszczególne literały w alternatywach również są zanegowane. Przenosząc to na język reguł, każda koniunkcja w wyrażeniu DNF odpowiada pojedynczej regule decyzyjnej, literały z kolei reprezentują warunki logiczne. Korzystając z równania (5.1) można dokonać przekształcenia reguły DNF na regułę CNF poprzez zanegowanie poszczególnych jej warunków oraz całej przesłanki. Autorzy pracy stosują tutaj trik, polegający na wykorzystaniu tak zwanej odwróconej przestrzeni pokrycia. Zamiast indukować regułę DNF dla klasy pozytywnej, indukują regułę dla klasy przeciwnej, wykorzystując przy jej wzroście negacje warunków. Następnie na koniec przekształcają przesłankę reguły z koniunkcji na alternatywę z wykorzystaniem Równania 5.1. Poprzez wykorzystanie odwróconej przestrzeni pokrycia negacje w równaniu znoszą się, dając w rezultacie regułę niezawierającą negacji. Poniżej został rozpisany przykład indukcji tego typu reguły dla klasy *1*. Uzyskana przez algorytm reguła ma postać daną równaniem 5.2.

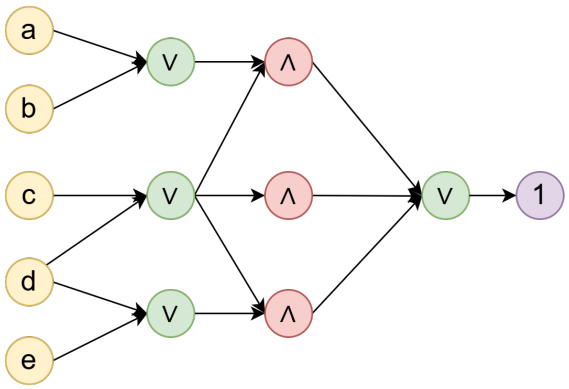
$$\textbf{JEŻELI } a \neq \{0\} \wedge b \ \neq \{1\} \textbf{ TO } klasa \neq \{1\} \tag{5.2}$$

Po zastosowaniu dla niej Równania 5.1 otrzymujemy podstawę daną równaniem (5.3)

$$\textbf{JEŻELI } a = \{0\} \vee b \ = \{1\} \textbf{ TO } klasa = \{1\} \tag{5.3}$$

Zauważyć należy, że powyższe przekształcenie funkcjonuje jedynie w przypadku klasyfikacji dwuklasowej.

W oparciu o wyżej wspomniane algorytmy LORD i $\overline{LORD}$, Beck i Fürnkranz proponują metodę głębokiej indukcji reguł o nazwie CORD [11]. W pierwszym kroku



Rysunek 5.1: Reprezentacja reguł głębokich w postaci sieci

generowany jest zbiór reguł w formacie CNF z wykorzystaniem $\overline{LORD}$. Następnie dla każdej reguły ze zbioru tworzony jest nowy atrybut binarny, określający, czy reguła pokrywa dany przykład uczący. Tak wygenerowane nowe atrybuty dodawane są do oryginalnego zbioru treningowego. Na rozszerzonym zbiorze treningowym dokonywana jest następnie indukcja zbioru reguł DNF z użyciem LORD. W ten sposób poszczególne reguły wynikowego zbioru mogą, lecz nie muszą, odpowiadać wyrażeniom w postaci CNF. Warto zauważyć jednak, że druga iteracja indukcji z użyciem LORD nie musi wykorzystać w przesłankach reguł nowo dodanych atrybutów. Tym samym wynikowe reguły mogą być w niektórych sytuacjach równoważne z regułami uzyskanymi algorytmem LORD na oryginalnym zbiorze treningowym.

Autorzy w dalszej części artykułu proponują również wersję metody o nazwie DORC, umożliwiającą w analogiczny sposób uzyskiwanie reguł w postaci DNF. Beck i Fürnkranz porównują kolejne iteracje indukcji reguł oraz modyfikację zbioru treningowego do przekształceń dokonywanych przez kolejne warstwy sieci głębokich. W ten sposób opisany wcześniej algorytm odpowiada sieci z dwiema warstwami. Przykład takiej sieci znajduje się na rysunku 5.1. Kolorem zielonym są oznaczone operacje alternatywy logicznej a kolorem czerwonym koniunkcji. Fioletowy kolor oznacza wyjście modelu, wskazujące na klasę pozytywną.

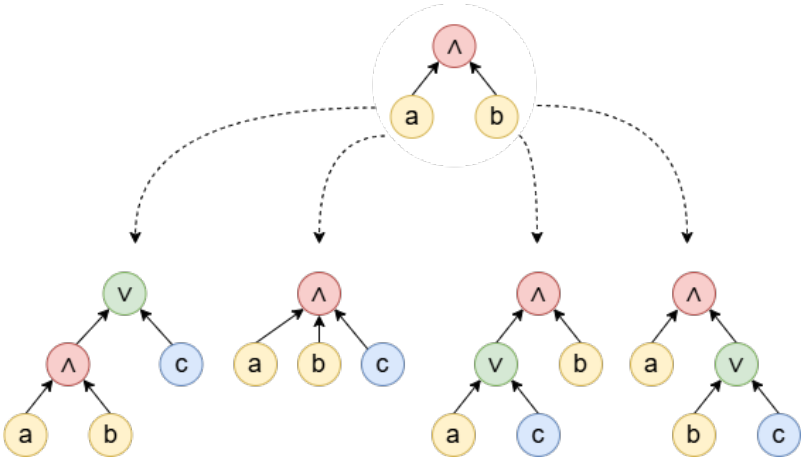
Zakładając, że atrybuty a, b, c i d są atrybutami binarnymi, sieć ta definiuje następujący zbiór reguł:

JEŻELI $(a \vee b) \wedge (c \vee d)$ **TO** $klasa = \{\mathbf{1}\}$
JEŻELI $c \vee d$ **TO** $klasa = \{\mathbf{1}\}$
JEŻELI $(c \vee d) \wedge (d \vee e)$ **TO** $klasa = \{\mathbf{1}\}$

Autorzy, w dalszej części artykułu, przeprowadzają także eksperymenty dla algorytmu o większej liczbie warstw (czterech). Prezentowane przez nich wyniki nie wykazały jednak jakoby większa liczba iteracji (warstw), związana była w istotny sposób z poprawą wyników predykcyjnych. Prowadziła ona czasami wręcz do ich pogorszenia w stosunku do podejścia dwuwarstwowego.

W ramach niniejszej pracy, w rozdziale 3, została przedstawiona metoda indukcji reguł ostrych DeepRules, zgodna z definicją dyskretnego głębokiego uczenia zaproponowaną przez Beck i Fürnkranz. Opierając się na wynikach opisanych przez tychże autorów, uwaga została skupiona na podejściu „dwuwarstwowym". Ten sam algorytm można jednak wykorzystać do uczenia bardziej zagnieżdżonych struktur logicznych. Proponowana metoda nie wymaga wielokrotnej indukcji reguł, jak w przypadku CORD. Zamiast tego reguły z zagnieżdżonymi wyrażeniami logicznymi generowane są w jednym kroku. Co więcej, algorytm ten umożliwia generowanie reguł zarówno w postaci CNF, jak i DNF. W przeciwieństwie do metody CORD nie operuje on tylko na atrybutach binarnych. Zamiast tego umożliwia indukcję dla danych nominalnych i numerycznych, zapewniając przy tym wbudowaną obsługę wartości brakujących. Dodatkowo algorytm wykorzystuje w regułach warunki negacji i relacji atrybutów, umożliwiając uzyskiwanie bogatszych opisów danych. Metoda znajduje zastosowanie również dla klasyfikacji wieloklasowej, bez konieczności uczenia osobnego zbioru reguł dla każdej z klas. Algorytm może także zostać użyty dla problemów regresji i analizy przeżycia.

Równolegle z prowadzonymi pracami badawczymi, ukazała się najnowsza publikacja Beck, Fürnkranz i Van Quoc Huynh. Autorzy proponują w niej alternatywną metodę głębokiego uczenia regułowego o nazwie ABPT (z ang. Alternating Boolean Pattern Trees) [12]. Nie wykorzystuje ona algorytmu LORD, zamiast tego bazuje na zachłannym przeszukiwaniu z wykorzystaniem heurystyk. Poszczególne reguły zawierające zagnieżdżone formuły logiczne z wykorzystaniem koniunkcji i alternatyw reprezentowane są tutaj w postaci drzewa. Liśćmi drzewa są warunki elementarne. Metoda, podobnie jak CORD, operuje jedynie na atrybutach binarnych, stąd warunki te mają postać $a = 1$, którą można zapisać w uproszczeniu jako a. Algorytm obsługuje problemy wieloklasowe poprzez indukcje reguł osobno dla każdej z klas. Warunki dodawane są do reguł w sposób iteracyjny, gdzie liczba iteracji określona jest parametrem algorytmu. Autorzy nie definiują żadnego innego kryterium stopu dla procesu wzrostu reguły. Metoda nie posiada także osobnej fazy przycinania. Po wykonaniu zadanej liczby iteracji reguła jest dodawana w swej obecnej formie do



Rysunek 5.2: Przykład wstawiania nowego warunku do reguły w algorytmie ABPT

wynikowego zbioru reguł. Przykład reprezentacji reguły w postaci drzewa oraz proces wstawiania do niej nowego warunku został ukazany na rysunku 5.2. Ponownie kolorem czerwonym oznaczono operacje koniunkcji, a zielonym alternatywy. Kolor żółty oznacza tutaj warunki elementarne. Niebieskim kolorem oznaczono nowo wstawiany warunek. W przykładzie tym zakładamy, że a, b, c i d to atrybuty binarne zbioru treningowego, a obecna postać przesłanki reguły to $a \wedge b$. Algorytm podejmuje próbę dodania do niej warunku c. Możliwe są aż cztery sposoby wstawienia go do drzewa. Są to kolejno:

- $(a \wedge b) \vee c$
- $a \wedge b \wedge c$
- $(a \vee c) \wedge b$
- $(b \vee c) \wedge a$

Jak widzimy, może ono rosnąć zarówno wszerz, jak i w głąb. Algorytm wybiera najlepszą możliwą formę drzewa, kierując się heurystyczną miarą jakości opartą na pokryciu reguły. Przeszukiwanie odbywa się z wykorzystaniem wiązki, gdzie przechowywanych jest kilka najlepszych form drzewa do wzrostu przy kolejnej iteracji.

ABPT wykazuje pewne podobieństwo do metody DeepRules, proponowanej w rozdziale 3. Wprowadza ona względem algorytmu CORD obsługę klasyfikacji wieloklasowej. Umożliwia także generowanie bardziej złożonych formuł logicznych o większej liczbie zagnieżdżeń. Wciąż jednak operuje ona tylko na atrybutach binarnych. Problem znajdowania optymalnych przedziałów dyskretyzacji atrybutów numerycznych

i ich kodowania nie jest przez autorów adresowany. Podobnie jak DeepRules, ABPT generuje reguły głębokie w jednym kroku indukcji, bez konieczności wielokrotnego trenowania modelu - jak w przypadku metody CORD.

6 Opracowane metody indukcji reguł

W niniejszej pracy autor proponuje aż trzy różne metody indukcji reguł ostrych, nawiązujące do opisanej w poprzednim rozdziale 5 koncepcji głębokiego dyskretnego uczenia regułowego. Zostały one przedstawione w porządku chronologicznym, w jakim zostały opracowane. Wszystkie z nich skupiają się na generowaniu bogatszych, bardziej złożonych opisów danych. Pierwsze dwie dokonują tego poprzez wykorzystanie różnego rodzaju warunków złożonych w przesłankach reguł. Ostatnia z metod stosuje w tym celu zagnieżdżone formuły logiczne.

Z powodów wspomnianych w rozdziale 5 zdecydowano się wykorzystać bardziej tradycyjne podejście do indukcji reguł, oparte na strategii sekwencyjnego pokrywania. Umożliwia ono, poprzez dobór odpowiednich miar jakości, kontrolowanie kompromisu pomiędzy liczbą generowanych reguł a ich zdolnościami predykcyjnymi [66]. Pozwalają one także wpływać na to, jak bardzo precyzyjne reguły zamierzamy uzyskać. Niewątpliwą zaletą tego podejścia jest to, że w swoich założeniach dąży ono do pokrycia wszystkich przykładów zbioru treningowego [64]. Choć w pewnych specyficznych przypadkach może to prowadzić do uzyskiwania reguł pokrywających bardzo niewiele przykładów, jest to jednak czasami cecha pożądana w kontekście odkrywania wiedzy i eksploracji danych. Pokrycie całego zbioru treningowego nie zawsze jest jednak równie łatwe do osiągnięcia w przypadku algorytmów generujących zbiory reguł na podstawie sieci neuronowych. Na etapie ekstrakcji reguł z wytrenowanej sieci następuje zwykle pewna utrata informacji [158]. Może to potencjalnie prowadzić do uzyskania zbioru reguł, który nie pokrywa pełnego zbioru treningowego. Zbiór taki może jednak wciąż osiągać dobre wyniki predykcyjne poprzez zastosowanie popularnego w uczeniu regułowym mechanizmu reguły domyślnej. Określa ona wartość predykcji przypisywaną przykładom niepokrytym przez jakąkolwiek z reguł [66].

Metody wykorzystujące sieci neuronowe optymalizują podczas treningu tak zwaną funkcję straty, która oparta jest zwykle na błędzie popełnianym przez cały model w trakcie predykcji [180]. W ten sposób możliwe jest osiągnięcie zadowalających wyników, nawet gdy część reguł w zbiorze sama w sobie nie posiada wysokiej jakości.

W przypadku metod sekwencyjnego pokrywania, gdzie optymalizowana jest jakość poszczególnych reguł, taka sytuacja jest mniej prawdopodobna. Jest to kolejna zaleta z punktu widzenia problemu odkrywania wiedzy, gdzie reguły o niskiej jakości mogą nie dostarczać istotnej wiedzy na temat danych.

Większości metod generujących reguły na podstawie sieci neuronowych polega na ekstrakcji interpretowalnego modelu, jakim jest zbiór reguł, z modelu czarnej skrzynki, jakim jest sieć neuronowa. Jest to jednak często związane z pogorszeniem jego zdolności predykcyjnych [158]. Dzieje się tak, ponieważ uzyskany zbiór reguł stanowi jedynie aproksymację wytrenowanej sieci neuronowej.

Z wyżej wspomnianych powodów w niniejszej pracy uwagę skupiono na poszukiwaniu efektywnych metod głębokiej indukcji reguł bezpośrednio z danych, z zastosowaniem strategii sekwencyjnego pokrywania.

Pierwszy z zaproponowanych algorytmów o nazwie ComplexConditions skupia się na indukcji zbiorów reguł ostrych w postaci DNF. Bazuje ona na istniejącym, dobrze przebadanym i opisanym algorytmie RuleKit. Rozszerza go jednak o możliwość generowania reguł zawierających różne rodzaje warunków złożonych, pozwalające odkrywać i opisywać nowe zależności w danych.

Kolejna proponowana metoda, nazwana MofNRules, jest rozwinięciem poprzedniej, wprowadzając do niej warunki typu M-z-N. Opracowane zostały dwa warianty algorytmu, umożliwiające generowanie reguł zarówno z warunkami przynajmniej M-z-N, jak i dokładnie M-z-N. Wartości zarówno M jak i N mogą być kontrolowane przez użytkownika poprzez parametry. Uzyskiwane tutaj zbiory reguł wciąż mogą zawierać wszystkie rodzaje warunków złożonych, użytych w pierwszej metodzie. Ich przesłanki nie muszą więc składać się wyłącznie z warunków M-z-N.

Ostatni opisany algorytm o nazwie DeepRules skupia się na indukcji reguł z zagnieżdżonymi wyrażeniami logicznymi koniunkcji i alternatyw w przesłankach. Nawiązuje on tym samym do proponowanej przez Beck'a i Fürnkranz'a koncepcji głębokiego dyskretnego uczenia. Umożliwia on generowanie reguł zawierających wszystkie rodzaje warunków złożonych stosowanych we wcześniej wspominanych algorytmach. Zastosowanie zagnieżdżonych koniunkcji i alternatyw w przesłankach pozwala operować na mniejszej przestrzeni poszukiwań, redukując przy tym czasy obliczeń.

Wszystkie ze wspomnianych metod umożliwiają generowanie reguł zarówno dla problemów klasyfikacji, regresji, jak i analizy przeżycia.

1 Metoda indukcji reguł z warunkami złożonymi – ComplexConditions

Metoda ComplexConditions umożliwia generowanie zbiorów reguł ostrych w postaci DNF, które w swoich przesłankach zawierają różnorodne warunki złożone. Szczegółowa specyfikacja tych warunków zostanie przedstawiona w dalszej części rozdziału. Przed przejściem do opisu jej działania, omówione zostaną pokrótce podstawy algorytmu RuleKit [74], na którym jest ona oparta. Ten krótki wstęp pozwoli czytelnikowi nabrać lepszego zrozumienia procesów i koncepcji stosowanych we wszystkich trzech opracowanych metodach.

1.1 Algorytm RuleKit - generowanie reguł z prostymi warunkami elementarnymi

Algorytm rozpoczyna działanie z pustym zbiorem reguł. Następnie iteracyjnie dodaje kolejne reguły aż do momentu pokrycia pewnej określonej części zbioru treningowego (w domyślnej konfiguracji jest to cały zbiór danych). Po każdej iteracji przykłady pokryte przez nowo dodaną regułę są eliminowane ze zbioru treningowego. Dzięki temu kolejne reguły są generowane w oparciu o przykłady niepokryte w poprzednich krokach. Warunkiem dodania reguły do zbioru jest pokrycie przez nią minimalnej, zdefiniowanej parametrem, liczby dotychczas niepokrytych przykładów.

Proces indukcji pojedynczej reguły składa się z dwóch faz: wzrostu i przycinania. Faza wzrostu dodaje w sposób iteracyjny warunki do przesłanki reguły. W oryginalnej wersji algorytmu RuleKit warunki łączone są zawsze operatorem koniunkcji. W każdej iteracji dodawany jest warunek maksymalizujący heurystyczną miarę jakości, bazującą na pokryciu reguły. Za każdym razem rozpatrywane są wszystkie możliwe warunki proste, mogące zostać utworzone na podstawie zbioru przykładów pokrytych przez aktualną formę reguły. Algorytm RuleKit oferuje wiele różnych miar jakości reguły, które mogą zostać użyte podczas wzrostu, jak i przycinania, umożliwiając również tworzenie własnych [73]. Abstrahując od miary jakości, algorytm w ogólnym przypadku dąży do maksymalizacji liczby pokrytych przykładów pozytywnych, minimalizując przy tym liczbę pokrywanych przykładów negatywnych. Poprzez dobór odpowiednich miar można sterować procesem indukcji, kontrolując kompromis pomiędzy precyzją uzyskiwanych reguł a liczbą przykładów przez nie pokrywanych [66]. Pseudokod procesu wzrostu reguły w algorytmie RuleKit przedstawiony został na

listingu 1. Funkcja WYZNACZPULEWARUNKÓW, wywoływana w linii 5, generuje pulę warunków możliwych do dodania do reguły, na podstawie zbioru przykładów przez nią pokrytych. Warunki wyznaczane są osobno dla każdego z atrybutów zbioru treningowego, a następnie łączone w jeden zbiór C. Proces ten różni się w zależności od typu rozpatrywanego atrybutu. Dla atrybutów nominalnych brane są pod uwagę wszystkie warunki proste dla każdej wartości atrybutu, pojawiającej się w zbiorze pokrytych przez regułę przykładów. W przypadku atrybutów numerycznych proces rozpoczyna od wyznaczenia listy wszystkich unikalnych wartości atrybutu, oznaczonej jako v. Na tym etapie usuwane są wartości brakujące. W następnym kroku lista ta sortowana jest w porządku rosnącym. Następnie wyznaczany jest zbiór wartości E, określony równaniem (6.1). Dla każdego elementu e_j należącego do zbioru E generowane są dwa warunki: $a_i < e_j$ oraz $a_i \geqslant e_j$, gdzie a_i to atrybut numeryczny, dla którego wyznaczany jest zbiór warunków.

$$E = \left\{ \frac{v_{i+1} + v_i}{2} : 0 \leqslant i < |v| - 1 \right\} \tag{6.1}$$

Stosując wprowadzone w rozdziale 1 oznaczenia liczby wierszy n i liczby kolumn m, możliwe jest określenie maksymalnej liczby warunków ocenianych w trakcie wzrostu reguły. Obliczenia te okażą się pomocne podczas szacowania złożoności obliczeniowej proponowanych metod. Przy założeniu, że atrybut a_i jest nominalny oraz przyjmuje n możliwych wartości, można utworzyć dla niego n warunków prostych. W przypadku, gdy atrybut jest numeryczny, zakładając, że jego wartość dla każdego przykładu jest unikalna, jest ich $2n - 2$. Wynika to z faktu, że maksymalny rozmiar zbioru E to $n - 1$, a dla każdego elementu $e \in E$ rozpatrywane są dwa warunki.

Działanie procedury WYZNACZNAJLEPSZYWARUNEK przedstawione zostało na listingu 2. Dla każdego warunku ze zbioru C tworzona jest nowa reguła poprzez dodanie go do aktualnej postaci przesłanki. Następnie, na podstawie pokrycia tej nowo utworzonej reguły, wyznaczana jest wartość miary jakości, użytej do oceny poszczególnych warunków. W rzeczywistości więc oceniane są nie tyle warunki, co sama reguła po ich dodaniu. W przypadku, gdy wartość miary jest taka sama dla dwóch różnych warunków, zostaje wybrany ten, po którego dodaniu reguła pokrywa największą liczbę przykładów (linia 8).

Faza przycinania polega na iteracyjnym usuwaniu warunków z przesłanki reguły. W każdej iteracji wybierany jest warunek, którego usunięcie maksymalizuje wartość miary oceniającej przycinaną regułę. Algorytm dopuszcza użycie dwóch różnych miar dla fazy wzrostu i przycinania reguły. Usuwanie warunków trwa do momentu, gdy

Algorytm 1 Wzrost reguły w algorytmie RuleKit

Wejście:
D — zbiór treningowy,
D_u — zbiór niepokrytych dotąd przykładów,
D_r — przykłady pokryte przez r,
Wyjście: r —reguła.

function WZRASTAJ(D, D_u, T)
  $r \leftarrow \varnothing$
  repeat
    $D_r \leftarrow$ OBLICZPRZYKLADYPOKRYTE(r, D)
    $C \leftarrow$ WYZNACZPULEWARUNKOW(D, D_r, D_u)
    $c_{best} \leftarrow$ WYZNACZNAJLEPSZYWARUNEK(C, r, D)
    if $c_{best} \neq \varnothing$ **then**
      $r \leftarrow r \ \wedge c_{best}$
    end if
  until $c_{best} \neq \varnothing$
  return r
end function

Algorytm 2 Szukanie najlepszego warunku w algorytmie RuleKit

Wejście:
C — zbiór warunków do oceny,
r — aktualna postać reguły,
D — zbiór treningowy.
Wyjście: c_{best} — najlepszy warunek.

function WYZNACZNAJLEPSZYWARUNEK(C,r,D)
  $c_{best} \leftarrow \varnothing$ ▷ najlepszy warunek
  $q_{best} \leftarrow -\infty$ ▷ jakość najlepszego warunku
  $cov_{best} \leftarrow \varnothing$ ▷ liczba przykładów pokrywanych przez najlepszy warunek
  for $c \in C$ **do**
    $r_c \leftarrow r \ \wedge \ c$ ▷ dodanie warunku do reguły
    $q \leftarrow$ OBLICZJAKOSCREGULY(r_c, D)
    $found\_better \leftarrow (q > q_{best})$ ▷ porównanie jakości reguły
    if $\neg found\_better \wedge q \ = q_{best}$ **then**
      ▷ porównanie liczby pokrytych przykładów
      $cov \leftarrow$OBLICZLICZBEPOKRYTYCHPRZYKLADOW(c, D)
      $found\_better \leftarrow cov > cov_{best}$
    end if
    if $found\_better$ **then**
      $c_{best} \leftarrow c$, $q_{best} \leftarrow q$, $cov \leftarrow cov_{new}$
    end if
  end for
  return c_{best}
end function

dalsze redukcje nie poprawiają jakości reguły lub gdy przesłanka zawiera tylko jeden warunek.

Algorytm RuleKit umożliwia generowanie reguł dla trzech typów problemów: klasyfikacji, regresji i analizy przeżycia. Proces indukcji reguł przebiega dla każdego z nich

w podobny sposób z niewielkimi różnicami, które jednak są warte dokładniejszego opisu.

W przypadku klasyfikacji algorytm uruchamiany jest osobno dla każdej z klas. Za zbiór niepokrytych przykładów służą wszystkie przykłady należące do zadanej klasy. W rezultacie otrzymujemy osobne zbiory reguł dla każdej z klas, które są następnie scalane w jeden, wynikowy zbiór reguł. Dla pozostałych typów problemów proces indukcji reguł odbywa się jednorazowo, na całym zbiorze treningowym. Dodatkową różnicą jest sposób, w jaki ustalana jest konkluzja reguły. Dla problemu klasyfikacji jest ona ustalona na początku i wskazuje na klasę, w ramach której generowane są reguły. W przypadku regresji i analizy przeżycia konkluzja obliczana jest dynamicznie na podstawie pokrycia reguły. Dla regresji stanowi ona wartość średnią lub medianę (w zależności od ustawień parametrów) wartości atrybutu decyzyjnego dla przykładów pokrytych przez regułę. W analizie przeżycia konkluzje reguły stanowi estymator Kaplana-Meiera, dopasowany do zbioru pokrytych przez regułę przykładów.

Miary jakości używane w procesie wzrostu i przycinania bazują na liczbie pokrytych przez regułę przykładów pozytywnych i negatywnych (p i n). Pojęcie przykładu negatywnego i pozytywnego odnosi się ściśle do problemu klasyfikacji binarnej. W przypadku problemów wieloklasowych, za przykłady negatywne dla danej reguły uznaje się wszystkie nienależące do określonej klasy, na jaką wskazuje jej konkluzja. Pojęcia tego nie sposób jednak zastosować bezpośrednio dla regresji i analizy przeżycia. Algorytm RuleKit stosuje w tym celu pewien trik, polegający na przekształceniu problemu w problem klasyfikacji binarnej [162]. W przypadku regresji obliczenie p i n rozpoczyna się od wyznaczenia zbioru pokrytych przez regułę przykładów D_c. Następnie obliczana jest wartość konkluzji reguły y oraz przedział ufności zdefiniowany jako $[y - \mathrm{std}(\delta(D_c)),\ y + \mathrm{std}(\delta(D_c))]$. p jest tutaj liczbą wszystkich przykładów pokrywanych przez regułę, należących do przedziału ufności, n z kolei określone jest jako $|D_c| - p$. W przypadku problemu przeżycia, wszystkie przykłady pokryte przez regułę uznawane są za pozytywne. Powoduje to jednak problem z oceną jej jakości, jako że RuleKit dąży w fazach wzrostu i przycinania do uzyskania reguły z maksymalnym p i minimalnym n. Z punktu widzenia algorytmu idealną regułą w tym wypadku jest taka, która pokrywa cały zbiór treningowy. Aby rozwiązać ten problem, w przypadku analizy przeżycia, stosowana jest zawsze jedna specyficzna miara jakości, niewykorzystująca wartości p i n. Zamiast tego porównuje ona dwa estymatory Kaplana-Meira. Pierwszy, oznaczony jako Kc, dopasowywany jest do zbioru pokrytych przez regułę przykładów oraz drugi K_u dopasowany do pozosta-

łej części zbioru treningowego. Oba estymatory są porównywane z użyciem testu statystycznego logrank [109]. Za jakość reguły uznawana jest wartość wyrażenia $1 - p$, gdzie p jest p-wartością, uzyskaną w ramach testu. Innymi słowy algorytm stara się poszukiwać reguł, dla których estymator Kaplana-Meira jest jak najbardziej różny od estymatora dla przykładów niepokrytych.

1.2 Rodzaje generowanych warunków złożonych

ComplexConditions wprowadza do wyżej opisanego algorytmu RuleKit możliwość generowania reguł zawierających różne typy warunków złożonych. Poniżej znajduje się dokładny opis każdego z nich. Wyjaśniony zostanie także sposób, w jaki są one generowane oraz poczynione przez autora założenia i ograniczenia.

Warunki przedziałów: Algorytm rozszerza pulę warunków ocenianych dla atrybutów numerycznych. W przypadku algorytmu RuleKit brane pod uwagę były jedynie warunki w postaci $a_i <$ wartość oraz $a_i \geqslant$ wartość. W przypadku ComplexConditions dodatkowo rozpatrywane są warunki przedziałów w postaci $a_i \in < [l, r)$, gdzie $r > l \ \wedge \ l \neq -\infty \ \wedge r \ \neq \infty \ \wedge l \ \in E$. Zbiór E wyznaczany jest identycznie jak w przypadku algorytmu RuleKit (patrz równanie (6.1)). Wartość r to wszystkie wartości atrybutu a_i, pojawiające się w rozpatrywanym zbiorze przykładów, które spełniają wcześniej wspomniane ograniczenia.

Aby obliczyć liczbę rozpatrywanych warunków przedziałów dla przypadku pesymistycznego, należy najpierw rozważyć, ile możliwych przedziałów $[l, r)$ da się utworzyć dla posortowanych rosnąco n wartości. Lewa granica przedziału może być dowolną z nich, z wyjątkiem ostatniej. Istnieje więc $n - 1$ możliwych wartości dla l. W przypadku prawej granicy r, jej wybór zależny jest od wartości l, ponieważ musi zostać spełniony warunek $l < r$. Mamy tutaj do czynienia z szeregiem arytmetycznym $S_{n-1} = (n - 1) + (n - 2) + \ldots + 1$. Liczbę możliwych przedziałów można zatem obliczyć, korzystając z wzoru na sumę szeregu arytmetycznego. Wynosi ona $\frac{n^2-n}{2}$.

Przykładowy warunek przedziału zaprezentowany został poniżej.

$$\text{wiek} \in [18, 65)$$

Negacje warunków: Rozpatrywane są one dla wszystkich warunków prostych, utworzonych na atrybutach nominalnych, przyjmując postać $a_i \neq$ wartość. Dodatkowo oceniane są negacje wyżej wspomnianych warunków przedziałów.

Poniżej znajdują się przykładowe zanegowane warunki, które mogą pojawiać się w przesłankach reguł.

$$\text{wiek} \notin [18, 65),\ \text{grupa\_krwi} \neq \{ARh+\}$$

Warunki relacji atrybutów: Algorytm ocenia możliwe do utworzenia warunki relacji atrybutów zgodnie z przedstawioną w rozdziale 4, sekcji 3 definicją. Dla atrybutów nominalnych rozpatrywane są relacje równości i nierówności. W przypadku atrybutów numerycznych są to relacje równości oraz mniejszości i większości. Dodatkowo dla atrybutów nominalnych została wprowadzona specjalna, dodatkowa forma warunku w postaci $a_i = a_j = ...a_N$. W celu uzyskania rozsądnych czasów obliczeń oraz możliwie łatwych w interpretacji reguł, maksymalna wartość N została ustalona jako 3.

Warunki relacji atrybutów generowane są pomiędzy wszystkimi numerycznymi atrybutami w zbiorze treningowym oraz pomiędzy wszystkimi nominalnymi, które posiadają wspólny zbiór możliwych wartości.

Dla m atrybutów można na ich podstawie utworzyć $m^2 - m$ par. Wykorzystany został tutaj wzór na liczbę wariancji bez powtórzeń, jako, że kolejność elementów jest znacząca. W przypadku pesymistycznym, zakładając, że wszystkie m atrybuty w zbiorze danych są numeryczne, dla każdej pary testowane są trzy niesymetryczne relacje. Korzystając ze wzoru na liczbę wariacji bez powtórzeń, możliwe jest więc utworzenie dla nich maksymalnie $3m^2 - 3m$ warunków relacji atrybutów. Sytuacja wygląda zgoła odmiennie w przypadku atrybutów nominalnych, gdzie testujemy jedynie dwie relacje. Uwzględniamy tutaj nie tylko pary, ale także trójki atrybutów. Relacje równości oraz nierówności są jednak symetryczne. Bez znaczenia jest zatem kolejność elementów w parach i trójkach. Mamy więc tutaj do czynienia z kombinacjami bez powtórzeń. Możliwe jest zatem utworzenie $\binom{m}{2} = \frac{m^2-m}{2}$ takich par oraz $\binom{n}{3} = \frac{m^3-3m^2+2m}{6}$ trójek. Sumarycznie, dla takiego przypadku, ocenionych zostanie $\frac{m^3-m}{3}$ warunków, ponieważ dla każdej pary i trójki ocenione zostaną dwie relacje.

Przykłady warunków relacji atrybutów kolejno dla atrybutu nominalnego i numerycznych przedstawione zostały poniżej.

$$\text{cena\_produktu\_a} < \text{cena\_produktu\_b},$$

$$\text{ocena\_eksperta\_a} = \text{ocena\_eksperta\_b} = \text{ocena\_eksperta\_c}$$

Warunki zbioru dyskretnego: Stanowią one specyficzny przypadek warunków wewnętrznej alternatywy (patrz podrozdział 3), gdzie wszystkie warunki składowe są warunkami prostymi opartymi na jednym, wybranym atrybucie nominalnym. W celu zachowania czytelności reguł oraz rozsądnych czasów działania, maksymalna liczba warunków w alternatywie została ograniczona do trzech.

Dla wariantu pesymistycznego dla n możliwych wartości atrybutu jest możliwe utworzenie $\binom{n}{2} + \binom{n}{3} = \frac{n^3-n}{6}$ warunków tego typu. Obliczenia są tutaj identyczne jak w przypadku wcześniej wspominanych warunków relacji atrybutów dla atrybutów nominalnych. Ponownie tworzone są pary i trójki, tym razem jednak nie generujemy dla nich dwóch warunków, lecz jeden. Dlatego też końcowy wynik należy podzielić przez dwa.

Poniżej przedstawiony został przykład warunku zbioru dyskretnego mogący pojawić się w indukowanych regułach.

$$\text{dzień\_tygodnia} \in \{\text{piątek, sobota, niedziela}\}$$

Warunki wewnętrznej alternatywy dla atrybutów numerycznych: Warunki tego typu są odpowiednikiem wyżej wymienionych warunków zbioru dyskretnego dla atrybutów numerycznych. Mają one postać $c_1 \vee c_2 \vee \ldots \vee c_N$, gdzie warunki $c_1 \ldots c_N$ są warunkami prostymi lub warunkami przedziału opartymi na jednym atrybucie. Maksymalna liczba składników alternatywy N kontrolowana jest przez parametr.

W przypadku wcześniej opisywanych typów warunków złożonych, w trakcie indukcji reguł generowane, a następnie oceniane były wszystkie możliwe do utworzenia warunki. Dla tego konkretnego typu warunków zostało jednak zastosowane odmienne podejście. Rozpatrywanie wszystkich możliwych kombinacji pełnej puli prostych warunków numerycznych oraz warunków przedziału byłoby wysoce nieefektywne. Choć we wcześniejszych rozważaniach często pojawiał się scenariusz, w którym atrybut nominalny posiada n unikalnych wartości, w rzeczywistych zbiorach danych jest on rzadziej spotykany. Atrybuty nominalne wskazują najczęściej na przynależność przykładu do pewnej skończonej liczby kategorii [78]. Liczba takich kategorii zwykle jest znacząco mniejsza od liczby wierszy w zbiorze danych. Tak skrajna sytuacja ma większe szanse wystąpienia w przypadku kolumn numerycznych. Nietrudno wyobrazić sobie taki przypadek dla np. danych dotyczących pomiarów wielkości fizycznych, gdzie wartości w każdym wierszu mogą być inne. Dodatkowo liczba możliwych do utworzenia warunków prostych dla atrybutu numerycznego, jak zostało to wykazane,

jest kwadratowo wyższa niż dla nominalnego. Wszystko to przemawia za użyciem odmiennej techniki do generowania warunków wewnętrznych alternatyw dla atrybutów numerycznych. W tym celu została zmodyfikowana procedura wzrostu reguły zgodnie z listingiem 3. Po wybraniu najlepszego warunku c_{best} z puli wszystkich pozostałych ich rodzajów, algorytm przechodzi do generowania puli warunków wewnętrznych alternatyw, z użyciem przeszukiwania wiązką. Krok ten jest wykonywany tylko, gdy c_{best} jest warunkiem prostym lub warunkiem przedziału opartym na atrybucie numerycznym a. Na początku tworzone jest N alternatyw z jednym warunkiem c_{best}, gdzie N jest rozmiarem wiązki określonym parametrem algorytmu. Następnie z puli C wybierane są wszystkie warunki oparte na atrybucie a, mogące pojawić się w alternatywie, tworząc zbiór C'. Algorytm rozszerza następnie każdą z N alternatyw w wiązce, generując nowe poprzez dodanie do nich pojedynczego warunku z puli C. Każda nowo utworzona alternatywa jest następnie oceniona z użyciem miary jakości reguły w sposób identyczny jak pozostałe rodzaje warunków. Z wszystkich nowo wygenerowanych rozwiązań wybieranych jest N z najlepszą jakością, które stanowią w kolejnej iteracji nową wiązkę. Po wykonaniu określonej liczby iteracji przeszukiwania, która określona jest parametrem algorytmu, zwracana jest wynikowa pula warunków wewnętrznych alternatyw. W końcowym kroku sprawdzane jest, czy dodanie któregokolwiek z nich prowadzi do uzyskania lepszej reguły niż ta stworzona z użyciem c_{best}. Jeżeli tak, wybierany jest on jako nowa wartość c_{best} i dodawany do reguły.

Dla każdej z iteracji, dla każdego elementu wiązki rozpatrywane są wszystkie warunki proste oraz warunki przedziału oparte na wybranym atrybucie, a także ich negacje. Po zsumowaniu daje to $n^2 + 3n - 4$ warunków, przy założeniu, że atrybut a posiada n unikalnych wartości. Aby uzyskać liczbę wszystkich ocenianych alternatyw, należy przemnożyć wynik przez liczbę iteracji oraz rozmiar wiązki. Są one jednak zmienne i określane przez parametr algorytmu. Rząd wartości zostanie jednak co do zasady kwadratowy.

Przykładowy warunek wewnętrznej alternatywy dla atrybutu numerycznego zaprezentowany został poniżej.

$$(\text{wiek} \in [25, 40) \ \vee \ \text{wiek} > 65)$$

1.3 Generowanie reguł z warunkami złożonymi algorytmem ComplexConditions

Proces generowania reguł w algorytmu ComplexConditions przebiega podobnie jak w przypadku algorytmu RuleKit. Wszystkie rodzaje warunków złożonych, z wyjątkiem wewnętrznych alternatyw dla atrybutów numerycznych, dodawane są do puli warunków ocenianych w trakcie wzrostu. Pula ta jest tworzona na bazie wybranych przez użytkownika rodzajów warunków, które mają być rozpatrywane (listing 3 linia 5).

Algorytm 3 Wzrost reguły w algorytmie ComplexConditions

Wejście:
D — zbiór treningowy,
D_u — zbiór niepokrytych dotąd przykładów,
D_r — przykłady pokryte przez r,
T —lista rodzajów warunków złożonych do uwzględnienia.
Wyjście: r — reguła.

```
1: function WZRASTAJ(D, D_u, D_r, T)
2:     r ← ∅
3:     repeat
4:         D_r ← OBLICZPRZYKLADYPOKRYTE(r, D)
5:         C ← WYZNACZPULEWARUNKOW(T, D, D_r, D_u)
6:         c_best ← WYZNACZNAJLEPSZYWARUNEK(C, r, D)
7:         ▷ generowanie warunków wewnętrznych alternatyw
8:         if c_best ≠ ∅ ∧ T includes WEW_ALTERNATYWY then
9:             C_alt ← GENERUJALTERNATYWY(C, c_best, D, D_r, D_u)
10:            c_best ← WYZNACZNAJLEPSZYWARUNEK({c_best} ∪ C_alt, r, D)
11:        end if
12:        if c_best ≠ ∅ then
13:            r ← r ∧ c_best
14:        end if
15:    until c_best ≠ ∅
16:    return r
17: end function
```

Warunki wewnętrznych alternatyw dla atrybutów numerycznych są rozpatrywane po wybraniu najlepszego warunku ze wszystkich pozostałych rodzajów (listing 3 linia 8). W przypadku gdy warunek c_{best} nie jest oparty o atrybut numeryczny, zbiór C_{alt} będzie zbiorem pustym.

W przypadku algorytmu ComplexConditions, biorąc pod uwagę dość bogaty zestaw stosowanych przez niego warunków złożonych, możliwe jest uzyskanie kilku warunków różnego typu, opisujących dokładnie tę samą zależność w danych. Rozważmy następujący przykład. Atrybut *kolor_oczu* jest atrybutem nominalnym, przyjmującym

cztery możliwe wartości: niebieskie, zielone, brązowe i piwne. W trakcie wzrostu reguły będą rozpatrywane między innymi następujące warunki:

- $kolor\_oczu \in \{\text{niebieskie, brązowe, piwne}\}$,
- $kolor\_oczu \neq \{zielone\}$.

Pierwszy z nich jest przykładem warunku zbioru dyskretnego, drugi stanowi negację warunku prostego. Oba z nich pokrywają dokładnie te same przykłady. Warunki te, oceniane w sposób identyczny jak ma to miejsce w algorytmie RuleKit, byłyby równoważne. Wybrany zostałby pierwszy z nich, który został poddany ocenie. Warunek drugi reprezentuje tutaj jednak krótszą i bardziej czytelną formę zapisu, bardziej pożądaną w kontekście odkrywania wiedzy.

W nawiązaniu do wyżej opisanego problemu, zmodyfikowana została procedura WYZNACZNAJLEPSZYWARUNEK (patrz listing 4). W przypadku gdy dla obydwu warunków ich dodanie do przesłanki prowadzi do uzyskania reguły o identycznej jakości i pokryciu, porównywane są dla nich kolejno dwa dodatkowe kryteria. Pierwszym z nich jest liczba użytych w warunku atrybutów (linia 14). Dla przykładu, rozpatrując dwa warunki: $b = a$ oraz $b = v_1$, zostanie wybrany drugi z nich, ponieważ jest on oparty na jednym atrybucie.

Jeżeli zarówno jakość, jak i pokrycie oraz liczba użytych atrybutów będą identyczne dla dwóch różnych warunków, wybierany jest ten o najniższej wadze (linia 20). Wagi zostały w sposób arbitralny przypisane poszczególnym rodzajom warunków złożonych, starając się oddać ich subiektywny stopień opisowości i łatwości w interpretacji. Poniżej przedstawione zostały ich wartości dla różnych typów warunków:

- 0 dla warunków wewnętrznych alternatyw.
- 1 dla warunków prostych oraz ich negacji;
- 2 dla warunków relacji atrybutów oraz zbiorów dyskretnych;

Żadna z wyżej wspomnianych modyfikacji nie wpływa na przebieg fazy przycinania reguły. Poszczególne warunki składowe alternatyw nie są przycinane osobno. Usunięciu może zostać poddana jedynie cała alternatywa. Algorytm ComplexConditions obsługuje problemy regresji i przeżycia w sposób identyczny, jak ma to miejsce w przypadku metody RuleKit.

Algorytm 4 Szukanie najlepszego warunku w algorytmie ComplexConditions

Wejście:
C — zbiór warunków do oceny,
r — aktualna postać reguły,
D — zbiór treningowy.
Wyjście: c_{best} — najlepszy warunek.

function WyznaczNajlepszyWarunek(C,r,D)
  $c_{best} \leftarrow \varnothing$ ▷ najlepszy warunek
  $q_{best} \leftarrow -\infty$ ▷ jakość najlepszego warunku
  $cov_{best} \leftarrow \varnothing$ ▷ liczba przykładów pokrywanych przez najlepszy warunek
  for $c \in C$ **do**
    $r_c \leftarrow r \wedge c$ ▷ dodanie warunku do reguły
    $q \leftarrow$ ObliczJakoscReguly(r_c, D)
    $jest\_lepszy \leftarrow (q > q_{best})$ ▷ porównanie jakości reguły
    if $\neg jest\_lepszy \wedge q = q_{best}$ **then**
      ▷ porównanie liczby pokrytych przykładów
      $cov \leftarrow$ ObliczLiczbePokrytychPrzykladow(c, D)
      $jest\_lepszy \leftarrow cov > cov_{best}$
      if $\neg jest\_lepszy$ **then**
        ▷ preferuj warunki oparte na mniejszej liczbie atrybutów
        $jest\_lepszy \leftarrow$ (
            PoliczAtrybuty(c) < PoliczAtrybuty(c_{best})
)
        if $\neg jest\_lepszy$ **then**
          ▷ preferuj warunki z wyższą wagą
          $jest\_lepszy \leftarrow$ (Waga(c) < Waga(c_{best}))
        end if
      end if
    end if
    if $jest\_lepszy$ **then**
      $c_{best} \leftarrow c$, $q_{best} \leftarrow q$, $cov \leftarrow cov_{new}$
    end if
  end for
  return c_{best}
end function

1.4 Parametry algorytmu

Większość parametrów algorytmu ComplexConditions jest identyczna jak w przypadku metody RuleKit, na której bazuje. Posiada on jednak kilka dodatkowych, pozwalających m .in. kontrolować to, jakie warunki złożone mają zostać wykorzystane w trakcie indukcji. Poniżej znajduje się zestawienie wszystkich parametrów algorytmu wraz z opisem ich działania.

Parametry wspólne z algorytmem RuleKit

- ***min_rule_covered*** – Dodatnia liczba całkowita, określająca minimalną liczbę nowych przykładów, które musi pokryć nowo utworzona reguła; domyślnie 5.

- ***induction_measure*** – Miara jakości stosowana do oceny reguły w fazie wzrostu; domyślnie miara Correlation (patrz równanie 6.3).
- ***pruning_measure*** – Miara jakości stosowana do oceny reguły w fazie przycinania; domyślnie miara Correlation (patrz równanie 6.3).
- ***voting_measure*** – Miara jakości stosowana do rozwiązywania konfliktów pojawiających się przy predykcji w czasie głosowania; domyślnie miara Correlation (patrz równanie 6.3).
- ***max_growing*** – Nieujemna liczba całkowita określająca maksymalną liczbę warunków w regule; domyślnie zero (oznacza brak ograniczenia).
- ***enable_pruning*** – Flaga włączająca i wyłączająca fazę przycinania reguły; domyślnie włączona.
- ***ignore_missing*** – Flaga określająca czy przykłady zawierające wartości brakujące powinny być brane pod uwagę przy treningu; domyślnie wartość brakująca atrybutu sprawia, że warunek na nim oparty jest zawsze niespełniony,
- ***max_uncovered_fraction*** – Ułamek z zakresu $[0, 1]$ określający, jaka maksymalna część zbioru treningowego może zostać niepokryta przez zbiór reguł; domyślnie 0.
- ***select_best_candidate*** – Flaga kontrolująca czy po zakończeniu fazy wzrostu wybierana jest postać reguły po wykonaniu wszystkich iteracji, czy też taka, która osiągnęła najwyższą wartość miary w trakcie całego procesu; pierwsze z wymienionych zachowań jest domyślne.

$$C2 = Coleman \cdot \frac{P+p}{2P} \quad gdzie \quad Coleman = \frac{Np - Pn}{N(p+n)} \quad [88, 193] \tag{6.2}$$

Działanie parametru select_best_candidate zasługuje tutaj na dłuższe wyjaśnienie. Jego ustawienie może bowiem wpływać na rezultat fazy wzrostu reguły. W jej trakcie algorytm RuleKit dodaje kolejne warunki do przesłanki wzrastanej reguły aż do spełnienia kryterium stopu. Stara się on przy tym maksymalizować wartość wybranej miary jakości (parametr induction_measure). Proces ten trwa jednak aż do spełnienia kryterium stopy (patrz pseudokod 1). Oznacza to, że jakość reguły po dodaniu kolejnego warunku nie musi zawsze rosnąć. Przy domyślnym ustawieniu algorytmu

RuleKit i wyłączonej fladze select_best_candidate, na koniec fazy wzrostu zwrócona zostanie postać reguły po osiągnięciu kryterium stopu. Nie jest to jednak koniecznie postać osiągająca najwyższą wartość miary jakości. Włączenie tejże flagi powoduje, że algorytm zapamiętuje wszystkie pośrednie formy reguły po dodaniu każdego z kolejnych warunków wraz z ich jakością. Na koniec fazy wzrostu zwracana jest wtedy ta, dla której była ona najwyższa. Działanie flagi w przypadku algorytmu ComplexConditions jest identyczne jak dla RuleKit.

Wartości domyślne większości parametrów są identyczne jak te stosowane w algorytmie RuleKit. Istotna zmiana dotyczy domyślnej miary jakości wykorzystywanej przy wzroście, przycinaniu i głosowaniu. Dla RuleKit była to miara Correlation [65] (równanie 6.3). W przypadku metody ComplexConditions zdecydowano się na wykorzystanie popularnej w literaturze miary C2 (równanie 6.2), ponieważ faworyzuje ona reguły o wysokiej precyzji i umiarkowanym pokryciu [88, 193].

$$\text{Correlation} = \frac{pN - Pn}{\sqrt{PN(p+n)(P-p+N-n)}} \quad [65] \tag{6.3}$$

Parametry unikalne dla metody ComplexConditions:

- ***discrete_set_conditions_enabled*** – Flaga kontrolująca czy algorytm wykorzystuje w trakcie indukcji warunki zbioru dyskretnego; domyślnie włączone.
- ***negated_conditions_enabled*** – Flaga kontrolująca czy algorytm wykorzystuje w trakcie indukcji warunki zanegowane; domyślnie włączone.
- ***intervals_conditions_enabled*** – Flaga kontrolująca czy algorytm wykorzystuje w trakcie indukcji warunki przedziałów; domyślnie włączone.
- ***numerical_attributes_conditions_enabled*** – Flaga kontrolująca czy algorytm wykorzystuje w trakcie indukcji warunki relacji atrybutów dla atrybutów numerycznych; domyślnie włączone.
- ***nominal_attributes_conditions_enabled*** – Flaga kontrolująca czy algorytm wykorzystuje w trakcie indukcji warunki relacji atrybutów dla atrybutów nominalnych; domyślnie włączone.
- ***inner_alternatives_enabled*** – Flaga kontrolująca czy algorytm wykorzystuje w trakcie indukcji warunki wewnętrznych alternatyw dla atrybutów numerycznych; domyślnie włączone.

- ***inner_alternatives_search_beam_size*** – Szerokość wiązki wykorzystywana podczas poszukiwań warunków wewnętrznych alternatyw dla atrybutów numerycznych; domyślnie 3.
- ***inner_alternatives_search_max_iterations*** – Maksymalna liczba iteracji poszukiwań warunków wewnętrznych alternatyw dla atrybutów numerycznych; domyślnie 5.

1.5 Przykłady działania na zbiorach syntetycznych

Wprowadzenie zestawu warunków złożonych do algorytmu pozwala mu wyszukiwać nowe zależności w danych, dla których opis z użyciem warunków prostych wymagałby użycia większej liczby, gorszych jakościowo reguł. Pozwala to w niektórych przypadkach uzyskać prostsze i mniej liczne zbiory reguł. Oczywiście dzieje się tak jedynie wtedy, gdy tego typu zależności są faktycznie obecne w danych treningowych.

W celu zaprezentowania potencjału metody ComplexConditions do generowania bardziej zwięzłych zbiorów reguł przygotowano sześć syntetycznych zbiorów danych. Każdy z nich zawiera dwa atrybuty a oraz b, oraz dwie klasy: pozytywną i negatywną. Zbiory te zostały przedstawione na rysunku 6.1. Kolorem czerwonym została przedstawiona klasa pozytywna, kolorem niebieskim zaś klasa negatywna. Zależności opisujące klasę pozytywną dla każdego ze zbiorów zostały przedstawione poniżej:

(a) klasa 1 gdy $a \in [3, 7]$, dla pozostałych przykładów klasa 0,

(b) klasa 1 gdy $a \in [3, 5]$ lub $a \in [7, 8]$, dla pozostałych przykładów klasa 0,

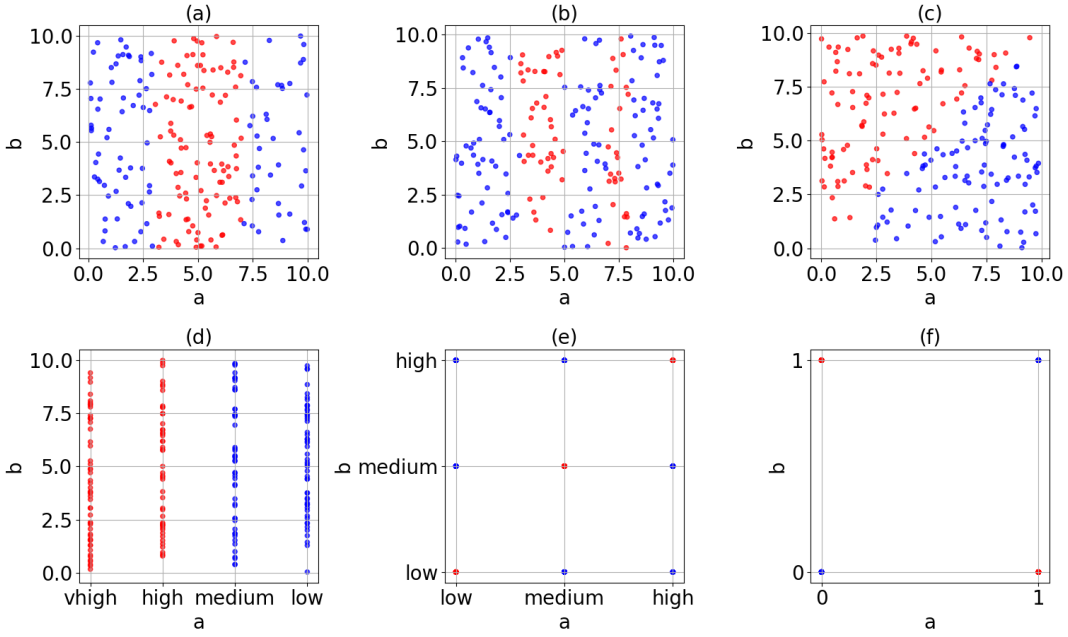
(c) klasa 1 dla przykładów, gdzie $a_i < b_i$, dla pozostałych klasa 0,

(d) klasa 1 gdy $a \in \{\text{medium, low}\}$, dla pozostałych przykładów 0,

(e) klasa 1 dla przykładów, gdzie $a_i = b_i$, dla pozostałych klasa 0,

(f) problem XOR.

Dla każdego z sześciu zbiorów syntetycznych zostały wygenerowane dwa zbiory reguł. Pierwszy z użyciem niezmodyfikowanego algorytmu RuleKit i drugi z użyciem metody ComplexConditions. Podczas prób została wykorzystana oficjalna, publicznie



Rysunek 6.1: Przykładowe syntetyczne zbiory danych

dostępna implementacja tego algorytmu[1]. Uzyskane zbiory reguł zostały zaprezentowane w tabeli 6.1. Zastosowano w niej, jak i w późniejszych rozdziałach, skrótową formę zapisu. Pominięte zostało tutaj słowo JEŻELI. Decyzja, na jaką wskazuje reguła, została zapisana po symbolu strzałki w prawo.

W przypadku pierwszego zbioru danych ComplexConditions zdołał opisać go z użyciem jedynie dwóch reguł, w przeciwieństwie do RuleKit, który potrzebował do tego trzech reguł. Uzyskany przez ComplexConditions opis jest w tym wypadku zwięźlejszy i bardziej czytelny.

Zbiór reguł dla zbioru (b), uzyskany z użyciem algorytmu RuleKit składa się z pięciu reguł. Dzięki zastosowaniu warunków wewnętrznych alternatyw, zbiór wygenerowany przez ComplexConditions zawiera tylko dwie reguły. Każda z nich posiada alternatywę dwóch warunków w przesłance.

Dla zbioru (c) algorytm RuleKit wygenerował aż 7 reguł, z czego część zawierała więcej niż jeden warunek w przesłance. Opis uzyskany metodą ComplexConditions posiada jedynie dwie reguły, posiadające pojedynczy warunek.

Również dla zbioru (d) zbiór reguł uzyskany algorytmem ComplexConditions jest mniej liczny (dwie reguły zamiast czterech). Jak można zauważyć, opisują

[1]Link do oficjalnej implementacji algorytmu RuleKit: `https://github.com/adaa-polsl/RuleKit`, [dostęp 16.09.2025].

	RuleKit	**ComplexConditions**
(a)	**r1:** a $\leqslant$ 3.12 $\rightarrow$ {0} **r2:** a $\geqslant$ 6.96 $\rightarrow$ {0} **r3:** a $\geqslant$ 3.12 $\wedge$ a $<$ 6.96 $\rightarrow$ {1}	**r1:** a $\notin$ [3.12, 7.04) $\rightarrow$ {0} **r2:** a $\in$ [3.12, 7.04) $\rightarrow$ {1}
(b)	**r1:** a $\leqslant$ 2.97 $\rightarrow$ {0} **r2:** a $\geqslant$ 7.99 $\rightarrow$ {0} **r3:** a $\geqslant$ 5.15 $\wedge$ a $<$ 7.03 $\rightarrow$ {0} **r4:** a $\geqslant$ 6.98 $\wedge$ a $<$ 7.99 $\rightarrow$ {1} **r5:** a $\geqslant$ 2.97 $\wedge$ a $<$ 4.92 $\rightarrow$ {1}	**r1:** a $\notin$ [2.97, 8.05) $\vee$ a $\in$ [4.92, 7.02) $\rightarrow$ {0} **r2:** a $\in$ [2.97, 5.03) $\vee$ a $\in$ [6.98, 8.05)$\rightarrow$ {1}
(c)	**r1:** b $\geqslant$ 7.02 $\rightarrow$ {0} **r2:** a $\leqslant$ 1.52 $\rightarrow$ {0} **r3:** b $\geqslant$ 1.89 $\wedge$ a $\leqslant$ 2.48 $\rightarrow$ {0} **r4:** b $\geqslant$ 5.36 $\wedge$ a $\leqslant$ 5.71 $\rightarrow$ {0} **r5:** b $\leqslant$ 7.59 $\wedge$ a $\geqslant$ 5.71 $\rightarrow$ {1} **r6:** b $\leqslant$ 2.26 $\wedge$ a $\geqslant$ 1.17 $\rightarrow$ {1} **r7:** b $\leqslant$ 5.43 $\wedge$ a $\geqslant$ 4.08 $\rightarrow$ {1}	**r1:** a $<$ b $\rightarrow$ {0} **r2:** a $>$ b $\rightarrow$ {1}
(d)	**r1:** a = {vhigh} $\rightarrow$ {0} **r2:** a = {high} $\rightarrow$ {0} **r3:** a = {low} $\rightarrow$ {1} **r4:** a = {medium} $\rightarrow$ {1}	**r1:** a = {vhigh, high} $\rightarrow$ {0} **r2:** a = {medium, low} $\rightarrow$ {1}
(e)	**r1:** b = {high} $\wedge$ a = {medium} $\rightarrow$ {0} **r2:** b = {high} $\wedge$ a = {low} $\rightarrow$ {0} **r3:** b = {low} $\wedge$ a = {medium} $\rightarrow$ {0} **r4:** b = {medium} $\wedge$ a = {medium} $\rightarrow$ {1} **r5:** b = {low} $\wedge$ a = {low} $\rightarrow$ {1} **r6:** b = {high} $\wedge$ a = {high} $\rightarrow$ {1}	**r1:** b $\neq$ a $\rightarrow$ {0} **r2:** b = a $\rightarrow$ {1}
(f)	**r1:** a = {0} $\wedge$ b = {0} $\rightarrow$ {0} **r2:** a = {1} $\wedge$ b = {1} $\rightarrow$ {0} **r3:** a = {0} $\wedge$ b = {1} $\rightarrow$ {1} **r4:** a = {1} $\wedge$ b = {0} $\rightarrow$ {1}	**r1:** b = a $\rightarrow$ {0} **r2:** b $\neq$ a $\rightarrow$ {1}

Tabela 6.1: Reguły z prostymi i złożonymi warunkami (zmienione przedziały i notacja)

one dokładnie tę samą zależność w sposób bardziej zwięzły, a przy tym łatwy do zrozumienia.

Ta sama sytuacja powtarza się dla zbioru (e). Klasa pozytywna występuje tam, gdzie wartości a i b są sobie równe. Ta prosta pozornie zależność w przypadku algorytmu RuleKit wymagała aż 6 reguł, by ją opisać. Dla metody ComplexConditions wystarczająca była pojedyncza reguła dla każdej z klas.

Zbór (f) opisuje klasyczny binarny problem XOR dla dwóch zmiennych. Dla tego konkretnego zbioru danych, algorytm RuleKit wygenerował zbiór z czterema regułami. Opisują one wszystkie możliwe kombinacje wartości atrybutów. Metoda

ComplexConditions zdołała opisać ten sam problem z użyciem dwóch, czytelnych reguł.

Na podstawie zaprezentowanych wyników można zaobserwować, jak zastosowanie bardziej złożonych warunków w przesłance może pomóc generować krótsze, bardziej zwięzłe, a co za tym idzie, interpretowalne zbiory reguł. Zaprezentowane przypadki pokazują także, że zapis z użyciem warunków prostych może być w niektórych przypadkach nadmiernie rozbudowany i nie najlepiej oddawać charakter danych.

Warto tutaj dodać, że ukazane w tej sekcji przykłady i wyniki skupiają się na zaprezentowaniu głównych zalet metody ComplexConditions. Dokładniejsze wyniki eksperymentów przeprowadzonych dla większej liczby zbiorów rzeczywistych i benchmarkowych, dostępne są w rozdziale 7.

1.6 Analiza złożoności obliczeniowej metody ComplexConditions

Przejdźmy teraz do krótkiej analizy złożoności obliczeniowej metody ComplexConditions dla przypadku pesymistycznego. Złożoność została potraktowana tutaj jako funkcja trzech zmiennych m, n i W, gdzie m i n oznaczają kolejno liczbę kolumn i wierszy w treningowym zbiorze danych, W z kolei określa średnią liczbę unikalnych wartości atrybutów. Wartość n stanowi w oczywisty sposób górne ograniczenie dla W.

Przed przejściem do wyznaczenia złożoności czasowej metody ComplexConditions zostanie przeprowadzona jej analiza dla algorytmu RuleKit. Autorzy metody szacują ją w swej pracy pod tytułem *„GuideR: A guided separate-and-conquer rule learning in classification, regression, and survival settings”* [162]. Przedstawiają ją jako funkcje n i m. Może ona jednak zostać łatwo zmodyfikowana w taki sposób, by zależała ona również od W, pamiętając, że maksymalna wartość W wynosi n. Oprócz podania samej złożoności, w notacji dużego O , zostanie odtworzony także cały proces myślowy przeprowadzony przez autorów w celu jej uzyskania. Posłuży on jako punkt startowy do rozważań na temat złożoności metod proponowanych w niniejszej pracy. Posiadają one bowiem liczne cechy wspólne z algorytmem RuleKit, bazując na nim w wielu aspektach. Jako operację elementarną dla wszystkich analizowanych metod wybrano, podobnie jak autorzy algorytmu RuleKit, operację oceny warunku.

Złożoność obliczeniowa algorytmu RuleKit

Proces indukcji reguł w algorytmie RuleKit składa się z faz wzrostu i przycinania. W trakcie pierwszej z nich warunki są iteracyjnie dodawane do reguły. Skorzystamy tutaj z wcześniej wyznaczonej w rozdziale 6 w podsekcji 1.1 maksymalnej liczby warunków prostych, jakie można utworzyć na zbiorze danych D. W przypadku algorytmu RuleKit tylko takie warunki są brane pod uwagę przy indukcji reguł. Dla każdego z atrybutów liczba ta wynosi maksymalnie $2W - 2$, gdy atrybut jest atrybutem numerycznym oraz W gdy jest on atrybutem nominalnym. Maksymalna liczba warunków w regule wynosi n. Wynika to z faktu, że algorytm pozwala na dodawanie do reguły wielu warunków opartych o ten sam atrybut. Każdy z nich z kolei redukuje pokrycie reguły o minimum jeden przykład. Zbiór reguł może z kolei zawierać maksymalnie n reguł.

Ocena warunku wymaga w praktyce oceny jakości całej reguły w dodanym warunku. W przypadku problemu klasyfikacji można tego dokonać w stałym czasie [162]. Dla regresji i przeżycia nie jest to jednak takie proste, ponieważ każda zmiana pokrycia reguły wymusza pewne dodatkowe obliczenia. W przypadku regresji jest to wyznaczenie mediany lub średniej wartości atrybutu decyzyjnego dla przykładów pokrytych. Dla analizy przeżycia konieczne jest dopasowanie estymatora Kaplana-Meiera dla przykładów pokrytych [162]. Z wyżej opisanych powodów, dla tych dwóch typów problemów, ocena warunku jest dokonywana nie w czasie stałym, lecz liniowym względem n [162]. Dalsze rozważania na temat złożoności algorytmu RuleKit jak i proponowanych przez autora metod będą prowadzone dla bardziej wymagających obliczeniowo problemów regresji i przeżycia.

W trakcie fazy przycinania warunki reguły są iteracyjnie usuwane, dążąc do zwiększenia jej jakości. Jako że maksymalna liczba warunków w regule wynosi n, a w najgorszym przycięte zostaną wszystkie warunki z wyjątkiem jednego, złożoność procesu dla pojedynczej reguły wyniesie $O(n^2)$ [162].

W trakcie wzrostu dla każdej z maksymalnie n reguł, dodane zostaną maksymalnie n warunków. Znalezienie każdego z nich wymaga przetestowania $2W-2$ warunków dla każdego spośród m atrybutów. Ocena pojedynczego warunku dla problemów regresji i przeżycia wymaga liczby operacji liniowej względem n. Podsumowując, maksymalna złożoność czasowa algorytmu RuleKit w notacji dużego O została przedstawiona wzorem 6.4 [162].

$$O(Wn^3m) \tag{6.4}$$

W przypadku pesymistycznym dla $W = n$ jest ona zadana wzorem 6.5.

$$O(n^4m) \tag{6.5}$$

Znając proces wyznaczania złożoności algorytmu RuleKit, można ją również wyrazić w alternatywny sposób za pomocą wzoru 6.6. Występujący w nim składnik $k = Wm$ reprezentuje maksymalną liczbę warunków rozważanych przez algorytm dla zbioru D. Pozostała część n^3 związana jest z maksymalną liczbą warunków w regule (n), maksymalną liczbą reguł w zbiorze (n) oraz złożonością oceny warunku dla problemów regresji i przeżycia (n). To spostrzeżenie pomoże nam później w szacowaniu złożoności proponowanych w niniejszej pracy metod.

$$O(kn^3) \tag{6.6}$$

Złożoność obliczeniowa algorytmu ComplexConditions

Główna różnica pomiędzy algorytmem ComplexConditions a RuleKit tkwi w zestawie warunków rozpatrywanych w trakcie wzrostu reguły. Faza przycinania reguły przebiega analogicznie jak w przypadku metody RuleKit. Co się tyczy przebiegu wzrostu reguły, jedyna zasadnicza różnica pojawia się w procesie budowania warunków wewnętrznych alternatyw. Wymaga on przeprowadzenia dodatkowego przeszukiwania wiązką. Maksymalna długość takiej alternatywy l oraz szerokość wiązki w są określone parametrami algorytmu i są stałe. Poczyniono w tym miejscu założenie, że ich wartości są dużo mniejsze od k', nie wpływając w istotny sposób na złożoność procesu wyszukiwania wewnętrznych alternatyw w algorytmie ComplexConditions. Wynika z tego, że jest ona liniowa względem liczby rozpatrywanych warunków k'.

Maksymalna liczba warunków ocenianych w ramach pojedynczej iteracji poszukiwania wewnętrznych alternatyw zadana jest wzorem 6.7 (patrz rozdział 6 podsekcja 1.2).

$$k' = \frac{n^2 + 3n - 4}{2} \tag{6.7}$$

Wprowadzenie warunków zanegowanych sprawia, że dla każdego warunku algorytm poddaje ocenie również jego zanegowaną wersję. Sprawia to, że ostateczną liczbę warunków należy jeszcze pomnożyć przez dwa. Nie ma to jednak wpływu na rząd

Warunki proste	$2W - 2$
Warunki przedziałów	$\frac{1}{2}(W^2 - W)$
Warunki relacji atrybutów	$3m^2 - 3m$ (atrybuty numeryczne) $\frac{1}{3}(m^3 - m)$ (atrybuty nominalne)
Warunki zbioru dyskretnego	$\frac{1}{6}(W^3 - W)$

Tabela 6.2: Maksymalna liczba warunków, jakie można utworzyć dla pojedynczego atrybutu

złożoności algorytmu. Na podstawie oszacowanej wartości k' można wywnioskować, że proces generowania wewnętrznych alternatyw nie wpływa na rząd wartości złożoności algorytmu ComplexConditions. Dzieje się tak, ponieważ jego złożoność jest mniejsza od złożoności samego algorytmu RuleKit. Z tego powodu proces wyszukiwania wewnętrznych alternatyw dla atrybutów numerycznych zostanie pominięty w dalszych rozważaniach. Należy jednak podkreślić, że ów dodatkowy krok w oczywisty sposób prowadzi do wydłużenia czasów obliczeń. W dalszej części skupiono uwagę na tym, jaki wpływ na złożoność algorytmu miało wprowadzenie pozostałych form warunków złożonych.

Maksymalna liczba warunków złożonych, jakie można wyznaczyć dla zbioru D została już wyznaczona dla każdego z ich rodzajów z osobna w rozdziale 6 w podsekcji 1.2. Zostały one dodatkowo zebrane i przedstawione w tabeli 6.2.

Biorąc pod uwagę wszystko powyższe, wyznaczenie rzędu złożoności czasowej algorytmu ComplexConditions sprowadza się do znalezienia maksymalnych rzędów wartości dla W i m dla wszystkich ocenianych rodzajów warunków. W przypadku W jest to potęga trzecia, występująca dla warunków zbioru dyskretnego. Dla m najwyższa potęga to trójka, występująca dla warunków relacji atrybutów.

Posiłkując się wzorem 6.4, możemy zapisać złożoność algorytmu ComplexConditions jako $O(kn^3)$. Znając maksymalne rzędy wielkości W i m dla składnika k, uzyskujemy ostateczny wynik zadany wzorem 6.8.

$$O(W^3 n^3 m^3) \tag{6.8}$$

Złożoność dla przypadku pesymistycznego, gdzie $W = n$ została przedstawiona równaniem 6.9.

$$O(n^6 m^3) \tag{6.9}$$

Jak można zaobserwować, wprowadzenie złożonych warunków do algorytmu nie obyło się bez znaczącego zwiększenia jego złożoności obliczeniowej. W przypadku algorytmu ComplexConditions rośnie ona również znacząco szybciej wraz ze wzrostem liczby kolumn oraz unikalnych wartości atrybutów, niż ma to miejsce dla metody RuleKit.

2 Metoda indukcji reguł z warunkami M-z-N – MofNRules

2.1 Wstęp

W przypadku klasycznych, ostrych reguł decyzyjnych, przesłanka reguły zawiera zwykle koniunkcje lub alternatywę warunków. W pierwszym przypadku mówimy o regule dyskryminacyjnej, w drugim z kolei o regule charakterystycznej [5]. Reguła dyskryminacyjna pokrywa dany przykład, gdy wszystkie z warunków przesłanki zostaną dla niego spełnione. W przypadku reguł charakterystycznych wystarczy, by którykolwiek z nich został spełniony. Tego typu działanie „wszystko albo nic" może jednak stanowić w niektórych przypadkach spore ograniczenie.

Łatwo wyobrazić sobie przypadek, gdzie do opisania pewnej grupy przykładów wystarcza spełnienie jedynie kilku spośród wszystkich warunków w regule. Rozpatrzmy przykładowy dwuklasowy zbiór danych, opisujący problem głosowania większościowego. Zbiór zawiera 6 atrybutów binarnych ($a_1 \dots a_6$) określających głosy poszczególnych uczestników głosowania. Klasa 1 przypisywana jest przykładom, które otrzymały większość głosów za.

Opis klasy pozytywnej w postaci zbioru reguł DNF wymaga użycia liczby reguł równej liczbie wszystkich 4-elementowych kombinacji atrybutów. Każda reguła zawiera koniunkcje czterech warunków. W przypadku zbioru reguł w postaci CNF będzie on zawierał tyle samo reguł, a przesłanka każdej z nich będzie zawierać cztery warunki, odpowiadające wszystkim kombinacjom. W tym wypadku jest możliwe wyznaczenie aż 15 takich kombinacji. Mimo stosunkowo prostego przykładu uzyskane w ten sposób zbiory reguł są tutaj dość złożone i niezbyt czytelne (patrz tabela 6.3).

#	DNF	CNF
r1:	$a_1 \wedge a_2 \wedge a_3 \wedge a_4 \rightarrow 1$	$a_1 \vee a_2 \vee a_3 \vee a_4 \rightarrow 1$
r2:	$a_1 \wedge a_2 \wedge a_3 \wedge a_5 \rightarrow 1$	$a_1 \vee a_2 \vee a_3 \vee a_5 \rightarrow 1$
r3:	$a_1 \wedge a_2 \wedge a_3 \wedge a_6 \rightarrow 1$	$a_1 \vee a_2 \vee a_3 \vee a_6 \rightarrow 1$
r4:	$a_1 \wedge a_2 \wedge a_4 \wedge a_5 \rightarrow 1$	$a_1 \vee a_2 \vee a_4 \vee a_5 \rightarrow 1$
r5:	$a_1 \wedge a_2 \wedge a_4 \wedge a_6 \rightarrow 1$	$a_1 \vee a_2 \vee a_4 \vee a_6 \rightarrow 1$
r6:	$a_1 \wedge a_2 \wedge a_5 \wedge a_6 \rightarrow 1$	$a_1 \vee a_2 \vee a_5 \vee a_6 \rightarrow 1$
r7:	$a_1 \wedge a_3 \wedge a_4 \wedge a_5 \rightarrow 1$	$a_1 \vee a_3 \vee a_4 \vee a_5 \rightarrow 1$
r8:	$a_1 \wedge a_3 \wedge a_4 \wedge a_6 \rightarrow 1$	$a_1 \vee a_3 \vee a_4 \vee a_6 \rightarrow 1$
r9:	$a_1 \wedge a_3 \wedge a_5 \wedge a_6 \rightarrow 1$	$a_1 \vee a_3 \vee a_5 \vee a_6 \rightarrow 1$
r10:	$a_1 \wedge a_4 \wedge a_5 \wedge a_6 \rightarrow 1$	$a_1 \vee a_4 \vee a_5 \vee a_6 \rightarrow 1$
r11:	$a_2 \wedge a_3 \wedge a_4 \wedge a_5 \rightarrow 1$	$a_2 \vee a_3 \vee a_4 \vee a_5 \rightarrow 1$
r12:	$a_2 \wedge a_3 \wedge a_4 \wedge a_6 \rightarrow 1$	$a_2 \vee a_3 \vee a_4 \vee a_6 \rightarrow 1$
r13:	$a_2 \wedge a_3 \wedge a_5 \wedge a_6 \rightarrow 1$	$a_2 \vee a_3 \vee a_5 \vee a_6 \rightarrow 1$
r14:	$a_2 \wedge a_4 \wedge a_5 \wedge a_6 \rightarrow 1$	$a_2 \vee a_4 \vee a_5 \vee a_6 \rightarrow 1$
r15:	$a_3 \wedge a_4 \wedge a_5 \wedge a_6 \rightarrow 1$	$a_3 \vee a_4 \vee a_5 \vee a_6 \rightarrow 1$

Tabela 6.3: Zbiór reguł DNF i CNF opisujących klasę $\delta = 1$

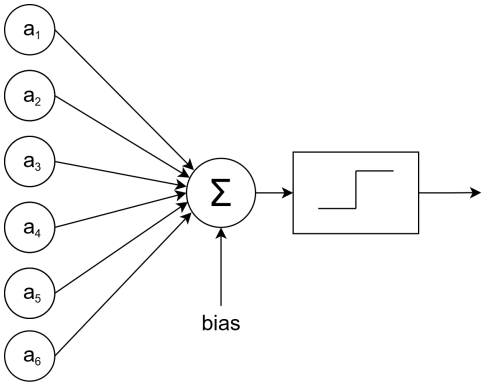
Możliwość generowania reguł zawierających warunki M-z-N, w tym wypadku „przynajmniej M-z-N", pozwala uprościć obydwa zbiory, redukując je do pojedynczej reguły w postaci:

$$\text{JEŻELI } 4\text{-z-}6(a_1, a_2, a_3, a_4, a_5, a_6) \; TO \; \delta = 1$$

2.2 Analogia do perceptronu

Powyższy przykład ukazuje potencjał warunków M-z-N do redukowania złożonych zbiorów reguł w niektórych przypadkach, zachowując przy tym łatwość w interpretacji. Nie jest to jednak ich jedyna zaleta. Warunki tego typu wprowadzają bowiem do systemów regułowych, znany z sieci neuronowych, mechanizm progu odcięcia. Z tego powodu warunki te były popularnie wykorzystywane w ekstrakcji reguł z wytrenowanych sieci neuronowych [133, 157, 171]. Pojedynczy warunek „przynajmniej M-z-N" można porównać do perceptronu ze skokową funkcją aktywacji, przy założeniu, że wszystkie jego wagi są równe 1. Przykładowy perceptron został przedstawiony na rysunku 6.2. Działanie perceptronu można wyrazić równaniem 6.10, gdzie y oznacza wartość na jego wyjściu [148]. Literą b został oznaczony tzw. bias. Jest on dodawany do zsumowanych wartości wejść przemnożonych przez ich wagi. Wynik tego działania przekazywany jest jako argument funkcji aktywacji, której wartość stanowi wartość wyjściową sztucznego neuronu.

$$y \ = f(\sum_{i=1}^{N} w_i a_i + b) \tag{6.10}$$



Rysunek 6.2: Perceptron

Biorąc pod uwagę powyższy opis oraz przyjmując pewne założenia, można przyrównać pojedynczy warunek „przynajmniej M-z-N" do perceptronu zgodnie z poniższym twierdzeniem:

Twierdzenie 1. *Funkcjonowanie warunku „przynajmniej M-z-N" jest identyczne do funkcjonowania pojedynczego perceptronu, który:*

- *posiada N binarnych wejść,*
- *posiada wszystkie wagi $w_1 \ldots w_N$ równe 1,*
- *posiada funkcję aktywacji, będącą funkcją skoku jednostkowego,*
- *posiada bias równy $1 - M$.*

Dowód takiego twierdzenia znajduje się poniżej.

Dowód. Stosując powyższe założenia, równanie 6.10 można uprościć do postaci:

$$y \ = f(\sum_{i=1}^{N} a_i - M) \tag{6.11}$$

Funkcja skoku jednostkowego określona jest równaniem 6.12.

$$f(x) = \begin{cases} 1, & \text{jeśli } x \ \geqslant 0 \\ 0, & \text{jeśli } x \ < 0 \end{cases} \tag{6.12}$$

Rozważmy teraz, w jakim przypadku na wyjściu perceptronu wystąpi wartość 1. Zgodnie z funkcją aktywacji stanie się tak, gdy jej argument x będzie większy lub równy 0. Podstawiając za x wyrażenie $\sum_{i=1}^{N} a_i - M$, na podstawie równania 6.11, otrzymujemy:

$$\sum_{i=1}^{N} a_i - M \geqslant 0 \tag{6.13}$$

Po przeniesieniu M na drugą stronę nierówności otrzymujemy:

$$\sum_{i=1}^{N} a_i \geqslant M \tag{6.14}$$

Jako że w założeniach wartości $a_1 \ldots a_N$ są wartościami binarnymi, nierówność tę można zinterpretować następująco. Perceptron osiągnie wartość 1 na wyjściu wtedy, gdy co najmniej M spośród N jego wejść będzie prawdziwych. Odpowiada to zatem definicji warunku „przynajmniej M-z-N". □

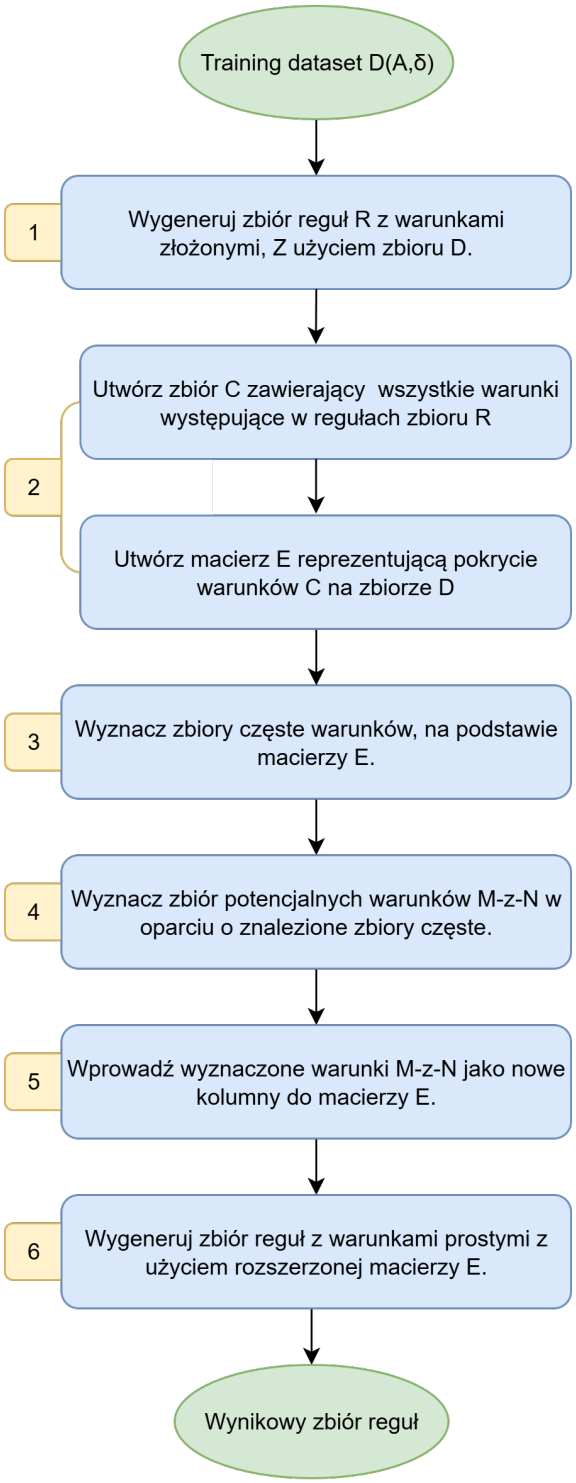
2.3 Generowanie reguł z warunkami M-z-N

Wcześniejsza analogia do perceptronu odnosiła się do warunków „przynajmniej M-z-N". Proponowana metoda umożliwia jednak również generowanie reguł zawierających warunki „dokładnie M-z-N". To, jaki typ warunków M-z-N ma być użyty, jest kontrolowane parametrem algorytmu.

Metoda MofNRules stosuje opisywaną w poprzedniej sekcji metodę ComplexConditions. Reguły z warunkami M-z-N są tutaj generowane w oparciu o idee konstruktywnej indukcji, sterowanej hipotezami (patrz rozdział 4 sekcja 2 podsekcja 2.2). Proces ten odbywa się w sześciu następujących po sobie krokach. Jego przebieg został przedstawiony w postaci schematu blokowego na rysunku 6.3. Poniżej znajduje się szczegółowy opis każdego z kroków.

1. Generowanie reguł z warunkami złożonymi metodą ComplexConditions

W pierwszym kroku generowany jest zbiór reguł, zawierający warunki złożone z zastosowaniem metody ComplexConditions. Proces ten jest dokładnie opisany w rozdziale 6 sekcja 1.



Rysunek 6.3: Schemat blokowy algorytmu MofNRules

2. Przygotowanie macierzy pokrycia warunków E

W drugim kroku algorytmu jest tworzona macierz pokrycia dla warunków zawartych w przesłankach reguł wygenerowanych w kroku pierwszym. Metoda ComplexConditions tworzy reguły, których przesłanki są zawsze koniunkcją warunków. Jako zbiór C, określający kolumny macierzy, używany jest zbiór wszystkich unikalnych warunków wchodzących w skład tych koniunkcji.

Wprowadzone zostało tutaj pojęcie macierzy pokrycia. Jest to macierz binarna, opisująca pokrycie pewnego zbioru warunków C na zbiorze D. Posiada ona $|D|$ wierszy oraz $|C|$ kolumn. Element macierzy $E_{i,j}$ określa, czy warunek c_j pokrywa przykład x_i.

3. Wyznaczenie zbiorów częstych

Macierz przygotowana w poprzednim kroku jest następnie używana do poszukiwania zbiorów częstych. Można je interpretować jako zbiory warunków, które często bywają prawdziwe dla tych samych przykładów uczących. Do poszukiwania zbiorów częstych wykorzystany został algorytm FP-Growth [160]. Poszukiwane są wszystkie zbiory częste o minimalnym wsparciu, zadanym parametrem algorytmu. Spośród wszystkich znalezionych zbiorów są wybierane następnie tylko te, które zawierają M elementów. Przykładowo, generując reguły z warunkami 2-z-3 dla minimalnego wsparcia 20%, wyszukiwane są wszystkie dwuelementowe zbiory częste, pojawiające się w minimum 20% wierszy macierzy E.

4. Wyznaczenie zbioru potencjalnych warunków M-z-N w oparciu o zbiory częste

Utworzone w poprzednim kroku M-elementowe zbiory częste, są używane jako swoista heurystyka do wyznaczenia zbiorów N-elementowych pełniących rolę potencjalnych warunków M-z-N.

Rozważmy przypadek warunku 2-z-4 (M=2, N=4). To, czy jest to warunek „przynajmniej 2-z-4” czy „dokładnie 2-z-4”, nie jest na tę chwilę istotne. Dla takiego przypadku, w kroku czwartym wyznaczone byłyby dwuelementowe zbiory częste. Innymi słowy, dostępna byłaby informacja o dwójkach warunków, które są często spełniane jednocześnie dla tych samych przykładów. Aby utworzyć warunek 2-z-4, potrzebne są jednak aż cztery warunki. Na potrzeby tego przykładu oznaczmy je jako c_1, c_2, c_3, c_4. Do ich poszukiwania metoda MofNRules stosuje poniższą heurystyczną tezę.

Twierdzenie 2. *N-elementowy zbiór wyznaczony poprzez zsumowanie M-elementowych zbiorów częstych, jest często spełnionym warunkiem M-z-N, wtedy gdy każdy M-elementowy zbiór jest zbiorem częstym, oraz zbiór zbiorów M-elementowych wyczerpuje wszystkie możliwe podzbiory M-elementowe zbioru N-elementowego.*

Poniżej znajduje się dowód twierdzenia dla warunków „przynajmniej M-z-N" oraz „dokładnie M-z-N". Należy podkreślić, że twierdzenie nie jest prawdziwe dla warunków „co najwyżej M-z-N".

Dowód. Niech Z będzie zbiorem M-elementowych zbiorów częstych. Jako S oznaczmy N-elementowy zbiór zdefiniowany jako suma wszystkich zbiorów Z. Zgodnie z założeniami tezy 2, każdy ze zbiorów zawartych w Z jest częsty, a Z wyczerpuje wszystkie możliwe M-elementowe podzbiory zbioru S.

Elementami zbiorów są w tym przypadku warunki logiczne. Zbiór jest uznawany za częsty, jeżeli warunki, które zawiera, były spełnione jednocześnie dla pewnej zadanej minimalnej liczby przykładów w zbiorze danych oznaczonej jako s.

Skoro każdy M-elementowy podzbiór S jest częsty, a Z wyczerpuje wszystkie M-elementowe podzbiory S, oznacza to, że każda kombinacja M elementów zbioru S również jest częsta. Wynika z tego, że dowolne M warunków, wybrane spośród N warunków tworzących zbiór S, są jednocześnie spełnione dla minimum s przykładów w zbiorze. Oznacza to, że warunek „dokładnie M-z-N", utworzony na podstawie zbioru, S jest często spełniony.

Rozpatrzmy teraz przypadek warunku „przynajmniej M-z-N". Niech Y będzie pewnym podzbiorem zbioru S, gdzie $|Y| \geqslant M$.

Ponieważ wszystkie M-elementowe podzbiory S są częste, oznacza to, że dowolnie wybrany Y zawiera w sobie jeden lub więcej zbiorów częstych. Będzie to dokładnie $\binom{N}{|Y|}$ zbiorów częstych. Im więcej elementów będzie zawierał zbiór Y, tym więcej podzbiorów częstych o wielkości M będzie zawierał. W skrajnym przypadku wszystkie podzbiory pokryją dokładnie te same przykłady w danych. W takim wypadku dowolnie wybrany Y wystąpi w danych s razy i będzie częsty. Im większe $|Y|$, tym większa jednak będzie szansa, że Y występuje w danych częściej niż s razy.

Wynika z tego, że utworzony na podstawie zbioru S warunek „przynajmniej M-z-N" jest spełniony dla wystarczającej liczby przykładów, by uznać go za często występujący. Dodatkowo im więcej spośród jego N warunków składowych zostanie spełnionych, tym większe prawdopodobieństwo, że jest on spełniony dla większej liczby przykładów w danych.

□

Oczywiście fakt, że warunek został spełniony dla dużej liczby przykładów w danych, nie musi oznaczać, że dany warunek dodany do reguły poprawi jej jakość. Powyższa teza służy jedynie do wyszukiwania i zawężania puli kandydatów na warunki M-z-N. One same są jednak następnie oceniane przez algorytm indukcji w sposób analogiczny jak w przypadku algorytmu ComplexConditions (patrz krok 6). Z tego też powodu mogą one, lecz nie muszą, pojawić się w wynikowym zbiorze reguł.

Wracając do naszego przykładu z warunkiem 2-z-4. W tym wypadku teza 2 będzie prawdziwa, jeżeli istnieje czwórka warunków, których wszystkie możliwe dwuelementowe kombinacje są obecne wśród znalezionych zbiorów częstych. W przypadku czterech elementów, istnieje 6 takich podzbiorów: $\{c_1, c_2\}$,$\{c_2, c_3\}$,$\{c_3, c_4\}$,$\{c_3, c_1\}$,$\{c_4, c_1\}$, $\{c_4, c_2\}$. Jeżeli wszystkie one są obecne wśród znalezionych w kroku 3 zbiorów częstych, to czwórka c_1, c_2, c_3, c_4 traktowana jest jako kandydat na warunek 2-z-4.

5. Rozszerzenie macierzy binarnej o dodatkowe kolumny reprezentujące warunki M-z-N

Aby zapobiegać problemowi eksplozji cech (z ang. feature explosion), pula wyznaczonych warunków M-z-N przycinana jest do pewnej maksymalnej wielkości zadanej parametrem algorytmu. Do wyboru tego, które spośród warunków zostaną wybrane, stosowana jest średnie wsparcie zbiorów częstych wykorzystywanych do zbudowania warunku. Wybierana jest ustalona liczba warunków, dla których było ono najwyższe.

Następnie dla każdego z potencjalnie jakościowych warunków M-z-N, obliczany jest wektor binarny reprezentujący ich pokrycie na zbiorze treningowym. W ten sposób warunki te zamieniane są niejako w nowe atrybuty binarne, które dodawane są następnie jako kolejne kolumny w macierzy E, zgodnie z wcześniej opisywaną koncepcją konstruktywnej indukcji sterowanej hipotezami. Rozszerzona o dodatkowe kolumny macierz E zostanie oznaczona jako E'.

6. Ponowna indukcja reguł z użyciem rozszerzonej macierzy binarnej

W ostatnim kroku następuje ponowne generowanie reguł z użyciem algorytmu ComplexConditions. Tym razem jednak zbiór treningowy zastępowany jest macierzą E'. Algorytm generuje tutaj reguły tylko i wyłącznie z warunkami prostymi bez negacji.

Aby lepiej uzasadnić to ograniczenie, rozważmy przykład z macierzą E wygenerowaną dla zbioru D i trzech warunków: $c_1 = a_1 > 5$, $c_2 = a_2 \neq \{1\}$ i $c_3 = a_2 < 3$. Posiada ona trzy kolumny i $|D|$ wierszy. W kroku piątym została do niej dodana kolumna $c_4 = 2\text{-z-}3(a_1 > 5,\ a_2 \neq \{1\},\ a_2 < 3)$, reprezentująca warunek 2-z-3. Macierz E' została zaprezentowana na równaniu 2.3.

$$E' = \begin{bmatrix} c_1 & c_2 & c_3 & c_4 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

W kroku 6 dla macierzy E' generowane są reguły zawierające jedynie niezanegowane warunki proste. Jako że wszystkie atrybuty (kolumny macierzy) są tutaj binarne, warunki te mogą przyjmować jedną z dwóch postaci: $c_i = 1$ lub $c_i = 0$. Pierwszy warunek, w oparciu o to, jak wyznaczana jest macierz E', spełniony jest wtedy, gdy spełniony jest warunek c_i. Warunek $c_i = 1$ jest więc równoważny warunkowi c_i. Warunek $c_i = 0$ odpowiada z kolei zanegowanemu warunkowi c_i i można go zapisać jako $\neg c_i$. W ten sposób algorytm umożliwia w pośredni sposób generowanie reguł z zanegowanymi warunkami, mimo iż nie są one tworzone wprost, lecz w oparciu o macierz E'.

Jak zostało zaprezentowane na przykładzie, reguły wyznaczane w kroku 6. mogą zawierać wszystkie warunki odpowiadające kolumnom macierzy E' oraz ich negacje. Macierz E' może z kolei potencjalnie zawierać wszystkie warunki złożone wygenerowane przez algorytm ComplexConditions (patrz rozdział 6 sekcja 1 podsekcja 1.2). W związku z powyższym oraz dla ograniczenia czasu obliczeń, reguły generowane w kroku 6 nie uwzględniają warunków złożonych.

2.4 Przykład

Niech zbiór danych D składa się z trzech atrybutów a_1, a_2 i a_3 oraz atrybutu decyzyjnego δ, określającego przynależność przykładu do jednej z dwóch klas: 0 i 1. Atrybut a_1 jest atrybutem nominalnym, atrybuty a_2 i a_3 są z kolei atrybutami numerycznymi. Algorytm uruchomiony zostaje dla $M = 2$, $N = 3$ oraz dla minimalnego wsparcia zbiorów częstych równemu 20%.

Poniżej znajduje się opis działania kolejnych kroków algorytmu dla zadanego przykładu.

1. Generowanie reguł z warunkami złożonymi metodą ComplexConditions

W pierwszym kroku zostały wygenerowane następujące reguły:

JEŻELI $a_1 = \{1\} \wedge a_2 > 5$ **TO** $\delta = \{\mathbf{1}\}$
JEŻELI $a_2 > 2 \wedge a_2 > a_3$ **TO** $\delta = \{\mathbf{1}\}$
JEŻELI $a_2 < 5 \wedge a_1 = \{0\}$ **TO** $\delta = \{\mathbf{0}\}$
JEŻELI $a_1 = \{2\} \wedge a_2 > 2$ **TO** $\delta = \{\mathbf{0}\}$

2. Przygotowanie macierzy pokrycia warunków E

Dla reguł wygenerowanych w pierwszym kroku została wyznaczona macierz E. Posiada ona 7 kolumn ($c_1, \ldots, c_7$), odpowiadającym wszystkim unikalnym warunkom występującym w regułach: c_1: $a_1 = \{0\}$, c_2: $a_1 = \{1\}$, c_3: $a_1 = \{2\}$, c_4: $a_2 > 5$, c_5: $a_2 < 5$, c_6: $a_2 > 2$, c_7: $a_2 > a_3$.

3. Wyznaczenie zbiorów częstych

Na podstawie macierzy E uruchomiony zostaje algorytm poszukiwania zbiorów częstych FP-Growth. Wyszukuje on następujące zbiory częste:

- $s_1 = \{a_1 = \{0\}\,,\, a_2 < 2\}$;
- $s_2 = \{a_1 = \{0\}\,,\, a_2 = a_3\}$;
- $s_3 = \{a_2 < 2\,,\, a_2 = a_3\}$.

4. Wyznaczenie zbioru potencjalnych warunków M-z-N w oparciu o zbiory częste

Dla znalezionych zbiorów częstych są poszukiwane zbiory trzech warunków, dla których wszystkie ich dwuelementowe podzbiory są zbiorami częstymi.

Dla rozpatrywanego przypadku istnieje jeden taki zbiór: $\{c_1, c_6, c_7\}$, na podstawie którego algorytm utworzył jeden warunek 2-z-3.

5. Rozszerzenie macierzy binarnej o dodatkowe kolumny reprezentujące warunki M-z-N

Macierz E zostaje rozszerzona o kolumnę binarną c_8, reprezentującą warunek 2-z-3, znaleziony w poprzednim kroku. W ten sposób zostaje utworzona nowa macierz E', zawierająca 8 kolumn.

6. Ponowne indukcja reguł z użyciem rozszerzonej macierzy binarnej

W ostatnim kroku, w oparciu o macierz E', zostają wygenerowane następujące reguły z warunkami prostymi:

$$\begin{aligned}
&\textbf{JEŻELI } c_8 = \{0\} \textbf{ TO } \delta = \{\mathbf{1}\} \\
&\textbf{JEŻELI } c_5 = \{1\} \wedge c_1 = \{1\} \textbf{ TO } \delta = \{\mathbf{0}\} \\
&\textbf{JEŻELI } c_3 = \{1\} \wedge c_6 = \{1\} \textbf{ TO } \delta = \{\mathbf{0}\}
\end{aligned}$$

Mapując atrybuty występujące w powyższych regułach na zbiór warunków $C = \{c_1, \ldots, c_8\}$, otrzymujemy wynikowy zbiór reguł w postaci:

$$\begin{aligned}
&\textbf{JEŻELI } \neg\text{2-z-3}(a_1 = \{0\}, a_2 < 2, a_2 = a_3) \textbf{ TO } \delta = \{\mathbf{1}\} \\
&\textbf{JEŻELI } a_2 < 5 \wedge a_1 = \{0\} \textbf{ TO } \delta = \{\mathbf{0}\} \\
&\textbf{JEŻELI } a_1 = \{2\} \wedge a_2 > 2 \textbf{ TO } \delta = \{\mathbf{0}\}
\end{aligned}$$

Warto zwrócić w tym miejscu uwagę na fakt, że algorytm, ze względu na sposób działania, umożliwia generowanie reguł z zanegowanymi warunkami M-z-N.

2.5 Parametry algorytmu

Poniżej znajduje się zestawienie parametrów algorytmu MofNRules wraz z opisem ich działania. Poza tymi wymienionymi, posiada on również wszystkie parametry algorytmu ComplexConditions wymienione w rozdziale 6 podsekcji 1.4. Ich wartości

używane są w pierwszym kroku do wygenerowania reguł zawierających warunki złożone. Krok szósty korzysta z tych samych wartości parametrów, wyłączając jedynie generowanie wszystkich rodzajów warunków złożonych. Oznacza to, że wyłączone zostają następujące flagi: discrete_set_conditions_enabled, negated_conditions-_enabled, intervals_conditions_enabled, numerical_attributes_conditions_enabled, nominal_attributes_conditions_enabled, inner_alternatives_enabled.

Oprócz tego algorytm posiada następujące specyficzne dla niego parametry:

- ***M*** – Parametr określający dla jakiego M poszukiwane powinny być warunki M-z-N; domyślnie 2.
- ***N*** – Parametr określający dla jakiego N poszukiwane powinny być warunki M-z-N; domyślnie 3.
- ***min_supp_fraction*** – Ułamek z przedziału (0,1) określający minimalne wsparcie zbiorów częstych poszukiwanych w kroku trzecim; domyślnie 0.2.
- ***exact_m_of_n*** – Flaga określający czy generowane warunki M-z-N powinny być traktowane jako warunki „dokładnie M-z-N", jeżeli wyłączona oznacza, że warunki traktowane są jako „przynajmniej M-z-N"; domyślnie włączona.
- ***max_candidates*** – Dodatnia liczba całkowita określająca maksymalną liczbę warunków M-z-N wyznaczonych w kroku czwartym, które zostaną dodane do macierzy binarnej w kroku piątym; domyślnie wybierane jest 1000 warunków o najwyższym średnim wsparciu zbiorów częstych użytych do ich zbudowania.

2.6 Analiza złożoności obliczeniowej metody MofNRules

W przypadku metody MofNRules oszacowanie złożoności jest trudniejsze niż miało to miejsce dla metody ComplexConditions. Dzieje się tak ze względu na fakt, że wykorzystuje ona zarówno algorytm FP-Growth do poszukiwania zbiorów częstych, jak i ComplexRules do indukcji reguł. Sama indukcja jest z kolei wykonywana dwukrotnie, co dodatkowo komplikuje obliczenia.

Złożoność algorytmu FP-Growth wynika w głównej mierze z rozmiaru tworzonego w trakcie działania FP-drzewa oraz progu minimalnego wsparcia [70]. Ten pierwszy zależy jednak również od rozkładu samych danych. Z tego powodu w literaturze nie występuje jeden konkretny wzór, wyrażający jego złożoność w domenie n, m oraz

W. Badaniami dotyczącymi wydajności algorytmu FP-Growth względem liczby kolumn w danych zajmowali się w swej pracy m .in. Kanwal i Deepak [70]. Stosowali oni jednak podejście eksperymentalne, porównując czasy obliczeń algorytmu dla różnych konfiguracji. Wskazywali oni na to, że wzrost m ma tutaj znacząco większy wpływ niż n. Uzyskane w ramach wspominanej pracy czasy wskazują jednak, że czas wykonania rośnie wolniej niż sześciennie względem m. Na tej podstawie można ostrożnie założyć, że uznany za wydajny algorytm FP-Growth nie powinien zdominować złożoności całego algorytmu MofNRules.

W ramach drugiego przebiegu indukcji generowane są reguły zawierające tylko i wyłącznie warunki proste. Jako zbiór danych brana jest tutaj rozszerzona macierz E'. Z racji, że każda z jej kolumn zawiera tylko i wyłącznie wartości binarne, oznacza to, że $W = 2$. Algorytm rozpatrywać będzie zatem tylko dwa warunki dla każdego z atrybutów. Dodatkowo nie są tutaj poszukiwane żadne z warunków złożonych ani ich negacje. Stosując równanie 6.4, gdzie $k = 2m'$, złożoność drugiego przebiegu indukcji reguł określona jest wzorem 6.15. Warto w tym miejscu zwrócić uwagę, że zamiast m w równaniu pojawia się nowe oznaczenie m'. To dlatego, że liczba kolumn w macierzy E' nie jest równa liczbie kolumn w oryginalnym zbiorze D.

$$O(m'n^3) \tag{6.15}$$

Liczba m' jest sumą dwóch składników: liczby wszystkich unikalnych warunków w zbiorze reguł wygenerowanym w pierwszym kroku oraz liczby warunków M-z-N wygenerowanych w kroku czwartym. Pierwszy składnik wynosi n^2, ponieważ każda z maksymalnie n reguł może zawierać maksymalnie n warunków w przesłance. Drugi składnik jest jednak trudniejszy do oszacowania. Zależy on bowiem zarówno od liczby znalezionych zbiorów częstych, ich rozmiaru, jak i tego, ile potencjalnych warunków M-z-N zostanie utworzonych na ich podstawie. W najgorszym możliwym scenariuszu wszystkie M-elementowe podzbiory Wn warunków okażą się częste. Oznacza to, że w kroku czwartym algorytmu MofNRules wyznaczonych zostanie $\binom{n^2}{N}$ warunków M-z-N. Rozpisując symbol Newtona, otrzymamy wielomian zmiennej n, którego stopień równy jest $N + 2$ (dwa wynika z początkowej drugiej potęgi). Stosując takie oszacowanie m', złożoność drugiej iteracji indukcji wyrazić można wzorem 6.16. Jak można zauważyć, nie jest ona zależna od liczby kolumn w oryginalnym zbiorze treningowym.

$$O(n^N) \tag{6.16}$$

Podsumowując, złożoność obliczeniowa algorytmu MofNRules jest skomplikowaną funkcją wielu zmiennych, nie tylko n, m oraz W. Zależy ona również, między innymi, od minimalnego wsparcia zastosowanego przez FP-Growth oraz tego, dla jakich M i N poszukiwane są warunki M-z-N. Wszystko to sprawia, że porównanie jego złożoności teoretycznej z pozostałymi metodami jest zadaniem trudnym. Warto również wspomnieć, że w rzeczywistych zbiorach danych sytuacja, gdzie wszystkie M-elementowe zbiory warunków okażą się częste, jest praktycznie niemożliwa do wystąpienia. Przykładowo, żaden zbiór warunków prostych opartych na jednym atrybucie nominalnym nie może być częsty z definicji. Przeprowadzone oszacowania pokazują jednak, że złożoność metody MofNRules rośnie gwałtownie wraz ze wzrostem N (nie mylić z n). W przypadku stosowanego przez nią algorytmu FP-Growth, kluczowy wpływ na jego złożoność ma wybór progu wsparcia dla zbiorów częstych [70]. Zbyt niskie wartości mogą tutaj znacząco wydłużyć czas obliczeń.

3 Metoda indukcji reguł z zagnieżdżonymi wyrażeniami logicznymi – DeepRules

3.1 Motywacja

Przed przedstawieniem ostatniej z metod proponowanych przez autora niniejszej pracy, aby lepiej wyjaśnić jej sens, wróćmy na moment do rodzajów warunków złożonych stosowanych przez dwie wcześniejsze. Dla algorytmu ComplexConditions były to:

- **Warunki proste** – np. wiek > 20 lub $kolor = \{czerwony\}$,
- **Warunki przedziałów** – np. wiek $\in [20, 35)$,
- **Warunki zbioru dyskretnego** – np. kolor $\in$ {czerwony, zielony, niebieski},
- **Warunki wewnętrznej alternatywy** (dla atrybutów numerycznych) – np. wiek $> 20 \vee wiek < 35$,
- **Warunki relacji atrybutów** – np. temperatura_1 $>$ temperatura_2,
- **Negacje wyżej wymienionych rodzajów warunków**

Dla metody MofNRules były to dodatkowo warunki „przynajmniej M-z-N" i „dokładnie M-z-N".

W metodzie ComplexConditions każdy rodzaj warunku był generowany i testowany osobno. Tego typu podejście w znaczący sposób zwiększało pulę warunków ocenianych w trakcie fazy wzrostu reguły (patrz rozdział 1.6). Dla warunków wewnętrznych alternatyw (dla atrybutów numerycznych) była zastosowana dodatkowo osobna procedura przeszukiwania wiązką. Algorytm wprowadzał jednak szereg ograniczeń, by utrzymać rozsądne czasy obliczeń. Z kolei w metodzie MofoNRules uzyskanie reguł z warunkami M-z-N wymagało aż dwukrotnej indukcji oraz dodatkowego kroku poszukiwania zbiorów częstych. Była ona także podatna na problem eksplozji cech, któremu zapobiegano poprzez przycinanie puli warunków M-z-N branych pod uwagę przez algorytm.

Patrząc na wyżej wymienione rodzaje warunków, zauważyć można, że większość z nich da się zapisać z użyciem samych tylko warunków prostych, stosując operatory koniunkcji i alternatywy. Wyjątkiem są tutaj negacje oraz warunki relacji atrybutów, gdzie nie sposób tego dokonać.

Warunek przedziału $a_i \in [l, r)$ potraktować można jako koniunkcję warunków prostych $a_i \geqslant l \ \wedge \ a_i < r$. Warunek zbioru dyskretnego $a_i \in \{v_1, v_2, \ldots, v_N\}$ można zapisać z kolei jako alternatywę warunków prostych: $a = \{v_1\} \vee a_i = \{v_2\} \vee \ldots \vee a_i = \{v_N\}$. Warunki wewnętrznej alternatywy dla atrybutów numerycznych można przedstawić w postaci: $c_1 \vee c_2 \vee \ldots \vee c_N$, gdzie $c_1, c_2, \ldots, c_N$ są warunkami prostymi lub warunkami przedziału dla tego samego atrybutu.

Podsumowując wszystkie rodzaje warunków stosowanych w metodzie ComplexConditions, z wyłączeniem warunków zanegowanych oraz warunków relacji atrybutów, można zapisać w alternatywnej postaci z użyciem koniunkcji lub alternatywy warunków prostych.

Sprawdźmy, czy podobna sytuacja występuje w przypadku warunków M-z-N. Jak zostało to wcześniej wykazane (patrz rozdział 4, sekcja 4 równania 3.2 i 3.3), zarówno „przynajmniej M-z-N", jak i „dokładnie M-z-N" można zapisać w postaci alternatywy koniunkcji. Warunki M-z-N w metodzie MofoNRules mogą jednak zawierać warunki proste oraz wszystkie rodzaje warunków złożonych występujące w metodzie ComplexConditions. W przypadku, gdy jeden z N warunków posiada w swoim zapisie operator alternatywy (np. warunek zbioru dyskretnego), do zapisu takiego M-z-N jest potrzebne jeszcze jedno zagnieżdżenie operatorów logicznych (alternatywa koniunkcji alternatyw).

Zarówno pierwsza, jak i druga z proponowanych metod skupiały się na generowaniu reguł, gdzie warunki w przesłance były łączone operatorem koniunkcji (mimo że poszczególne warunki mogły zawierać alternatywy logiczne). Ta cecha sprawiała, że nie było możliwe uzyskanie wcześniej wspomnianych alternatywnych zapisów warunków złożonych z użyciem warunków prostych. Rozważmy teraz hipotetyczny algorytm, umożliwiający generowanie reguł, których przesłanki mają postać CNF (koniunkcja alternatyw) lub DNF (alternatywa koniunkcji). Na podstawie wcześniejszych rozważań można stwierdzić, że taka metoda umożliwiałaby budowanie reguł zawierających wszystkie rodzaje warunków złożonych, używanych przez ComplexConditions, z wyjątkiem negacji i warunków relacji atrybutów. Algorytm taki oceniałby jednak dużo mniejszą pulę warunków (składającą się tylko z warunków prostych) w trakcie fazy wzrostu reguły. Ograniczenie związane z warunkami relacji atrybutów oraz negacjami można zlikwidować poprzez rozszerzenie tejże puli o te dodatkowe rodzaje warunków. Wciąż jednak liczba hipotez ocenianych w trakcie wzrostu będzie znacząco mniejsza niż ma to miejsce w przypadku metody ComplexConditions. Co więcej, metoda taka nie wymagałaby osobnej procedury przeszukiwania wiązką do wyszukiwania warunków wewnętrznych alternatyw. Idąc dalej, algorytm tego rodzaju byłby dodatkowo zdolny do budowania w pojedynczym kroku indukcji reguł zawierających warunki M-z-N. W tym wypadku wprowadzałby on jednak pewne ograniczenia względem metody MofNRules. Każda reguła mogłaby zawierać tutaj tylko pojedynczy warunek M-z-N, z racji że przesłanka reguły może posiadać wyłącznie postać CNF lub DNF. Dodatkowo nie mogłyby one zawierać w sobie żadnych warunków wykorzystujących alternatywę logiczną: warunki wewnętrznej alternatywy oraz zbioru dyskretnego.

Podsumowując, możliwość indukcji reguł w postaci CNF lub DNF, zawierających zagnieżdżone formuły koniunkcji i alternatyw logicznych w swych przesłankach, posiada szereg zalet, niewelując pewne spośród problemów obecnych w dwóch wcześniej proponowanych metodach. Te właśnie przemyślenia i wnioski stanowiły inspirację dla trzeciej z proponowanych metod o nazwie DeepRules.

3.2 Generowanie zbiorów reguł z zagnieżdżonymi wyrażeniami logicznymi algorytmem DeepRules

Generowanie reguł z użyciem algorytmu DeepRules odbywa się z użyciem strategii sekwencyjnego pokrywania. Indukcja pojedynczej reguły, podobnie jak ma to miejsce w RuleKit, składa się z dwóch faz: wzrostu oraz przycinania. Faza wzrostu przebiega

jednak tutaj w odmienny sposób, umożliwiając budowanie reguł z zagnieżdżonymi formułami logicznymi w przesłankach.

Wzrost reguły

Przebieg fazy wzrostu w algorytmie DeepRules został przedstawiony na pseudokodzie 5. Główna różnica względem wcześniejszych metod polega na iteracyjnym dodawaniu do reguły składników zamiast pojedynczych warunków. Taki składnik może mieć postać alternatywy lub koniunkcji warunków: prostych, relacji atrybutów lub ich negacji. Parametrem procedury WZRASTAJ, dokonującej wzrostu reguły, jest jej typ t. Określa on, czy przesłanka generowanej reguły powinna mieć postać CNF, czy też DNF. W zależności od tego procedura WZRASTAJSKLADNIK zwróci koniunkcję dla postaci DNF i alternatywę dla CNF. Procedura DODAJSKLADNIK wybiera na podstawie typu reguły, jakiego operatora logicznego użyć, dodając kolejny składnik reguły. Dla reguł w postaci CNF zostanie on dodany z użyciem operatora koniunkcji, a dla DNF z użyciem alternatywy.

Algorytm 5 Wzrost reguły w algorytmie DeepRules

Wejście:
r — reguła,
D — zbiór treningowy,
D_{uncov} — przykłady niepokryte przez regułę,
t — typ generowanych reguł (CNF lub DNF).
Wyjście: *rule*
function WZRASTAJ(r, D, D_{uncov}, t)
  while true do
    $e \leftarrow$ WZRASTAJSKLADNIK(r, D, D_{uncov}, t)
    if $|r| \geqslant max\_growing \vee e = \varnothing$ **then**
      break
    end if
    $r \leftarrow$ DODAJSKLADNIK(r, e)
  end while
  return $|r| > 0$
end function

Dla lepszego zilustrowania tego procesu znajduje się poniżej przykładowa reguła z przesłanką w postaci DNF, zawierającą dwa składniki: $(a_1 = \{1\} \wedge a_2 > 5)$ oraz $a_2 < 3$.

$$\textbf{JEŻELI } (a_1 = \{1\} \wedge a_2 > 5) \vee a_2 < 3 \textbf{ TO } \delta = \{\mathbf{1}\}$$

Warto zaznaczyć w tym miejscu, że składniki reguł, w skrajnym przypadku, mogą być alternatywą lub koniunkcją tylko jednego warunku. W takim wypadku,

niezależnie od postaci reguły, składnik taki jest równoważny temu warunkowi. To bardzo istotna własność, ponieważ dzięki niej format reguł generowanych przez algorytm jest bardziej elastyczny. W przypadku gdy każdy ze składników reguły typu CNF będzie zawierać tylko jeden element, otrzymamy tutaj klasyczną regułę koniunkcyjną podobną do tej generowanej przez algorytmy CN2 czy RuleKit.

Dodając do przykładowej reguły kolejny składnik w postaci $(a_3 = \{0\} \wedge a_2 < 10)$, otrzymamy następującą regułę (dodany składnik został podkreślony):

$$\textbf{JEŻELI } (a_1 = \{1\} \wedge a_2 > 5) \vee a_2 < 3 \vee \underline{(a_3 = \{0\} \wedge a_2 < 10)} \textbf{ TO } \delta = \{\mathbf{1}\}$$

Proces wzrostu reguły polega na iteracyjnym dodawaniu składników do jej przesłanki. Zakończony jest on osiągnięciem zdefiniowanej maksymalnej długości reguły (liczonej względem liczby jej składników), bądź gdy kolejny wygenerowany składnik jest pusty. Maksymalna liczba składników w regule kontrolowana jest parametrem.

W trakcie wzrostu reguł, po dodaniu każdego z warunków, postać reguły oceniana jest osobną miarą jakości. Jej wartość wraz z pośrednią postacią reguły jest zapisywana. Na koniec fazy wzrostu wybierana i zwracana jest ta reguła, dla której wartość miary jakości była najwyższa. Ze względu na działanie algorytmu nie musi być to wcale postać reguły uzyskana po zakończeniu fazy wzrostu. Możliwe jest zdefiniowanie osobnych miar wykorzystywanych do wyboru najlepszej postaci reguły DNF i CNF (patrz podsekcja 3.4).

Generowanie składników reguły

Proces generowania kolejnych składników reguły przypomina wzrost reguły w algorytmie RuleKit. Jego przebieg został przedstawiony na pseudokodzie 6. Rozpoczynamy z pustym składnikiem (pustą alternatywą lub koniunkcją). Następnie iteracyjnie podejmowana jest próba dodania do niego kolejnych warunków. Za każdym razem wybierany jest najlepszy pod kątem wybranej miary jakości. Podobnie jak w przypadku algorytmu RuleKit możliwe jest użycie dowolnie zdefiniowanej miary, opartej o liczbę pokrytych przykładów pozytywnych i negatywnych. Dla problemów regresji i przeżycia, gdzie pojęcia przykładów pozytywnych i negatywnych nie ma zastosowania, przyjęte zostało takie samo rozwiązanie, jak ma to miejsce w algorytmie RuleKit (patrz rozdział 6 sekcja 1 posekcja 1.1). Algorytm umożliwia także stosowanie różnych miar jakości podczas generowania reguł CNF i DNF (patrz podsekcja 3.4).

Proces wzrostu składnika trwa, aż do momentu osiągnięcia określonego kryterium stopu. Dzieje się tak albo w przypadku, gdy długość składnika osiągnie maksymalną

wartość określoną parametrem algorytmu, albo gdy dodawanie kolejnych warunków nie doprowadza do reguły pokrywającej minimalną liczbę nowych przykładów. Liczba ta również jest parametrem algorytmu.

Algorytm 6 Wzrastanie pojedynczego składnika reguły w algorytmie DeepRules

Wejście:
r — reguła,
D — zbiór treningowy,
D_{uncov} — przykłady niepokryte przez regułę,
t — typ generowanych reguł (CNF lub DNF).
Wyjście: e —pojedynczy składnik reguły (alternatywa bądź koniunkcja).

```
function WzrastajSkladnik(r, D, D_uncov, t)
    e ← ∅
    D_cov ← ObliczPrzykladyPokryte(r, D)
    while true do
        c_best ← ZnajdzNajlepszyWarunek(r, e, D, D_cov, D_uncov)
        if c_best = ∅ then
            break
        end if
        e ← DodajWarunek(e, c_best, t)
        if |e| ⩾ max_component_length then
            break
        end if
    end while
    return e
end function
```

Przycinanie reguły

Procedura przycinania reguły jest analogiczna do tej jaka stosowana jest w RuleKit (patrz rozdział 6 sekcja 1 1.1). Warto zaznaczyć jednak, że przycinane są tutaj dowolne warunki ze składników reguły, a nie same składniki. W skrajnym przypadku jednak dany składnik reguły może być pustą alternatywą lub koniunkcją po przycięciu. W takim wypadku jest on usuwany z reguły.

Indukcja mieszanych zbiorów reguł zawierających reguły z przesłankami w postaci CNF i DNF

Wyżej opisane procesy wzrostu i przycinania umożliwiają generowanie reguł z przesłankami w postaci CNF lub DNF. Dla uproszczenia będą one dalej nazywane regułami CNF oraz regułami DNF. Jak zostało to jednak wcześniej wspomniane, metoda DeepRules umożliwia tworzenie zbiorów reguł zawierających jednocześnie reguły obydwu typów.

W tym celu algorytm w sposób dynamiczny dokonuje wyboru, jakiego rodzaju reguła ma zostać dodana jako następna. Proces budowania zbioru reguł jest oparty o sekwencyjne pokrywanie. Kolejne reguły generowane są na podstawie informacji i niepokrytych dotąd przykładach. Dodawanie reguł do zbioru trwa do momentu pokrycia zadanej części (domyślnie całego zbioru) przykładów w zbiorze treningowym, lub do momentu, gdy nie sposób już wygenerować kolejnej reguły spełniającej warunek minimalnego pokrycia. Proces ten, co do zasady, przypomina ten stosowany w algorytmie RuleKit. Różnica tkwi tutaj jednak w sposobie, w jaki wyznaczana jest nowa reguła.

Algorytm w każdej iteracji wyznacza dwie reguły, zarówno CNF, jak i DNF. W momencie, gdy nie sposób już wyznaczyć kolejnej reguły danego typu, która spełniałaby kryterium minimalnego pokrycia, jako lepsza wybierana jest reguła drugiego typu, którą udało się wyznaczyć. Gdy dla obydwu typów nie jest możliwe wygenerowanie kolejnej reguły, proces budowy zbioru reguł kończy się. W momencie gdy obydwie reguły zostały wyznaczone, algorytm dokonuje wyboru jednej z nich, kierując się kryterium jakości predykcji. Proces ten został przedstawiony na pseudokodzie 7. Jako jakość reguły brana jest jakość predykcji zbioru reguł, rozszerzonego o tę regułę. Dla różnych typów problemów zostały zastosowane odmienne kryteria. W przypadku klasyfikacji zostało użyte tzw. balanced accuracy [28] (oznaczane też jako BAcc), dla regresji względny błąd średniokwadratowy (patrz równanie 6.17), a dla analizy przeżycia zintegrowany błąd Briera (z ang. Integrated Brier Score IBS, patrz równanie 6.18).

$$RRMSE = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \overline{y}\right)^2}}{\frac{\sum_{j=1}^{n} y_j}{n}} \tag{6.17}$$

$$IBS = \int_{t_0}^{t_{max}} BS^c(t)\, dw(t) \tag{6.18}$$

$$\mathrm{BS}^c(t) = \frac{1}{n}\sum_{i=1}^{n} I(y_i \leqslant t \ \wedge \delta_i = 1)\frac{(0 - \hat{\pi}(t|\mathbf{x}_i))^2}{\hat{G}(y_i)} + I(y_i > t)\frac{(1 - \hat{\pi}(t|\mathbf{x}_i))^2}{\hat{G}(t)} \tag{6.19}$$

3.3 Przykład

Aby lepiej wyjaśnić działanie algorytmu, zostanie ono przedstawione na prostym przykładzie. Niech zbiór danych D składa się z czterech atrybutów a_1, a_2, a_3 i a_4 oraz

Algorytm 7 Wybór reguły

Wejście:
1: $ruleset$ — aktualna postać zbioru reguł.
2: r_{dnf} — reguła DNF.
3: r_{cnf} — reguła CNF.
4: D — zbiór treningowy.
5: **function** WybierzNajlepszaRegule($ruleset$, r_{dnf}, r_{cnf}, D)
6: **if** $|r_{dnf}| = 0 \wedge |r_{cnf}| = 0$ **then**
7: **return** $\varnothing$
8: **end if**
9: $q_{current} \leftarrow$ ObliczJakoscPredykcji($ruleset, D$)
10: **if** $|r_{dnf}| > 0$ **then**
11: $q_{dnf} \leftarrow$ ObliczJakoscPredykcji($ruleset_{dnf}, D$)
12: **else**
13: $q_{dnf} \leftarrow -\infty$
14: **end if**
15: **if** $|r_{cnf}| > 0$ **then**
16: $q_{cnf} \leftarrow$ ObliczJakoscPredykcji($ruleset_{cnf}, D$)
17: **else**
18: $q_{cnf} \leftarrow -\infty$
19: **end if**
20: **if** $q_{dnf} > q_{current}$ **then**
21: **return** r_{dnf}
22: **else if** $q_{cnf} > q_{current}$ **then**
23: **return** r_{cnf}
24: **end if** ▷ Obydwie reguły mają te samą jakość
25: $p_{dnf} \leftarrow$ WyznaczMaleP(r_{dnf})
26: $p_{cnf} \leftarrow$ WyznaczMaleP(r_{cnf})
27: **if** $p_{cnf} > p_{dnf}$ **then**
28: **return** r_{cnf}
29: **else**
30: **return** r_{dnf}
31: **end if**
32: **end function**

atrybutu decyzyjnego δ, określającego przynależność przykładu do jednej z dwóch klas: 0 i 1. Dla uproszczenia wszystkie atrybuty są atrybutami binarnymi.

Algorytm rozpoczyna działanie od pustego zbioru reguł. Indukcja reguł dla problemu klasyfikacji, podobnie jak w poprzednich metodach, odbywa się dla każdej z klas z osobna. W pierwszej kolejności poszukiwana jest reguła dla klasy pozytywnej. W tym celu generowane są dwie reguły, pierwsza postaci CNF, druga w DNF. Dokładny przebieg fazy wzrostu reguły DNF został przedstawiony w tabeli 6.4. Pokazuje ona, jak zmieniała się jakość reguły (kolumna pierwsza), wraz z dodawaniem kolejnych warunków do jej przesłanki (kolumna druga). Warunek dodawany z danej iteracji został dodatkowo podkreślony. Dla zachowania bardziej zwięzłego zapisu warunki proste typu $a_i = \{1\}$, ze względu na fakt, że wszystkie atrybuty zbioru są binarne, zapisane zostały w skrótej postaci a_i. Na podstawie tejże tabeli możemy zauważyć, że jakość reguły wzrastała w pierwszych trzech iteracjach, zmalała w czwartej, by osiągnąć swoje maksimum w iteracji piątej. Na koniec fazy wzrostu wybrana zostaje postać reguły o najwyższej jakości. W tym wypadku jest to reguła odpowiadająca przedostatniemu wierszowi tabeli 6.4.

Tabela 6.4: Tabela przedstawiająca przebieg fazy wzrostu przykładowej reguły DNF

Jakość reguły	Przesłanka reguły po dodaniu kolejnego warunku
0.2	$\underline{a_1}$
0.4	$a_1 \wedge \underline{a_2}$
0.6	$(a_1 \wedge a_2) \vee \underline{a_3}$
0.5	$(a_1 \wedge a_2) \vee (a_3 \wedge \underline{a_4})$
0.9	$(a_1 \wedge a_2) \vee (a_3 \wedge a_4) \vee \underline{a_1}$
0.8	$(a_1 \wedge a_2) \vee (a_3 \wedge a_4) \vee (a_1 \wedge \underline{a_4})$

Następnie przekazywana jest ona do procedury przycinania. W tym miejscu algorytm iteracyjnie usuwa kolejne warunki z przesłanki, maksymalizując jej jakość ocenianą miarą przycinania. Za każdym razem usunięty może być dowolny warunek z dowolnego składnika. Proces trwa aż do momentu, gdy usunięcie żadnego z warunków nie powoduje dalszego wzrostu jakości reguły lub też w jej przesłance zostanie tylko jeden warunek.

Dokładny przebieg przycinania opisywanej reguły DNF został przedstawiony w tabeli 6.5. Przedstawia ona zmiany jakości reguły wraz z usuwaniem kolejnych warunków z jej przesłanki. Usuwane warunki zostały zaznaczone jako przekreślone.

W przeciwieństwie do fazy wzrostu, w przypadku fazy przycinania możliwy jest tylko ciągły wzrost jakości reguły.

Tabela 6.5: Tabela przedstawiająca przebieg fazy przycinania przykładowej reguły DNF

Jakość reguły	Przesłanka reguły po usunięciu kolejnego warunku
0.92	$(a_1 \wedge a_2) \vee (a_3 \wedge a_4) \vee (\cancel{a_4} \wedge a_4)$
0.94	$(a_1 \wedge a_2) \vee (\cancel{a_3} \wedge a_4) \vee (\cancel{a_4} \wedge a_4)$

Dla uproszczenia szczegółowy przebieg wzrostu i przycinania reguły CNF zostanie w tym miejscu pominięty. Poniżej przedstawione zostały finalne postacie reguł DNF i CNF.

$$\text{DNF: } (a_1 \wedge a_2) \vee a_4 \vee a_4$$
$$\text{CNF: } (a_1 \vee a_3) \wedge (a_2 \vee a_4)$$

Następnie algorytm wybiera, która spośród dwóch utworzonych reguł zostanie dodana do wynikowego zbioru. W tym celu tworzone są dwa tymczasowe zbiory reguł, jeden z dodaną regułą CNF i drugi z regułą DNF. Obydwa są następnie oceniane pod kątem określonego kryterium predykcji. W przypadku problemu klasyfikacji jest nim tzw. balanced accuracy. Wybierana jest reguła, której dodanie minimalizuje błąd predykcji. W przypadku gdy dla obydwu reguł wartość kryterium jest taka sama, decyzja zostaje podjęta na podstawie pokrycia reguły. Wybrana zostaje wtedy reguła o większym p. W przypadku problemów regresji i analizy przeżycia, wartości p ustalane są z użyciem identycznego triku, jak ma to miejsce w przypadku algorytmu RuleKit (patrz rozdział 6 podsekcja 1.1).

Zakładając, że BAcc zbioru reguł po dodaniu przykładowej reguły DNF wyniosło 0.75 a po dodaniu reguły CNF był równy 0.68, wybrana zostanie reguła DNF. Zostanie więc ona dodana do wynikowego zbioru reguł. Przykłady pokryte przez nią zostają następnie usunięte ze zbioru treningowego. Należy w tym miejscu zaznaczyć jednak, że krok wyboru reguły na podstawie kryterium predykcji dokonywany jest zawsze na całym zbiorze treningowym.

Algorytm przechodzi następnie do generowania kolejnej reguły. Pomijając w tym miejscu proces wzrostu i przycinania, załóżmy, że utworzona została następująca reguła DNF:

$$\text{JEŻELI } (a_2 \wedge \neg a_3) \vee \neg a_2 \text{ THEN } \delta = \{1\}$$

W trakcie wzrostu reguły CNF nie udało się jednak uzyskać reguły spełniającej kryterium minimalnego pokrycia. W takiej sytuacji reguła DNF zostaje dodana do zbioru automatycznie z pominięciem kroku oceny kryterium predykcji.

Generując kolejną regułę, nie udaje się uzyskać takiej spełniającej kryterium minimalnego pokrycia zarówno dla postaci CNF, jak i DNF. Zakańcza to całkowicie proces indukcji zbioru reguł dla tejże klasy.

Dalszy proces indukcji reguł dla pozostałych klas następuje analogicznie.

3.4 Parametry algorytmu

Niniejsza podsekcja zawiera zestawienie parametrów algorytmu DeepRules wraz z opisem ich działania. Część z nich ma działanie identyczne do niektórych parametrów algorytmu RuleKit.

- ***min_cov*** – Odpowiada parametrowi *min_rule_covered* algorytmu RuleKit; domyślna wartość pozostaje bez zmian.
- ***max_uncovered_fraction*** – Odpowiada parametrowi algorytmu RuleKit o tej samej nazwie; domyślna wartość pozostaje bez zmian.
- ***voting_measure*** – Odpowiada parametrowi algorytmu RuleKit o tej samej nazwie; domyślnie miara Correlation (patrz równanie 6.3).
- ***enable_pruning*** – Odpowiada parametrowi algorytmu RuleKit o tej samej nazwie, domyślna wartość pozostaje bez zmian.
- ***max_layers_count*** – Dodatnia liczba całkowita określająca maksymalną liczbę składników (alternatyw lub koniunkcji) w regule; domyślnie 10.
- ***max_component_length*** – Dodatnia liczba całkowita określająca maksymalną liczbę warunków w pojedynczym składniku (alternatywie lub koniunkcji) reguły; domyślnie 3.
- ***dnf_quality_measure*** – Miara jakości stosowana przy wyborze najlepszego warunku dodawanego w fazie wzrostu reguł DNF; domyślnie miara Correlation (patrz równanie 6.3).
- ***dnf_quality_measure*** – Miara jakości stosowana do wyboru najlepszej postaci reguły DNF, spośród wszystkich uzyskanych w trakcie jej fazy wzrostu; domyślnie miara C2 (patrz równanie 6.2).

- ***dnf_pruning_measure*** – Miara jakości stosowana do oceny reguły DNF w fazie przycinania; domyślnie miara C2 (patrz równanie 6.2).
- ***cnf_quality_measure*** – Miara jakości stosowana przy wyborze najlepszego warunku dodawanego w fazie wzrostu reguł CNF; domyślnie miara C2 (patrz równanie 6.2).
- ***cnf_select_best_candidate_measure*** – Miara jakości stosowana do wyboru najlepszej postaci reguły CNF, spośród wszystkich uzyskanych w trakcie jej fazy wzrostu; domyślnie miara Correlation (patrz równanie 6.3).
- ***cnf_pruning_measure*** – Miara jakości stosowana do oceny reguły CNF w fazie przycinania; domyślnie miara C2 (patrz równanie 6.2).
- ***enable_attributes_conditions*** – Flaga kontrolująca czy algorytm wykorzystuje w trakcie indukcji warunki relacji atrybutów; domyślnie włączone.
- ***enable_negations*** – Flaga kontrolująca czy algorytm wykorzystuje w trakcie indukcji warunki zanegowane; domyślnie włączone.

Algorytm, w przeciwieństwie do dwóch wcześniejszych, posiada aż siedem parametrów określających miary jakości reguły. Dzieje się tak ze względu na fakt, że metoda generuje wewnętrznie dwa rodzaje reguł (DFN i CNF), a następnie w każdej iteracji wybiera tę lepszą pod kątem kryterium predykcji. Każdy z procesów indukcji reguł DNF i CNF stosuje trzy miary jakości. Jedna do wyboru najlepszego warunku w fazie wzrostu reguły (parametry dnf_quality_measure i cnf_quality_measure). Druga służy do oceny reguły w trakcie jej przycinania (parametry dnf_pruning_measure i cnf_pruning_measure). Trzecia z nich (parametry dnf_select_best_candidate-_measure i cnf_select_best_candidate_measure) wymaga dłuższego wyjaśnienia. Wzrost reguły i składnika w metodzie DeepRules dokonywany jest aż do spełnienia określonych kryteriów stopu (patrz pseudokod 5 i 6). W każdej iteracji do składnika reguły dodawany jest pojedynczy warunek, którego dodanie prowadzi do postaci reguły o najwyższej wartości miary jakości. Nie gwarantuje to jednak, podobnie jak w przypadku algorytmu RuleKit, że jakość ta rośnie przez cały czas trwania procesu wzrostu. W przypadku RuleKit oraz ComplexConditions, dostępny był specjalny parametr select_best_candidate w postaci flagi, który włączony sprawiał, że zapamiętywane były wszystkie pośrednie postacie reguły uzyskiwane w trakcie fazy wzrostu wraz z ich wartościami miary (patrz rozdział 6 podsekcja 1.4). Na

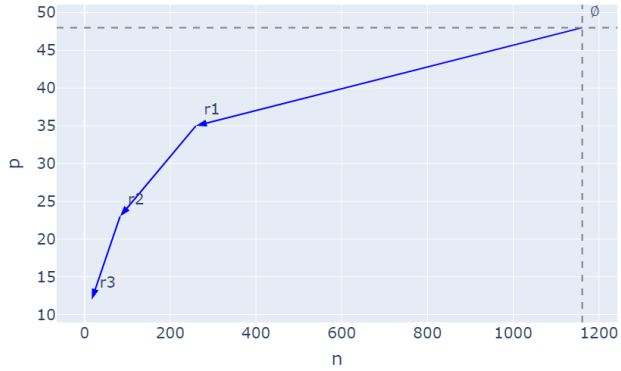
koniec zwracana była ta postać, dla której wartość miary była najwyższa. W przypadku metody DeepRules, takie zachowanie jest zachowaniem domyślnym, jednak sama reguła w trakcie wzrostu oceniana jest dwiema miarami. Pierwsza z nich, już omawiana, stosowana jest przy wyborze kolejnego dodawanego warunku. Wartości drugiej spośród nich (parametry dnf_select_best_candidate_measure i cnf_select-_best_candidate_measure) zapamiętywane są po dodaniu każdego z warunków, dla wszystkich przejściowych postaci reguły. Na koniec procesu wzrostu wybierana jest ta postać, dla której wartość owej drugiej miary była najwyższa.

3.5 Analiza złożoności obliczeniowej metody DeepRules

Algorytm DeepRules umożliwia generowanie reguł mających odmienną strukturę niż w przypadku wcześniejszych metod. Każda z nich może zawierać przesłankę w postaci CNF lub DNF. W pierwszym kroku szacowania złożoności tej metody należy zweryfikować, czy zmiana ta wpływa na maksymalną liczbę warunków w pojedynczej regule. Do wyznaczenia tej liczby będzie pomocna wizualizacja fazy wzrostu reguły w postaci łamanej w przestrzeni pokrycia. Łamaną tę nazwiemy, dla potrzeb dalszej analizy, krzywą wzrostu reguły. Podobny sposób prezentacji został zastosowany przez Beck i innych w pracy zatytułowanej „Layerwise Learning of Mixed Conjunctive and Disjunctive Rule Sets" [11]. Pozwala on zilustrować, w jaki sposób zmienia się pokrycie reguły wraz z jej wzrostem.

W pierwszym kroku zostanie zaprezentowana krzywa dla przykładowej reguły, gdzie przesłanka jest koniunkcją warunków elementarnych (rysunek 6.4). Oś pozioma reprezentuje liczbę negatywnych, a pionowa pozytywnych przykładów pokrywanych przez regułę. Wzrost reguły rozpoczyna się od punktu $\varnothing$, a więc reguły z pustą przesłanką pokrywającą wszystkie przykłady zbioru treningowego. Kierunek strzałek pokazuje kolejność dodawania kolejnych warunków do przesłanki. Każdy z punktów $r1, r2, r3$ reprezentuje kolejne przejściowe postacie reguły, gdzie $r3$ oznacza ostateczną formę reguły po zakończeniu jej wzrostu.

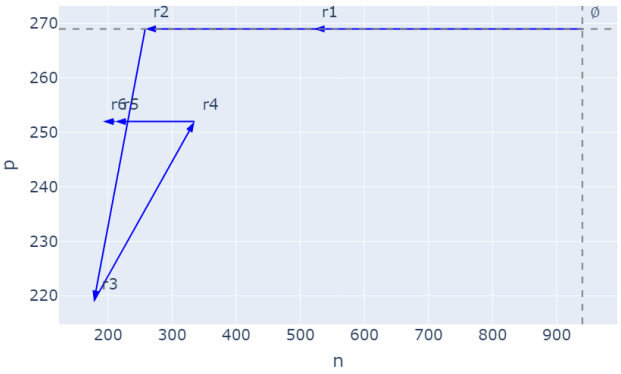
Dodawanie kolejnych warunków do przesłanki będącej koniunkcją może jedynie doprowadzić do reguły, która pokrywa mniejszą liczbę przykładów. Z tego też powodu można zaobserwować, że wraz z kolejnymi iteracjami krzywa „wędruje" jedynie w lewo i w dół wykresu. W pesymistycznym przypadku każdy kolejny warunek zmniejsza pokrycie reguły o jeden przykład (jednostkowe przesunięcie w lewo lub w dół).



Rysunek 6.4: Krzywa wzrostu reguły, której przesłanka ma postać koniunkcji.

Sytuacja wygląda jednak odmiennie dla reguł generowanych przez algorytm DeepRules. Rozważania zacznijmy od reguł DNF, gdzie przesłanka może przybrać postać alternatywy koniunkcji. Rysunek 6.5 przedstawia krzywą wzrostu dla poniższej reguły wygenerowanej dla popularnego zbioru benchmarkowego car[2] [23]

IF $(safety \neq \text{low} \wedge persons \neq 2 \wedge maint \neq \text{vhigh}) \vee$
$(safety = \text{high} \wedge persons \neq 2 \wedge buying \neq \text{high})$ **THEN** $class = \text{acc}$



Rysunek 6.5: Krzywa wzrostu reguły DNF.

Przesłanka reguły zawiera dwa składniki zawierające trzy warunki, stąd liczba punktów na rysunku 6.5 wynosi 6. Można zaobserwować, że dla trzech pierwszych

[2]Link do zbioru danych car: `https://archive.ics.uci.edu/dataset/19/car+evaluation`, [dostęp 16.09.2025].

pokrycie reguły malało, podobnie jak dla wcześniejszego przypadku. Z momentem dodania czwartego warunku pokrycie jednak ponownie wzrosło. Pośrednia postać reguły w punkcie r_4 prezentuje się następująco:

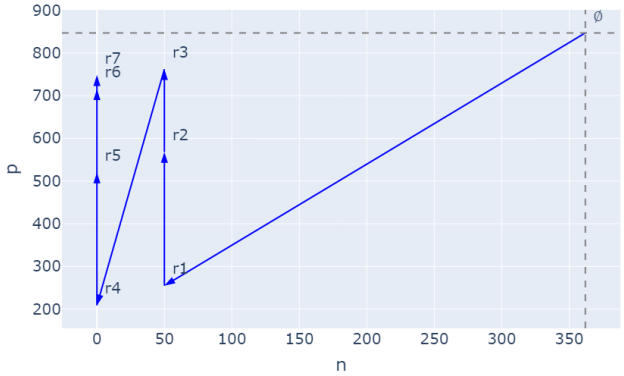
$$\textbf{IF } (safety \neq \text{low} \wedge persons \neq 2 \wedge maint \neq \text{vhigh}) \vee$$
$$safety = \text{high } \textbf{THEN } class = f\{\text{acc}\}$$

Jak widać, dodawany warunek został połączony z wcześniejszym składnikiem operatorem alternatywy logicznej. Z tego powodu pokrycie reguły, zamiast spadać, jak miało to miejsce wcześniej, wzrosło. Z racji specyfiki algorytmu każdy punkt musi mieć unikalne współrzędne. Dwa punkty o takich samych oznaczałyby, że algorytm uzyskał już wcześniej postać reguły o identycznym pokryciu.

Przykładowa krzywa dla reguły CNF przedstawiona na rysunku 6.6. Została ona wyrysowana dla następującej reguły:

$$\textbf{IF } (maint \neq \text{vhigh} \vee safety = low \vee persons = 2) \wedge$$
$$(buying = \text{high} \vee safety = low \vee persons = 2 \vee buying = vhigh)$$
$$\textbf{THEN } class = f\{\text{acc}\}$$

Jak widać w tym przypadku, pokrycie rośnie wraz z dodawaniem warunków do składnika. W momencie, gdy dodawany jest nowy składnik, zawsze spada.



Rysunek 6.6: Krzywa wzrostu reguły CNF.

Warto w tym miejscu zauważyć, że owe różnice spotykane dla reguł generowanych algorytmem DeepRules mogą być postrzegane jako swoista zaleta metody. Idealną sytuacją w trakcie wzrostu reguły jest oczywiście dotarcie w lewy górny róg wykresu,

pokrywając wszystkie przykłady pozytywne i żadnego negatywnego. W przypadku reguł, gdzie przesłanka jest koniunkcją warunków, dodawanie kolejnych warunków prowadzić może tylko do coraz bardziej specyficznych reguł. Każde zmniejszenie p jest w tym wypadku nieodwracalne. W przypadku reguł generowanych z użyciem DeepRules sytuacja jest jednak odmienna. Algorytm w trakcie fazy wzrostu może w bardziej swobodny sposób poruszać się po przestrzeni pokrycia. Nawet gdy dodanie pewnego warunku zredukuje wartość p kosztem zmniejszenia n, może ona zostać z powrotem uogólniona poprzez dodanie kolejnego warunku (patrz rysunek 6.6 punkt r_2).

Omówione przed chwilą krzywe wzrostu reguły dają pogląd na to, jak może wyglądać górne ograniczenie liczby warunków w regułach CNF i DNF. W trakcie wzrostu algorytm może poruszać się w każdym kierunku, odwiedzając każdy możliwy punkt na wykresie tylko raz. Na tej podstawie można ustalić, że reguła taka może zawierać maksymalnie $\frac{1}{2}(n+1)^2$ warunków. Stanie się tak dla przestrzeni pokrycia, gdzie liczba przykładów pozytywnych i negatywnych będzie równa, a reguła „odwiedzi" każdy możliwy punkt. W takim wypadku p i n reguły mogą przyjmować dowolne wartości z przedziału $[0, \frac{1}{2}n]$.

W przypadku algorytmu DeepRules, w trakcie indukcji rozpatrywane są tylko niektóre spośród warunków złożonych stosowanych w metodzie ComplexRules. Są to warunki relacji atrybutów oraz negacje warunków prostych. Pozostałe typy warunków mogą zostać uzyskane w sposób pośredni na podstawie uzyskanych koniunkcji i alternatyw logicznych (patrz sekcja 3.1). Na tej podstawie, opierając się na tabeli 6.2, możemy określić maksymalną liczbę warunków rozpatrywanych w trakcie indukcji reguł dla pojedynczego atrybutu jako $(4W-4)+(3m^2-3m)+(\frac{1}{3}m^3-\frac{1}{3}m)$. Pierwszy nawias wyrażenia odpowiada liczbie warunków prostych wraz z ich negacjami. Drugi i trzeci nawias reprezentują liczbę warunków relacji kolejno dla atrybutów numerycznych i nominalnych.

W oparciu o wszystkie powyższe informacje, stosując wzór 6.4, możliwe jest oszacowanie złożoności obliczeniowej algorytmu DeepRules. Została ona przedstawiona wzorem 6.20.

$$O(Wn^3m^3) \tag{6.20}$$

Dla przypadku pesymistycznego, gdzie $W = n$, przybierze ona postać zadaną wzorem 6.21.

$$O(n^4 m^3) \tag{6.21}$$

Jak można tutaj zauważyć, algorytm DeepRules posiada znacząco niższą złożoność od dwóch pozostałych metod względem liczby wierszy n. Jej wzrost względem liczby kolumn jest tutaj identyczny jak w przypadku ComplexRules i MofNRules. Dzieje się tak ze względu na fakt, że warunki relacji atrybutów nie są możliwe do uzyskania i zapisania w postaci prostej formuły CNF i DNF.

4 Posumowanie

Proponowane w niniejszej pracy metody adresują problem głębokiego dyskretnego uczenia (w rozumieniu zgodnym z opisem z rozdziału 5) w dwojaki sposób. Po pierwsze poprzez użycie w przesłankach reguł złożonych warunków elementarnych (metoda ComplexConditions i MofNRules). Po drugie poprzez tworzenie reguł z zagnieżdżonymi, hierarchicznymi formułami logicznymi (metoda DeepRules).

Kolejność, w jakiej zaprezentowane zostaną algorytmy, ilustruje rozwój prac badawczych prowadzonych przez autora. Każda kolejna metoda umożliwia odkrywanie bogatszych i bardziej złożonych wzorców w danych niż poprzednia. Ostatnia z metod DeepRules adresuje dodatkowo niektóre z problemów wydajnościowych. jakie dotyczą dwóch wcześniejszych metod.

Dla lepszego zobrazowania podobieństw i różnic pomiędzy poszczególnymi metodami, poniżej zamieszczona została tabela 6.6 zestawiająca je wszystkie.

Wszystkie trzy metody generują reguły poprzez zachłanny algorytm sekwencyjnego pokrywania. Za każdym razem w fazie wzrostu i przycinania reguły wybierane są warunki prowadzące do najlepszej wartości heurystycznej funkcji jakości. Algorytm ComplexConditions do generowania reguł z warunkami wewnętrznych alternatyw dla atrybutów numerycznych, stosuje dodatkową procedurę zachłannego przeszukiwania wiązką. W przypadku MofNRules, do wyszukiwania potencjalnie użytecznych warunków M-z-N stosowana jest natomiast heurystyka oparta o poszukiwanie zbiorów częstych z użyciem algorytmu FP-Growth.

Każdą z proponowanych metod można również postrzegać jako metodę konstruktywnej indukcji, gdzie nowo utworzonymi cechami są generowane warunki złożone oraz zagnieżdżone formuły logiczne. ComplexConditions i DeepRules zaklasyfikować można do metod konstruktywnej indukcji opartej o dane. Obydwie metody generują złożone zbiory reguł, testując pewien zbiór hipotez, kierując się bezpośrednio ich

Tabela 6.6: Zestawienie opracowanych metod

Cecha	ComplexConditions	MofNRules	DeepRules
Sposób generowania reguł	Zachłanne przeszukiwanie	Zachłanne przeszukiwanie oraz wyszukiwanie zbiorów częstych	Zachłanne przeszukiwanie, (ograniczona przestrzeń poszukiwań)
Typ stosowanej konstruktywnej indukcji	Sterowana danymi	Mieszana (sterowana danymi oraz hipotezami)	Sterowana danymi
Warunki zanegowane	Tak	Tak	Tak
Warunki relacji atrybutów	Tak	Tak	Tak
Warunki wewnętrznych alternatyw	Tak (tylko takie gdzie wszystkie warunki składowe są warunkami prostymi opartymi o ten sam atrybut)	Tak (tylko takie gdzie wszystkie warunki składowe są warunkami prostymi opartymi o ten sam atrybut)	Tak (oparte o dowolne warunki i atrybuty)
Warunki M-z-N	Nie	Tak	Tak (w sposób pośredni, z ograniczeniem jednego warunku per reguła)
Rodzaje generowanych zbiorów reguł	DNF	DNF	Każda z reguł może mieć postać DNF lub CNF

oceną na zbiorze treningowym. MofNRules prezentuje z kolei podejście mieszane. Indukując pierwszy zbiór reguł w ramach pierwszego kroku indukcji, podobnie jak dla ComplexConditions, tutaj stosowana jest konstruktywna indukcja sterowana danymi. Reguły z warunkami M-z-N tworzone są poprzez ponowną indukcję reguł na rozszerzonej przestrzeni atrybutów, co postrzegać można jako konstruktywną indukcję sterowaną hipotezami.

Jedną z głównych różnic pomiędzy proponowanymi metodami jest zestaw stosowanych przez nie warunków. Dla metody ComplexConditions są to warunki: proste, zanegowane, relacji atrybutów oraz wewnętrznych alternatyw. Te ostatnie jednak podlegają pewnym ograniczeniom związanym z wysoką złożonością obliczeniową algorytmu. Jednym z nich jest fakt, że mogą one być budowane tylko w oparciu o warunki dotyczące tego samego atrybutu. Metoda MofoNRules również nie jest wolna od tych ograniczeń, wprowadza jednak do reguł dodatkowo warunki M-z-N. Dopiero ostatni algorytm DeepRules pozwala tworzyć reguły z warunkami wewnętrznych alternatyw zawierających dowolne warunki oparte o dowolne atrybuty zbioru

treningowego. Z racji zasady działania metoda wprowadza jednak pewne ograniczenia dotyczące warunków M-z-N, które wyszukiwane są w sposób pośredni, na podstawie formuł logicznych obecnych w przesłankach reguł. Z tego powodu przesłanki reguł zawierać mogą jedynie pojedynczy warunek M-z-N. One same zaś nie mogą zawierać w sobie warunków zbioru dyskretnego ani wewnętrznych alternatyw.

Zarówno ComplexConditions, jak i MofNRules generują zbiory reguł w postaci DNF, gdzie przesłanka pojedynczej reguły ma postać koniunkcji literałów (warunków). Ze stwierdzeniem tym można polemizować, jako że poszczególne reguły mogą zawierać tutaj warunki wewnętrznych alternatyw. Z racji ograniczeń metod dotyczących ich wyszukiwania oraz samej metodyki są one jednak traktowane jako pojedyncze literały. W taki sam sposób są one również oceniane w trakcie procesu indukcji reguł. W przypadku algorytmu DeepRules, przesłanka każdej z wygenerowanych reguł może mieć postać zarówno koniunkcji alternatyw (CNF) jak i alternatywy koniunkcji (DNF) literałów. W skrajnym przypadku, gdy każdy składnik reguły posiada tylko jeden warunek, możliwe jest uzyskanie klasycznych zbiorów reguł w formacie DNF.

Zaprezentowane w niniejszym rozdziale oszacowania złożoności proponowanych metod pokazują jasno, że uzyskiwanie bardziej kompleksowych reprezentacji regułowych nieodłącznie wiąże się ze wzrostem złożoności obliczeniowej procesu indukcji. Należy jednak zaznaczyć, że ów wzrost był w tym wypadku spodziewany i zgodny z powszechnie znanym „twierdzeniem o nieistnieniu darmowych obiadów" (z ang. „no free lunch" (NFL)) [190].

Zgodnie z intuicją i oczekiwaniami, metoda DeepRules posiada mniejszą pesymistyczną złożoność względem liczby wierszy w zbiorze treningowym. W przeciwieństwie do pozostałych dwóch metod rośnie ona liniowo względem średniej liczby unikalnych wartości atrybutów.

Opracowane metody wciąż posiadają jednak wysoką złożoność, co ogranicza ich użyteczność w przypadku naprawdę dużych zbiorów danych. Należy jednak podkreślić, że przeprowadzone w tym rozdziale rozważania dotyczyły przypadku pesymistycznego. Wiele spośród poczynionych założeń jest mało prawdopodobnych do spełnienia dla rzeczywistych zbiorów danych. Przykładowo zbiór reguł, gdzie ich liczba równa jest liczbie wierszy w danych, jest sytuacją rzadką. Co więcej, reguły takie nie niosłyby ze sobą żadnej wartości w kontekście odkrywania wiedzy i eksploracji danych. Podobnie reguła posiadająca n warunków w przesłance, nawet w przypadku małych zbiorów danych, jest najczęściej całkowicie nieinterpretowalna i nieporządna.

Problem wysokiej złożoności obliczeniowej został częściowo zaadresowany poprzez parametry algorytmów. Każda z proponowanych metod posiada opcję ograniczenia maksymalnej liczby warunków w regule. Możliwe jest także kontrolowanie minimalnej liczby przykładów, jakie musi pokrywać nowo utworzona reguła, tak by unikać reguł pokrywających jeden lub kilka przykładów. Dodatkowo dla każdej z nich możliwe jest określenie maksymalnej części zbioru, która może zostać niepokryta przez reguły, co dodatkowo może zmniejszyć rozmiar wynikowego zbioru. Wszystko to sprawia, że w przypadku realnych zbiorów danych algorytmy te osiągają zwykle zdecydowanie lepszą złożoność od prezentowanych tutaj górnych oszacowań.

7 Eksperymenty

W niniejszym rozdziale opisano badania mające na celu ocenę proponowanych metod i ich porównanie pod kątem dwóch kryteriów: zdolności predykcyjnych oraz wielkości i złożoności uzyskiwanych zbiorów reguł. Ich wyniki zostały zestawione z tymi uzyskiwanymi przez algorytm RuleKit oraz drzewa decyzyjne. W przypadku tego pierwszego został on porównany w literaturze z innymi popularnymi metodami uczenia regułowego [75, 191]. Dlatego odniesienie do RuleKit pośrednio jest również odniesieniem do innych popularnych algorytmów indukcji reguł zarówno ze względu na ich zdolności predykcyjne, jak również na złożoność generowanych przez nie opisów. Jest tak dlatego, że stosujemy identyczne metody i miary oceny eksperymentalnej jak w przywoływanych publikacjach. Dodatkowo do zestawień zostały dodane wyniki osiągane przez drzewa decyzyjne, będące efektywną metodą budowy parami rozłącznych reguł. Do ich uczenia użyto implementacji dostarczanej przez popularny pakiet języka Python do uczenia maszynowego o nazwie scikit-learn[1] w wersji 1.2.2. Biblioteka ta zawiera implementacje drzew decyzyjnych dla problemów klasyfikacji i regresji, nie adresuje jednak problemu analizy przeżycia. W tym celu został użyty inny pakiet o nazwie scikit-survival[2] w wersji 0.21.0. Implementuje on algorytm uczenia drzew przeżyciowych zaproponowany przez LeBlanc i Crowley [103].

Prezentowane wyniki osiągane przez algorytm RuleKit oraz drzewa decyzyjne pozwalają ocenić, jak zaproponowane w niniejszej pracy metody wypadają na tle innych istniejących w literaturze rozwiązań. Eksperymenty przeprowadzono na możliwie dużej liczbie zbiorów danych dotyczących trzech typów problemów: klasyfikacji, regresji i analizy przeżycia. Wśród nich znajdują się zarówno rzeczywiste zbiory danych, odzwierciedlające realne zjawiska, jak i popularne zbiory syntetyczne. Dokładniejsza charakterystyka zbiorów znajduje się w sekcji 2. W trakcie badań poszukiwano

[1] Link do repozytorium pakietu scikit-learn: `https://github.com/scikit-learn/scikit-learn`, [dostęp 16.09.2025].

[2] Link do repozytorium pakietu scikit-survival: `https://github.com/sebp/scikit-survival`, [dostęp 16.09.2025].

odpowiedzi na dwa główne pytania: jak użycie bardziej złożonych postaci reguł i warunków wpływa na zdolności predykcyjne uzyskiwanych zbiorów reguł oraz jaki jest ich wpływ na złożoność modeli regułowych.

1 Metodologia przeprowadzonych badań

Badania zostały przeprowadzone z zastosowaniem 10-krotnej walidacji krzyżowej. W tym celu każdy z użytych zbiorów danych został podzielony w losowy sposób na 10 podzbiorów o możliwie równej wielkości (tzw. foldów). W przypadku zbiorów klasyfikacyjnych oraz dotyczących problemu analizy przeżycia, podział ten był dodatkowo stratyfikowany względem kolumny decyzyjnej. Sprawia to, że rozkład zmiennej decyzyjnej w poszczególnych podzbiorach jest możliwie najbardziej zbliżony do tego występującego w oryginalnym zbiorze. Następnie był przeprowadzany 10-krotny trening modeli. W każdej iteracji inny z podzbiorów pełnił rolę zbioru testowego, pozostałe zaś służyły jako dane treningowe [82]. Technika walidacji krzyżowej jest szczególnie przydatna w przypadku małych zbiorów danych, pozwalając, by każdy z przykładów w nich zawartych, użyty został w procesie uczenia modelu. Dodatkowo, dzięki 10-krotnemu powtórzeniu procesu treningu, zmniejsza ona wpływ losowego podziału przykładów pomiędzy zbiór treningowy i testowy na wyniki oceny modelu. Jedynymi zbiorami, dla których metoda walidacji krzyżowej nie została zastosowana, były *monk-1*, *monk-2* i *monk-3*. Są to popularne w literaturze zbiory syntetyczne o ustalonym z góry podziale na część treningową i testową.

Wyniki prezentowane w niniejszym rozdziale są wynikami osiągniętymi przez poszczególne metody, uśrednionymi dla wszystkich 10 foldów walidacji krzyżowej. Każda z metod została oceniona na identycznie podzielonych zbiorach danych w celu zachowania porównywalności wyników.

W celu oceny opracowanych algorytmów użyto 30 zbiorów klasyfikacyjnych i regresyjnych. W przypadku problemu analizy przeżycia ich liczba zbiorów była mniejsza i wyniosła 14. Mniejsza dostępność tego typu danych wynika m .in. z faktu, że ich gromadzenie jest często procesem kosztownym i długotrwałym, wymagającym np. śledzenia przebiegu leczenia pacjentów przez wiele lat. Ponadto dane te często dotyczą medycyny, gdzie liczne przepisy o ochronie prywatności (np. RODO, HIPAA) mogą utrudniać ich publiczne udostępnianie.

Ze względu na dużą liczbę analizowanych zbiorów oraz ich różnorodną tematykę, generowanie warunków relacji atrybutów (patrz rozdział 3 sekcja 6) dla wszystkich

zaproponowanych przez autora metod nie zostało poddane żadnym ograniczeniom. Oznacza to, że mogły być w nich porównywane dowolne atrybuty numeryczne lub dowolne atrybuty nominalne o tych samych domenach możliwych wartości.

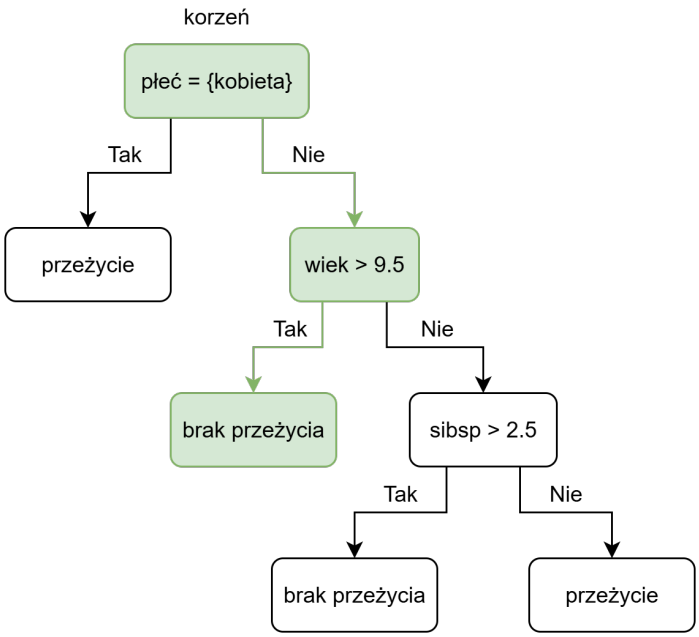
Zdolności predykcyjne testowanych algorytmów oceniono z użyciem popularnych, występujących w literaturze metryk. Dla klasyfikacji jest to tak zwane balanced accuracy (oznaczane skrótowo jako BAcc) [28]. Jest ono wyrażone wzorem 7.1, jako średnia wartości czułości (z ang. recall) modelu dla każdej z klas.Metryka ta pozwala w bardziej miarodajny sposób ocenić skuteczność klasyfikacji modelu dla problemów wieloklasowych i niezbalansowanych, niż ma to miejsce w przypadku klasycznego accuracy [28].

$$\mathrm{Bacc} = \frac{1}{N_{\mathrm{klas}}} \sum_{i=1}^{N_{\mathrm{klas}}} \left(\frac{\mathrm{TP}_i}{\mathrm{TP}_i + \mathrm{FN}_i} \right) \tag{7.1}$$

W przypadku zbiorów regresyjnych do oceny predykcji użyto względnego błędu średniokwadratowego, w skrócie RRMSE (z ang. relative root-mean-squared error), obliczanego zgodnie ze wzorem 6.17. Jego zaletą jest fakt, że zakres osiąganych przez niego wartości nie zależy od zakresu wartości zmiennej decyzyjnej. Ułatwia to porównywanie wyników osiągniętych przez metody na różnych zbiorach danych.

Dla analizy przeżycia został użyty całkowy wskaźnik Briera, w skrócie IBS. Obliczany jest on zgodnie ze wzorem 6.18. Jest to obecnie jedna z najpopularniej stosowanych w literaturze metryk do oceny modeli predykcyjnych w analizie przeżycia [71, 155]. Całkowy wskaźnik Briera pozwala obliczyć błąd predykcji modelu dla wszystkich czasów obserwacji, obecnych w zbiorze testowym. Im jego wartość jest mniejsza, tym dokładniejsza jest dana prognoza. Wartość zero oznacza perfekcyjne dopasowanie, z kolei 0.25 odpowiada modelowi losowemu, zwracającemu prawdopodobieństwo przeżycia równe 0.5 dla każdego przykładu.

W ramach badań przeprowadzono także analizę złożoności modeli regułowych generowanych przez poszczególne metody. W literaturze trudno spotkać jedną powszechnie przyjętą metrykę pozwalającą na jej ocenę. Stosowana w tym celu bywa między innymi sumaryczna liczba reguł. Nie jest ona jednak wystarczająca, ponieważ same reguły mogą różnić się od siebie długością przesłanki. Inną popularną metryką jest średnia liczba warunków w regule. Ta z kolei nie bierze pod uwagę liczby reguł. Model, posiadający dużą liczbę krótkich reguł, wciąż może nie być łatwy w interpretacji. Idealne kryterium brałoby pod uwagę zarówno to, ile reguł występuje w zbiorze, jak i ich długość (rozumianą zwykle jako liczbę warunków).



Rysunek 7.1: Przykładowe drzewo decyzyjne dla zbioru danych dotyczącego pasażerów titanica. Kolorem zielonym zaznaczono przykładową ścieżkę odpowiadającą pojedynczej regule decyzyjnej

Można w tym celu wykorzystać sumaryczną liczbę warunków w całym zbiorze reguł. Daje ona lepszy, lecz wciąż niekompletny pogląd na całościową złożoność modelu. Z tego powodu zdecydowano się przeanalizować wszystkie trzy wspominane metryki. Zdaniem autora pozwala to w możliwie najpełniejszy sposób ocenić złożoność modeli regułowych generowanych przez poszczególne metody. Warunki M-z-N oraz warunki wewnętrznych alternatyw w przedstawionych poniżej wynikach liczone są jako pojedynczy warunek.

Obliczenie wspomnianych charakterystyk dla drzew decyzyjnych nie jest jednak możliwe bez przekształcenia ich w zbiory parami rozłącznych reguł. Dokonywane jest to poprzez wyznaczenie wszystkich ścieżek od korzenia aż do liści drzewa. Każda taka ścieżka tworzy osobną regułę decyzyjną. Przykładowe drzewo decyzyjne zaprezentowane zostało na rysunku 7.1. Kolorem zielonym zaznaczono na nim wybraną ścieżkę odpowiadającą regule JEŻELI płeć $=$ {kobieta} $\wedge$ wiek $>$ 9.5 TO brak przeżycia.

Dla tak uzyskanych na podstawie drzewa reguł możliwe jest policzenie powyższych metryk i porównanie ich z wartościami uzyskanymi dla pozostałych badanych metod.

Samo zestawienie wartości metryk uzyskanych przez poszczególne metody pozwala wprawdzie dokonać ich oceny, nie daje jednak informacji o tym, czy różnice między nimi są faktycznie znaczące. Aby odpowiedzieć na to pytanie, wykorzystano testy statystyczne: Friedmana, Nemenyi i Wilcoxona. Pierwszy z nich pozwala sprawdzić, czy pomiędzy co najmniej jedną parą algorytmów występuje statystycznie istotna różnica. W tym celu każdej ocenianej metodzie, dla każdego badanego zbioru danych, przypisywana jest ranga, będąca kolejną liczbą całkowitą, począwszy od 1. Niższa ranga oznacza tutaj lepszy wynik metody. W ten sposób są one porządkowane od najlepszej do najgorszej. W przypadku, gdy więcej niż jedna metoda osiągnie ten sam wynik na tym samym zbiorze danych, wybierane są dla nich kolejne numery rang, a następnie przypisywana jest im wszystkim ich średnia. Następnie obliczana jest wartość statystyki, na której podstawie, poprzez porównanie jej z rozkładem chi-kwadrat, obliczana jest p-wartość. Jeżeli jest ona mniejsza niż określony poziom istotności, hipoteza zerowa jest odrzucana. Oznacza to, że istotna statystycznie różnica istnieje dla przynajmniej jednej pary metod.

Test Friedmana pozwala określić, czy wśród wszystkich metod istnieje przynajmniej jedna para różniąca się w istotny sposób. Nie odpowiada jednak na pytanie, dla których par wystąpiła taka różnica. Aby na nie odpowiedzieć, można skorzystać z tzw. testów post hoc. Demšar rekomenduje w swej pracy użycie testu Nemenyi do porównywania różnych algorytmów uczenia maszynowego na wielu zbiorach danych [49]. W ramach niego, na podstawie liczby porównywanych prób, wyznaczana jest tak zwana krytyczna różnica CD (z ang. critical difference). Jeżeli dla danej pary algorytmów ich rangi różnią się o wartość większą lub równą CD, uznawane jest, że różnica między nimi jest istotna. Garcia i Herrera sugerują jednak, że w niektórych przypadkach test Nemenyi porównujący wiele modeli na wielu zbiorach danych może okazać się zbyt konserwatywny, nie wykazując żadnych różnic pomimo ich obecności [89]. Z tego powodu zastosowany zostanie dodatkowy test par obserwacji Wilcoxona [184]. Porównuje on między sobą wszystkie pary algorytmów, obliczając dla nich wartość statystyki testu. Następnie na jej podstawie wyznaczana jest p-wartość. Jeżeli jest ona mniejsza od przyjętego poziomu istotności, odrzucana jest hipoteza zerowa o nieistnieniu statystycznie istotnych różnic pomiędzy parą pomiarów. Test Wilcoxona stanowi alternatywę testu t-Studenta dla prób zależnych, gdzie założenia testu t-Studenta nie są spełnione. W naszym konkretnym przypadku, testując wiele metod na tym samym zestawie zbiorów danych, pomiary takie traktować można jako próby zależne.

Przeprowadzanie testu Wilcoxona dla wielu par algorytmów wiąże się jednak z koniecznością stosowania korekty uzyskiwanych w ten sposób p-wartości. W tym wypadku została zastosowana popularnie wykorzystywana w literaturze korekta FDR (ang. false discovery rate) zaproponowana przez Benjamini i Hochberga [15]. Redukuje ona ryzyko błędów typu pierwszego, a więc ryzyko uznania wyników testów za istotne, gdy w rzeczywistości takimi nie są.

2 Zbiory danych użyte w eksperymentach

W trakcie przeprowadzonych eksperymentów zostało użytych 74 (30 klasyfikacyjnych, 30 regresyjnych i 14 dotyczących analizy przeżycia) publicznie dostępnych, tabelarycznych zbiorów danych. Każdy z nich dotyczył jednego z trzech typów problemów branych pod uwagę w badaniach: klasyfikacji, regresji i analizy przeżycia. Dokładną charakterystykę zbiorów danych użytych podczas eksperymentów, wraz z podstawowymi informacjami na ich temat, przedstawiono w tabelach 7.1, 7.2 oraz 7.3. Znakiem $*$ zostały oznaczone te pochodzące z popularnego repozytorium danych zwanego UC Irvine Machine Learning Repository[3]. Te oznaczone znakiem † zostały zaczerpnięte z powszechnie znanego serwisu związanego z uczeniem maszynowym o nazwie Kaggle[4]. Zbiory danych, przy których nazwie występuje znak ‡ pochodzą z kolei z repozytorium OpenML[5]. Wszystkie trzy wspomniane repozytoria stanowią popularne źródła danych dla rozmaitych badań związanych z uczeniem maszynowym.

Dane do analizy przeżycia oraz regresyjny zbiór *ozone*, zostały zaczerpnięte z innych źródeł, którymi w większości były różnorodne pakiety języka R . I tak zbiory *cancer*, *mgus* oraz *pbc* pochodzą z pakietu *survival*. Dane dla *follic*, *hd* i *lung* zostały uzyskane z pakietu *randomForestSRC*. Z kolei zbiór *GBSG2* zaczerpnięto z pakietu *TH.data*, *Melanoma* z *riskRegression*, a *zinc* z pakietu *NestedCohort*. Regresyjny zbiór danych *ozone* wzięty został z pakietu języka R o nazwie *mlbench*. W przypadku pozostałych dwóch zbiorów o nazwach *whas1* oraz *whas500*, pochodzą one z książki „Case Studies in Biometry" autorstwa Nicholasa Lange i innych [100].

Jak można stwierdzić na podstawie tabel 7.1, 7.2 oraz 7.3, do badań użyto różnorodnych danych. Spośród nich 23 zawierały jedynie atrybuty numeryczne, w tym 7 z nich dotyczyło problemu klasyfikacji, 12 regresji i 4 analizy przeżycia. 12 zbiorów

[3]Link do repozytorium danych UCI `https://archive.ics.uci.edu`, [dostęp 16.09.2025].
[4]Link do serwisu Kaggle `https://www.kaggle.com`, [dostęp 16.09.2025].
[5]Link do serwisu OpenML `https://www.openml.org`, [dostęp 16.09.2025].

Tabela 7.1: Charakterystyka zbiorów opisujących problemy klasyfikacji

#	Nazwa zbioru	Wiersze	Atrybuty numeryczne	Atrybuty nominale	Wartości puste	Klasy decyzyjne
1	anneal ∗	898	6	32	Nie	5
2	auto-mpg ∗	398	5	2	Tak	3
3	autos ∗	205	15	10	Tak	6
4	balance-scale ∗	625	4	0	Nie	3
5	bupa-liver-disorders ∗	345	6	0	Nie	2
6	car ∗	1728	0	6	Nie	4
7	cleveland ∗	303	6	7	Tak	5
8	cylinder-bands ∗	540	17	18	Tak	2
9	echocardiogram ∗	131	9	2	Tak	2
10	ecoli ∗	336	7	0	Nie	8
11	flag ∗	194	10	18	Nie	4
12	glass ∗	214	9	0	Nie	6
13	hayes-roth ∗	132	0	4	Nie	3
14	heart-c ∗	303	6	7	Tak	2
15	heart-statlog ∗	270	7	6	Nie	2
16	hepatitis ∗	155	6	13	Tak	2
17	iris ∗	150	4	0	Nie	3
18	lymphography ∗	148	3	15	Nie	4
19	monk-1 ∗	324	0	6	Nie	2
20	monk-2 ∗	369	0	6	Nie	2
21	monk-3 ∗	554	0	6	Nie	2
22	mushroom ∗	8124	0	22	Tak	2
23	nursery ∗	12960	0	8	Nie	5
24	sonar ∗	208	60	0	Nie	2
25	soybean ∗	683	0	35	Tak	19
26	tic-tac-toe ∗	958	0	9	Nie	2
27	titanic †	2201	0	3	Nie	2
28	vote ∗	435	0	16	Tak	2
29	wine ∗	178	13	0	Nie	3
30	zoo ∗	101	1	15	Nie	7

Tabela 7.2: Charakterystyka zbiorów opisujących problemy regresyjne

#	Nazwa zbioru	Wiersze	Atrybuty numeryczne	Atrybuty nominale	Wartości puste
1	auto-mpg ∗	398	4	3	Tak
2	auto-price ∗	159	14	1	Nie
3	auto93 ‡	93	16	6	Nie
4	bodyfat †	252	14	0	Nie
5	bolts ‡	40	7	0	Nie
6	breasttumor ∗	286	1	8	Nie
7	cholesterol ‡	303	6	7	Tak
8	cloud †	108	4	2	Nie
9	concrete ∗	103	9	0	Nie
10	cpu ‡	209	6	1	Nie
11	dee ‡	365	6	0	Nie
12	diabetes ∗	43	2	0	Nie
13	echomonths ∗	130	6	3	Tak
14	ele-1 †	495	2	0	Nie
15	elusage ‡	55	1	1	Nie
16	fishcatch ‡	158	5	2	Nie
17	fruitfly ‡	125	2	2	Nie
18	gascons ‡	27	4	0	Nie
19	kidney ∗	76	3	2	Nie
20	lowbwt ‡	189	2	7	Nie
21	machine ∗	209	6	0	Nie
22	mbagrade ‡	61	1	1	Nie
23	ozone	330	8	0	Nie
24	pharynx ‡	195	1	10	Nie
25	pollution ‡	60	15	0	Nie
26	pwlinear ‡	200	10	0	Nie
27	pyrim ‡	74	27	0	Nie
28	servo ∗	167	0	4	Nie
29	strike ‡	625	5	1	Nie
30	veteran ‡	137	3	4	Nie

Tabela 7.3: Charakterystyka zbiorów dotyczących analizy przeżycia

#	Nazwa zbioru	Wiersze	Atrybuty numeryczne	Atrybuty nominale	Wartości puste	Liczba wystąpień zdarzenia
1	bmt-ch *	187	10	26	Tak	85
2	cancer	228	8	0	Tak	165
3	follic	541	4	1	Nie	348
4	GBSG2	686	6	3	Nie	299
5	hd	865	3	4	Nie	426
6	lung [6]	1032	6	2	Tak	764
7	Melanoma	205	4	4	Nie	71
8	mgus	241	8	2	Tak	184
9	pbc	418	17	1	Tak	161
10	std	877	20	2	Nie	347
11	uis	575	14	0	Nie	464
12	whas1	481	8	0	Nie	249
13	whas500	500	14	0	Nie	215
14	zinc	431	47	9	Tak	81

Tabela 7.4: Statystyki zbiorów danych wykorzystanych w eksperymentach.

Problem	**Liczba zbiorów**		**Atrybuty**		**Wiersze**	
	Wszystkie	**Z wartościami brakującymi**	**Średnia**	**Min/Max**	**Średnia**	**Min/Max**
Klasyfikacja	30	10	15	3/60	1139	101/12 960
Regresja	30	3	8	2/27	189	27/625
Przeżycie	14	6	16	5/56	519	187/1032

zawierało tylko atrybuty nominalne, spośród nich 11 stanowiły zbiory klasyfikacyjne, a pozostały jeden był zbiorem regresyjnym. Pozostałe zawierały w sobie zarówno kolumny nominalne, jak i numeryczne. W przypadku klasyfikacji 14 zbiorów dotyczyło klasyfikacji binarnej, a pozostałe 16 wieloklasowej. 19 spośród wszystkich zestawów danych zawierało w sobie również wartości brakujące.

Użyte zbiory danych różniły się między sobą w znaczący sposób także pod względem liczby atrybutów, jak i przykładów. Najmniejszy zbiór zawierał jedynie 27 przykładów, a zbiór największy miał ich niemal 13 tysięcy. Minimalna liczba atrybutów występujących w zbiorze wyniosła 2, a największa 60.

[6]Link do zbioru danych dotyczącego wczesnego wykrywania raka płuc: `https://www.stats.ox.ac.uk/pub/datasets/csb/`, [dostęp 16.09.2025].

3 Parametryzacja badanych algorytmów

Z uwagi na użycie dużej liczby zbiorów danych, wszystkie eksperymenty przeprowadzono z użyciem tych samych wartości parametrów.

Do przeprowadzenia eksperymentów dla algorytmu RuleKit użyty został pakiet języka Python[7], w wersji 1.6.0, stanowiący pewnego rodzaju „nakładkę" na oficjalną implementację napisaną w języku programowania Java.

Podczas indukcji reguł metodami RuleKit, ComplexConditions oraz MofNRules zastosowano popularną miarę jakości reguł C2 [29,193] (patrz równanie (6.2). Została ona użyta w fazie wzrostu i przycinania reguł, a także do rozwiązywania konfliktów podczas głosowania. Decyzja o jej użyciu podyktowana była tym, że faworyzuje ona reguły o wysokiej precyzji i umiarkowanym pokryciu [162, 193]. Wpływ użycia różnych miar jakości na przebieg indukcji reguł został już dobrze przebadany i opisany w literaturze [164, 192]. Parametr *min_rule_covered* został dla wszystkich algorytmów regułowych ustawiony na 2. Jest to rozmiar najmniejszej spośród wszystkich klas występujących w użytych zbiorach danych. Wyższa wartość mogłaby prowadzić do zbioru reguł nieobejmujących wszystkich klas decyzyjnych. Ustawienie parametru *max_uncovered_fraction* sprawiło, że indukowane zbiory reguł nie musiały pokrywać wszystkich przykładów zbioru treningowego. Maksymalnie 2% przykładów mogło pozostać niepokrytych przez jakąkolwiek regułę. Taka konfiguracja pozwala algorytmowi ignorować przykłady odstające w zbiorze treningowym, czyniąc go mniej podatnym na przeuczenie. Dodatkowo parametr *select_best_candidate* został ustawiony na jeden dla metod RuleKit, ComplexConditions oraz MofNRules.

W przypadku eksperymentów dotyczących analizy przeżycia, podobnie jak w przypadku algorytmu RuleKit, do wzrostu, przycinania i wyboru najlepszego warunku został użyty test Logrank dla wszystkich proponowanych metod (patrz 6 podsekcja 1.1).

Dokładny wykaz wartości parametrów ustawianych dla metod proponowanych w ramach niniejszej pracy został przedstawiony w tabelach 7.5, 7.6 oraz 7.7.

W przypadku drzew decyzyjnych parametry starano się ustalić w taki sposób, by uzyskać możliwie podobne zachowanie jak dla algorytmu RuleKit. Maksymalna głębokość drzewa pozostawiona została bez ograniczeń. Dzięki temu długość reguł, rozumianych jako ścieżki od korzenia do liści, zawierać mogła dowolną liczbę węzłów.

[7]Link do repozytorium pakietu RuleKit: `https://github.com/adaa-polsl/RuleKit-python`, [dostęp 16.09.2025].

Tabela 7.5: Wartości parametrów metody ComplexConditions zastosowane podczas eksperymentów.

Nazwa parametru	Wartość
min_rule_covered	2
max_uncovered_fraction	0.02
select_best_candidate	true
induction_measure	C2
pruning_measure	C2
voting_measure	C2

Tabela 7.6: Wartości parametrów metody MofNRules (dla obydwu wariantów) zastosowane podczas eksperymentów.

Nazwa parametru	Wartość
min_rule_covered	2
max_uncovered_fraction	0.02
select_best_candidate	true
induction_measure	C2
pruning_measure	C2
voting_measure	C2
M	2
N	3
min_supp_fraction	0.2

Oddaje to zachowanie parametru *max_growing* ustawionego na zero dla algorytmu RuleKit. Minimalna liczba przykładów konieczna do utworzenia nowego podziału w drzewie ustalona została, przez analogię do parametru *min_rule_covered* algorytmu RuleKit, na tę samą wartość – 2.

Kryteria jakości podziału, zastosowane dla drzew klasyfikacyjnych i regresyjnych, różnią się jednak od miar jakości wykorzystanych w metodach regułowych. Dla problemu klasyfikacji zastosowany został współczynnik Giniego [27]. Określa on prawdopodobieństwo niepoprawnego sklasyfikowania przykładu, przypisując mu etykietę w sposób losowy na podstawie rozkładu klas decyzyjnych. Dla drzew regresyjnych zastosowanym kryterium jakości podziału jest błąd średniokwadratowy – MSE (z ang. mean squared error). Jedynie w przypadku analizy przeżycia zastosowane zostało kryterium oparte o test logrank, tak jak ma to miejsce dla pozostałych badanych metod.

Podczas poszukiwania zbiorów częstych w ramach metody MofNRules, minimalny próg wsparcia dla algorytmu FP Growth został ustawiony na 0.2. Przed dokona-

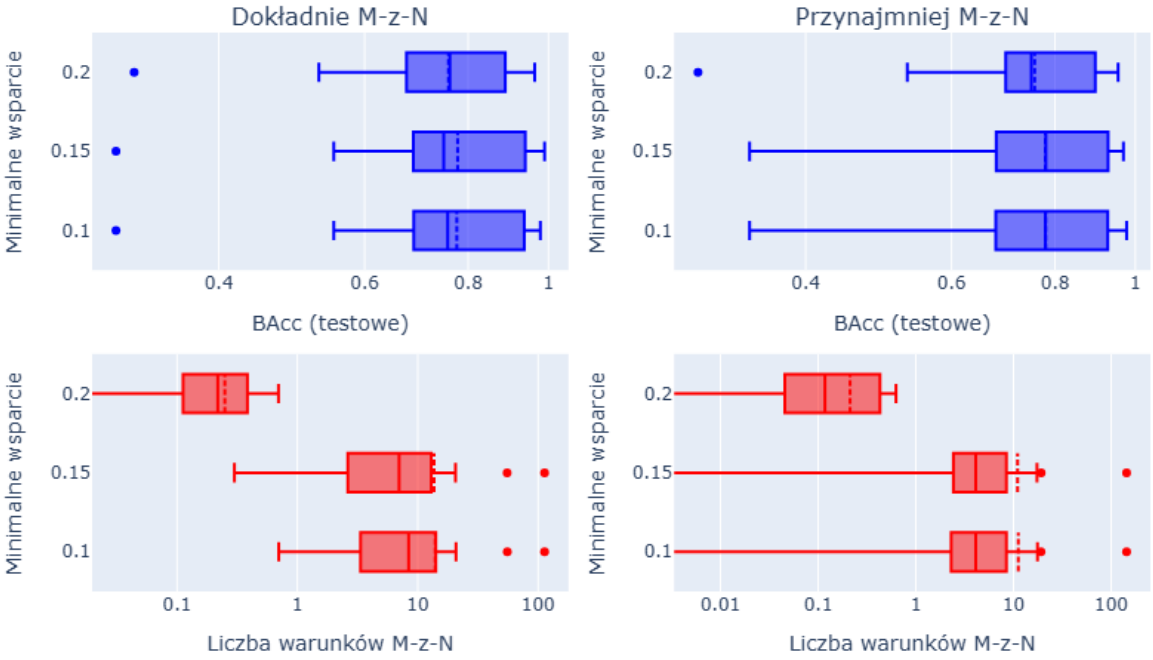
Tabela 7.7: Wartości parametrów metody DeepRules zastosowane podczas eksperymentów.

Nazwa parametru	Wartość
min_rule_covered	2
max_uncovered_fraction	0.02
max_layers_count	20
max_component_length	10
dnf_quality_measure	Correlation
dnf_pruning_measure	C2
dnf_select_best_candidate_measure	C2
cnf_quality_measure	C2
cnf_pruning_measure	C2
cnf_select_best_candidate_measure	Correlation

niem tego wyboru przeprowadzono eksperymenty dla trzech różnych wartości progu: 0.1, 0.15 oraz 0.2. Zostały porównane dwa wskaźniki: BAcc na zbiorze testowym oraz sumaryczna liczba warunków M-z-N w całym zbiorze reguł. Zagregowane wyniki zostały przedstawione na wykresie pudełkowym 7.2 oraz w tabelach: 7.8 i 7.9. Lewa oraz prawa granica pudełka to dolny i górny kwartyl Q1 i Q3. Linia ciągła wewnątrz pudełka reprezentuje medianę wartości, a przerywaną średnią. Tak zwane „wąsy" pudełka określają górne i dolne ogrodzenie (z ang. upper/lower fence), oznaczane dalej w tekście skrótowo jako uf i lf. Nie definiują one globalnego maksimum i minimum, lecz obliczane są zgodnie ze wzorem 7.2, gdzie IQR oznacza odstęp pomiędzy górnym i dolnym kwartylem. Wszystkie punkty na wykresie wykraczające poza zakres wyznaczany przez „wąsy" uznawane są za wartości odstające. Taki sposób prezentacji pozwala w czytelny sposób zawrzeć wiele istotnych informacji na jednym rysunku. Dla wykresu 7.2, dla zachowania lepszej czytelności i podkreślenia różnic została zastosowana skala logarytmiczna.

$$\begin{aligned} min &= Q1 - 1.5(IQR) \\ max &= Q3 + 1.5(IQR) \end{aligned} \tag{7.2}$$

Na ich podstawie zauważyć można, że liczba warunków M-z-N pojawiająca się w uzyskanych zbiorach reguł jest podobna dla progów minimalnego wsparcia 0.1 i 0.15. Maleje ona jednak znacząco dla wartości 0.2. Wynika to z faktu, że w oczywisty sposób prawdopodobieństwo znalezienia zbioru częstego jest tym mniejsze, im wyższa jest wartość progu minimalnego wsparcia. Różnica ta jest widoczna zarówno dla wariantu „przynajmniej M-z-N" jak i „dokładnie M-z-N". Wpływ wartości minimalnego wspar-



Rysunek 7.2: Wykres pudełkowy przedstawiający wpływ progu minimalnego wsparcia zbiorów częstych na dokładność predykcji oraz liczbę warunków M-z-N.

Tabela 7.8: Statystyki wskaźnika BAcc dla różnych poziomów minimalnego wsparcia.

Wariant	Min. wsparcie	lw	up	q1	q3	średnia	mediana
„dokładnie M-z-N”	0.100	0.337	1.280	0.691	0.927	0.775	0.756
	0.150	0.331	1.287	0.689	0.928	0.777	0.748
	0.200	0.382	1.187	0.684	0.885	0.757	0.760
„przynajmniej M-z-N”	0.100	0.318	1.282	0.679	0.920	0.779	0.779
	0.150	0.318	1.282	0.680	0.921	0.779	0.780
	0.200	0.410	1.177	0.698	0.890	0.756	0.749

Tabela 7.9: Statystyki sumarycznej liczby warunków M-z-N w całym zbiorze reguł, dla różnych poziomów minimalnego wsparcia (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

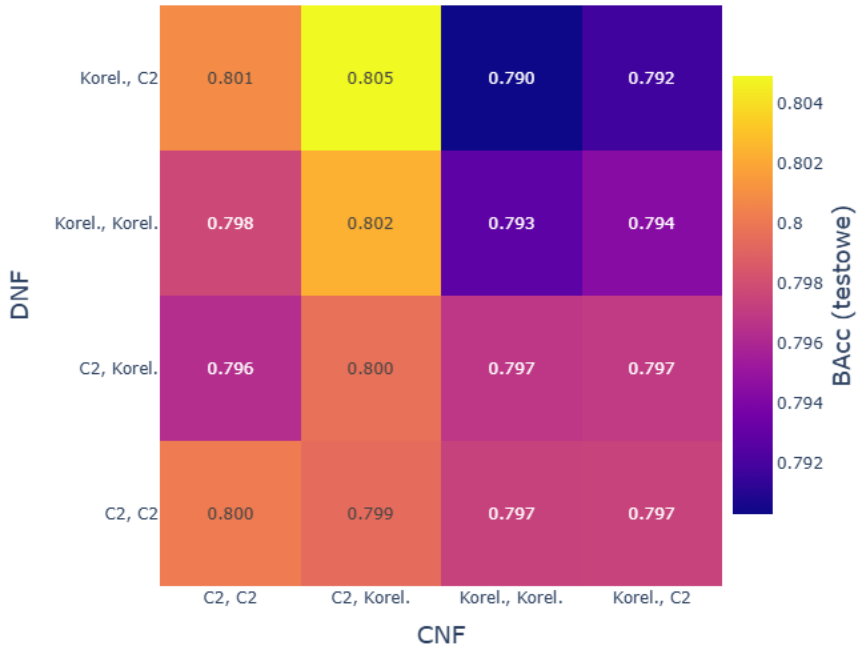
Wariant	Min. wsparcie	lf	uf	q1	q3	średnia	mediana
„dokładnie M-z-N”	0.100	0.0	29.275	3.400	13.750	13.963	8.400
	0.15	0.0	27.975	2.850	12.900	13.660	7.000
	0.20	0.0	0.764	0.118	0.376	0.250	0.218
„przynajmniej M-z-N”	0.100	0.0	17.150	2.400	8.300	11.193	4.100
	0.15	0.0	16.925	2.550	8.300	11.015	4.100
	0.20	-0.0	0.968	0.048	0.416	0.212	0.117

cia na jakość predykcji jest mniej zauważalny. Dla obydwu wariantów wszystkie trzy testowane progi prowadziły do zbliżonych rezultatów. Wraz ze spadkiem minimalnego wsparcia można zauważyć również wzrost wariancji wyników klasyfikacji.

Jak zostało wcześniej wspomniane, celem opracowanych metod jest nie tylko osiąganie wysokich zdolności predykcyjnych, ale również eksploracja danych i odkrywanie wiedzy. Warunki M-z-N opisują dość specyficzny rodzaj relacji w danych i z tego powodu spodziewać się można, że nie będą one pojawiać się w nich zbyt często. Wyniki uzyskane dla progów 0.15 i 0.1 wskazują jednak na dużą liczbę wystąpień (średnio powyżej 10 warunków w regule). Rezultaty dla minimalnego wsparcia równego 0.2 są tutaj bardziej zgodne z intuicją. Średnia i mediana mniejsze od 1 sugerują, że dla niektórych zbiorów relacja taka nie występowała w danych w ogóle. W subiektywnej ocenie autora tej pracy, reguły zawierające dużą ilość warunków M-z-N mogą być przez to trudniejsze w zrozumieniu i interpretacji. Z powyższych powodów do dalszych eksperymentów użyto wartość minimalnego wsparcia równa 0.2.

W przypadku algorytmu DeepRules posiada on aż siedem parametrów do definiowania miar używanych w trakcie indukcji i głosowania. Umożliwia to stosowanie różnych miar podczas tworzenia reguł DNF i CNF. Miara głosowania jest określana dla całego wygenerowanego zbioru reguł. Aby zapewnić spójność wyników, zastosowano miarę C2, tak samo jak w przypadku pozostałych metod. Zarówno dla reguł DNF, jak i CNF w fazie przycinania zdecydowano się na użycie miary C2. Pozostałe 4 parametry kontrolują miary używane do wzrostu reguły, jak i wyboru najlepszego warunku dla obydwu ich postaci (patrz rozdział 6 sekcja 3). Aby dobrać ich wartości, przeprowadzono eksperymenty dla problemu klasyfikacji na wcześniej wspominanych 30 zbiorach danych, dla wszystkich możliwych kombinacji miar Correlation i C2. Oznaczało to przeprowadzenie 16 eksperymentów dla 30 zbiorów danych, których wyniki porównano pod kątem średniej wartości BAcc na zbiorze testowym. Wyniki te zostały przedstawione w formie mapy ciepła na rysunku 7.3. Oś x przedstawia wszystkie kombinacje miar używanych do indukcji reguł CNF, a oś y dla reguł DNF. Pierwsza miara z każdej z par była używana do wzrostu reguł (parametry: *dnf_quality_measure* i *cnf_quality_measure*) a druga do wyboru najlepszego warunku (parametry: *dnf_quality_measure* i *cnf_quality_measure*). Wybrano konfigurację, która uzyskała najwyższą średnią wartość BAcc na zbiorze testowym (patrz tabela 7.7).

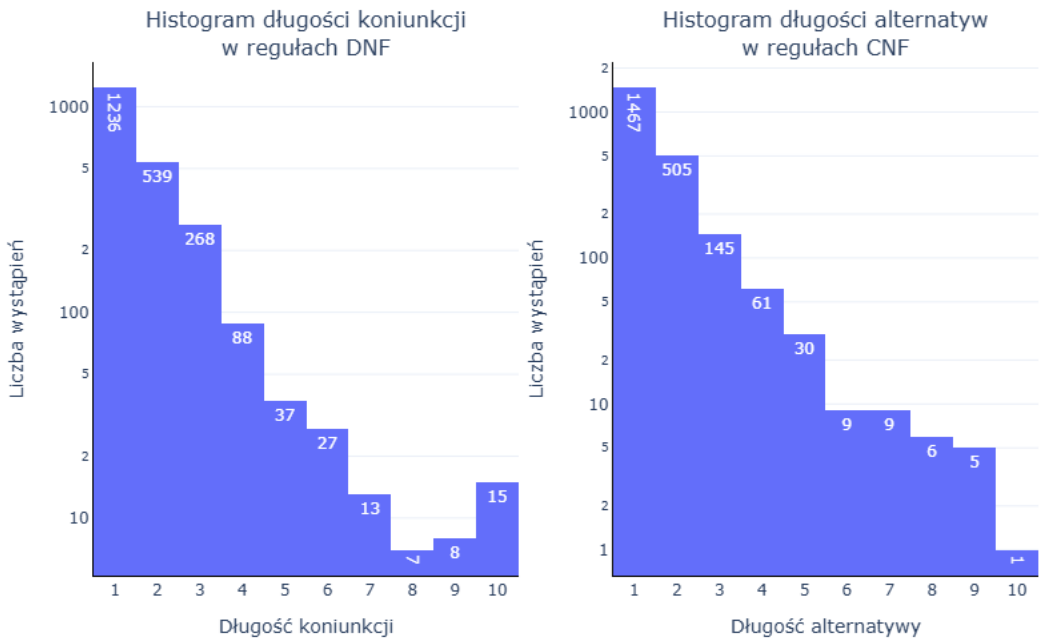
Wybór wartości dla parametrów kontrolujących maksymalną długość oraz liczbę składników w regułach, dla metody DeepRules, został dokonany na początku intu-



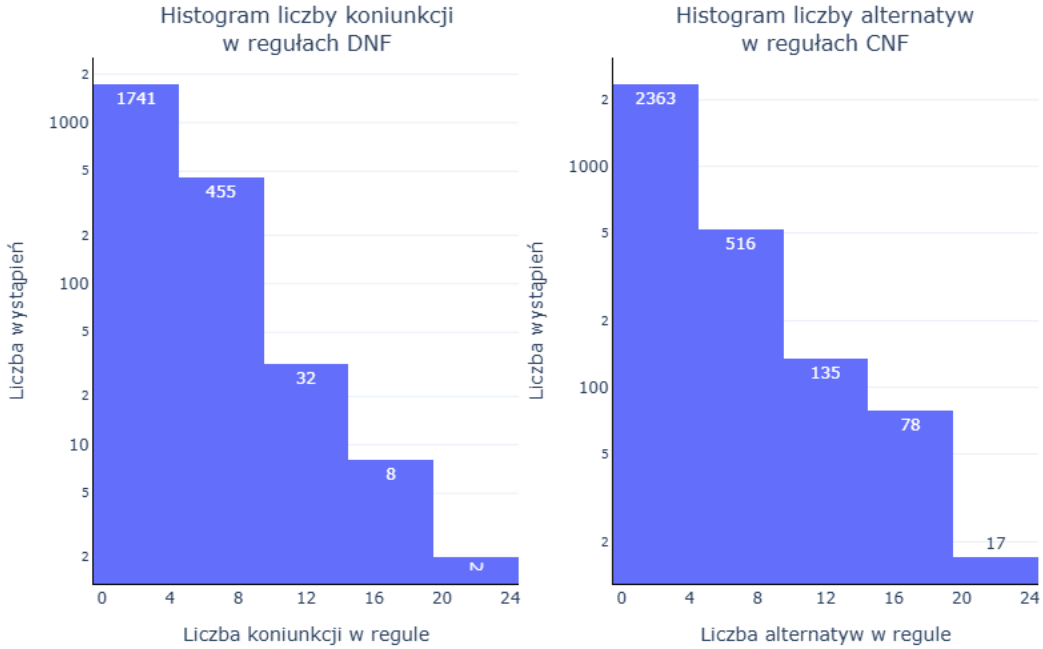
Rysunek 7.3: Mapa ciepła obrazująca wpływ użytych miar jakości reguł na wartość wskaźnika BAcc dla metody DeepRules.

icyjnie. Dla pierwszego z nich została wybrana wartość 10, a dla drugiego 22. W subiektywnej ocenie autora reguły o dłuższych składnikach lub posiadające większą ich liczbę mogłyby być trudne w interpretacji i zrozumieniu. Po przeprowadzeniu badań wykonano dodatkowo analizę rozkładów liczby składników w regułach oraz ich długości. Histogramy dla tych rozkładów zostały przedstawione na rysunkach 7.4 oraz 7.5. Dla obydwu z nich, dla lepszej czytelności, została zastosowana skala logarytmiczna.

Na podstawie rysunku 7.4 można zauważyć, że składniki o maksymalnej długości 10 warunków pojawiają się w regułach stosunkowo rzadko. Liczbę wystąpień należy tutaj rozumieć jako liczbę wszystkich składników o danej długości, we wszystkich regułach foldów walidacji krzyżowej dla wszystkich badanych zbiorów danych. Widać tutaj, jak liczba składników maleje wraz ze wzrostem ich długości. Na tej podstawie można stwierdzić, że dobrana wartość parametru kontrolującego ich maksymalną długość jest rozsądna. Można domniemać tutaj, że po jej zwiększeniu składniki o większej liczbie warunków byłyby potencjalnie jeszcze bardziej rzadkie, nie powodując znaczącej różnicy w wynikach.



Rysunek 7.4: Histogram rozkładu długości składników w regułach DNF i CNF dla metody DeepRules.



Rysunek 7.5: Histogram rozkładu liczby składników w regułach DNF i CNF dla metody DeepRules.

W przypadku rozkładu liczby składników w regule można zauważyć podobny trend. Reguły o większej ich liczbie są rzadziej obecne w zbiorach. Sugeruje to, że dalsze zwiększenie wartości parametru nie wpłynęłoby w znaczący sposób na wyniki przeprowadzonych badań.

4 Badania zdolności predykcyjnych proponowanych metod

Dokładne wyniki badanych metod dla wszystkich użytych zbiorów z osobna widoczne są w tabelach 7.10, 7.11 i 7.12. Te same wyniki zostały również zagregowane i przedstawione dla problemów klasyfikacji, regresji i przeżycia w tabeli 7.13. Zawiera ona średnie wartości wskaźników osiągane przez każdą z nich oraz odchylenie standardowe ich wartości. Prezentowane w niej wyniki są średnią ze wszystkich zbiorów, a więc średnią uśrednionych wyników z wszystkich foldów walidacji krzyżowej. Kolumna BAcc dotyczy wyników uzyskanych na zbiorach klasyfikacyjnych, RRMSE na zbiorach regresyjnych, a IBS przeżyciowych. W przypadku dwóch ostatnich należy pamiętać, że mamy do czynienia z metrykami błędów, gdzie niższa wartość jest bardziej pożądaną niż wyższa. W przypadku BAcc (kolumna pierwsza), wyższa wartość oznacza lepszy wynik. Dwa pierwsze wiersze tabeli oddzielone od reszty dotyczą wyników dla metod referencyjnych, a więc algorytmu RuleKit oraz drzew decyzyjnych. Dają one pogląd na to, jak proponowane w pracy metody wypadają na tle istniejących już rozwiązań.

Na podstawie tabeli 7.13 można zauważyć, że wyniki dla problemu klasyfikacji są zbliżone do siebie dla wszystkich testowanych metod. Najwyższą średnią wartość uzyskała tutaj metoda DeepRules (0.805), najniższą z kolei osiągnęły drzewa decyzyjne (0.761). Dla algorytmów proponowanych w pracy średnio najgorszym okazał się MofNRules. W ramach późniejszej analizy ocenione zostanie, czy zaobserwowane różnice były istotne statystycznie. Odchylenie standardowe pozostawało na zbliżonym poziomie dla wszystkich metod regułowych (około 1,16). Jest ono jednocześnie na nieco wyższym poziomie w przypadku drzew decyzyjnych (0.197).

Dla problemu regresji można zauważyć widocznie niższą wartość RRMSE dla metod RuleKit (0.776) i DeepRules (0.762), z pewną przewagą tego drugiego. Wyniki metody MofNRules są zbliżone dla obydwu wariantów (0.832 i 0.836). Algorytm ComplexConditions poradził sobie dla tego problemu najgorzej, ze średnią wartością

Zbiór danych	RuleKit	Drzewo decyzyjne	ComplexConditions	MofNRules („dokładnie M-z-N”)	MofNRules („przynajmniej M-z-N”)	DeepRules
anneal	0.961	0.974	0.947	0.946	0.946	0.949
auto-mpg	0.67	0.696	0.622	0.663	0.699	0.625
autos	0.737	0.816	0.75	0.736	0.744	0.823
balance-scale	0.608	0.558	0.896	0.962	0.933	0.942
bupa-liver-disorders	0.694	0.633	0.7	0.709	0.697	0.675
car	0.666	0.96	0.675	0.585	0.552	0.925
cleveland	0.287	0.281	0.273	0.316	0.297	0.277
cylinder-bands	0.785	0.693	0.751	0.76	0.789	0.729
echocardiogram	0.862	0.824	0.831	0.849	0.837	0.837
ecoli	0.747	0.405	0.744	0.723	0.728	0.701
flag	0.568	0.551	0.57	0.528	0.531	0.577
glass	0.633	0.567	0.606	0.536	0.567	0.653
hayes-roth	0.709	0.811	0.837	0.749	0.727	0.832
heart-c	0.815	0.776	0.795	0.796	0.792	0.809
heart-statlog	0.816	0.793	0.801	0.783	0.794	0.803
hepatitis	0.685	0.681	0.614	0.606	0.61	0.677
iris	0.927	0.907	0.953	0.96	0.953	0.933
lymphography	0.812	0.729	0.798	0.784	0.749	0.803
monk-1	0.967	1.0	1.0	1.0	1.0	1.0
monk-2	0.71	0.84	0.819	0.751	0.752	0.796
monk-3	0.913	0.933	0.991	0.974	0.974	1.0
mushroom	0.967	1.0	0.99	0.778	0.778	0.999
nursery	0.732	0.995	0.655	0.746	0.746	0.841
sonar	0.748	0.497	0.678	0.664	0.674	0.703
soybean	0.952	0.945	0.934	0.882	0.88	0.954
tic-tac-toe	0.975	0.939	0.943	0.889	0.9	0.854
titanic	0.711	0.682	0.711	0.703	0.703	0.711
vote	0.935	0.933	0.956	0.96	0.942	0.956
wine	0.945	0.473	0.932	0.927	0.932	0.902
zoo	0.86	0.943	0.888	0.908	0.908	0.86

Tabela 7.10: Wyniki BAcc na zbiorze testowym dla wszystkich badanych metod (najwyższa wartość dla każdego zbioru została podkreślona).

Zbiór danych	RuleKit	Drzewo decyzyjne	ComplexConditions	MofNRules („dokładnie M-z-N")	MofNRules („przynajmniej M-z-N")	DeepRules
auto-mpg	0.44	0.391	0.498	0.45	0.447	0.445
auto-price	0.703	0.771	0.806	0.757	0.787	0.739
auto93	0.936	0.891	0.889	0.877	0.864	0.872
bodyfat	0.378	0.601	0.345	0.356	0.362	0.307
bolts	0.81	0.894	0.724	0.602	0.698	0.704
breasttumor	0.55	0.655	0.561	0.56	0.561	0.549
cholesterol	0.761	0.761	0.77	0.773	0.77	0.761
cloud	0.728	1.14	0.924	0.984	0.997	0.784
concrete	0.68	0.678	0.605	0.618	0.635	0.641
cpu	1.355	1.053	1.629	1.524	1.488	1.464
dee	0.418	0.663	0.385	0.412	0.408	0.364
diabetes	0.844	0.905	0.828	0.816	0.816	0.762
echomonths	0.572	0.614	0.742	0.687	0.684	0.608
ele-1	0.656	0.812	0.72	0.684	0.684	0.645
elusage	0.698	0.705	0.764	0.8	0.8	0.707
fishcatch	0.628	0.76	0.69	0.725	0.711	0.602
fruitfly	0.972	0.981	0.904	0.94	0.931	0.865
gascons	0.657	0.951	0.679	0.755	0.76	0.646
kidney	0.793	1.143	0.845	0.8	0.824	0.827
lowbwt	0.444	0.442	0.483	0.479	0.471	0.45
machine	1.144	0.986	1.237	1.264	1.366	1.204
mbagrade	0.652	0.709	0.676	0.636	0.636	0.626
ozone	0.636	0.686	0.704	0.657	0.614	0.573
pharynx	0.735	0.802	0.924	0.711	0.728	0.707
pollution	0.768	0.765	0.687	0.769	0.756	0.68
pwlinear	0.429	0.322	0.544	0.581	0.578	0.443
pyrim	0.94	0.922	0.949	0.995	0.997	0.908
servo	0.812	0.741	1.424	1.375	1.321	0.863
strike	1.817	1.803	2.021	1.966	1.951	1.752
veteran	1.332	1.307	1.406	1.406	1.428	1.348

Tabela 7.11: Wyniki RRMSE na zbiorze testowym dla wszystkich badanych metod (najniższa wartość dla każdego zbioru została podkreślona).

Zbiór danych	RuleKit	Drzewo decyzyjne	ComplexConditions	MofNRules („dokładnie M-z-N”)	MofNRules („przynajmniej M-z-N”)	DeepRules
GBSG2	0.182	0.276	0.182	0.184	0.186	0.173
Melanoma	0.188	0.218	0.2	0.203	0.212	0.19
bmt-ch	0.215	0.186	0.249	0.262	0.249	0.197
cancer	0.15	0.276	0.171	0.158	0.162	0.17
follic	0.202	0.269	0.197	0.203	0.196	0.185
hd	0.214	0.121	0.214	0.213	0.213	0.188
lung	0.148	0.208	0.151	0.155	0.154	0.149
mgus	0.179	0.288	0.179	0.182	0.175	0.164
pbc	0.157	0.221	0.153	0.156	0.152	0.17
std	0.213	0.293	0.218	0.22	0.222	0.197
uis	0.17	0.26	0.178	0.155	0.149	0.153
whas1	0.203	0.191	0.211	0.218	0.221	0.198
whas500	0.202	0.183	0.2	0.207	0.199	0.195
zinc	0.094	0.232	0.1	0.097	0.097	0.033

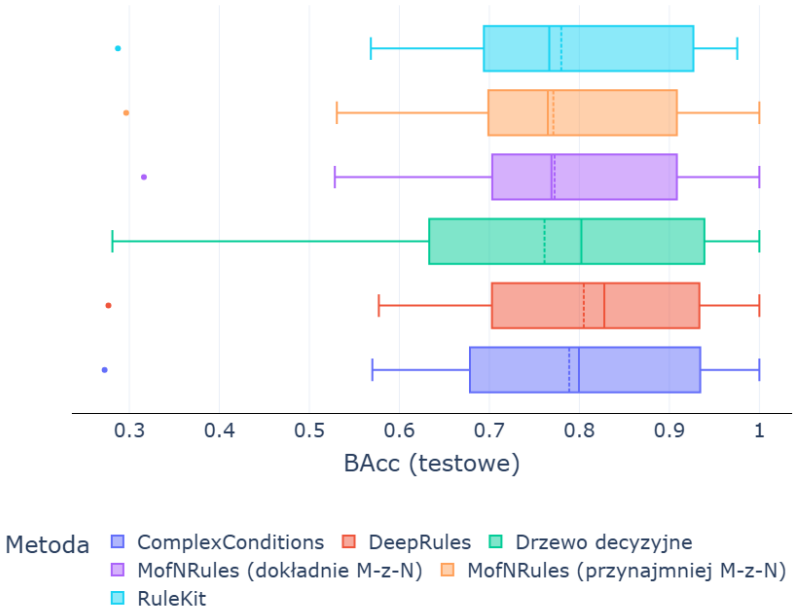
Tabela 7.12: Wyniki IBS na zbiorze testowym dla wszystkich badanych metod (najniższa wartość dla każdego zbioru została podkreślona).

RRMSE równą 0.845. Odchylenie standardowe wyników było najniższe dla drzew decyzyjnych (0.285). Dla metod RuleKit i DeepRules wyniosło kolejno 0.31 i 0.319. Dla algorytmów ComplexConditions i MofNRules było ono stosunkowo podobne (około 0.36).

Większe zróżnicowanie wyników widoczne jest dla problemu analizy przeżycia. Metodą osiągającą najlepszy wynik jest DeepRules ze średnim IBS równym 0.169 i stosunkowo niskim odchyleniem 0.043. Poradził on sobie tutaj znacząco lepiej niż

Tabela 7.13: Wyniki predykcyjne badanych metod.

Metoda	BAcc	RRMSE	IBS
Drzewo decyzyjne	0.761 ± 0.197	0.829 ± 0.285	0.212 ± 0.069
RuleKit	0.78 ± 0.153	0.776 ± 0.31	0.18 ± 0.034
ComplexConditions	0.789 ± 0.163	0.845 ± 0.369	0.186 ± 0.036
MofNRules („dokładnie M-z-N”)	0.772 ± 0.158	0.832 ± 0.358	0.187 ± 0.04
MofNRules („przynajmniej M-z-N”)	0.771 ± 0.158	0.836 ± 0.356	0.185 ± 0.039
DeepRules	0.805 ± 0.156	0.762 ± 0.319	0.169 ± 0.043



Rysunek 7.6: Wykres pudełkowy prezentujący zdolności predykcyjne (BAcc) badanych algorytmów, dla problemu klasyfikacji.

metody referencyjne, w tym RuleKit (IBS = 0.18) oraz drzewa decyzyjne (IBS = 0.212), które były w tym wypadku najgorsze. Pozostałe algorytmy ComplexRules i MofNRules ponownie osiągnęły raczej zbliżone wyniki. Dla pierwszej z nich IBS wyniósł 0.186. MofNRules dla wariantu „dokładnie M-z-N" osiągnął IBS równy 0.187, a dla „przynajmniej M-z-N" równy 0.185.

W dalszej części rozdziału przeprowadzona zostanie dokładniejsza analiza wyników osobno dla każdego z badanych problemów.

4.1 Problem klasyfikacji

Aby lepiej ocenić różnice w zdolnościach klasyfikacyjnych badanych metod, uzyskane wyniki zostały przedstawione w formie wykresu pudełkowego na rysunku 7.6. Na osi x zostały zobrazowane wartości metryki BAcc. Te same dane zostały także zaprezentowane w formie tabeli 7.15, pozwalając na odczytanie dokładnych wartości statystyk.

Na wykresie widoczne są wartości odstające dla wszystkich metod z wyjątkiem algorytmu RuleKit. Jako wartość odstająca traktowana była tutaj wartość różniąca się od wartości średniej o więcej niż $1.5(Q3-Q1)$, gdzie $Q1$ i $Q3$ to kolejno kwartyl dolny

Metoda	BAcc (treningowe)	BAcc (testowe)
Drzewo decyzyjne	1.0	0.281
RuleKit	0.909	0.287
ComplexConditions	0.93	0.273
MofNRules („dokładnie M-z-N")	0.91	0.316
MofNRules („przynajmniej M-z-N")	0.968	0.297
DeepRules	0.735	0.277

Tabela 7.14: Wyniki uzyskane dla zbioru cleveland

i górny. Wartość taką stanowił tutaj zawsze wynik dla jednego i tego samego zbioru danych o nazwie cleveland. Dokładne wyniki dla tego zbioru zaprezentowane zostały w tabeli 7.14. Dla wszystkich metod można tutaj zaobserwować duże pogorszenie pomiędzy BAcc na zbiorze treningowym i testowym. Jest to przykład popularnie napotykanego w uczeniu maszynowym problemu przeuczenia, gdzie model nie generalizuje wystarczająco dobrze dla nowo napotkanych danych. Spójność wyników osiąganych przez poszczególne metody nie wskazuje jednak na błąd w implementacji którejkolwiek z nich.

Wracając do wykresu 7.6 oraz tabeli 7.15, można zauważyć, że metoda DeepRules wyróżnia się na tle innych. Osiągnęła ona najwyższą wartość średniej (0.805) oraz mediany (0.828). Dla wszystkich analizowanych metod widoczne jest duże zróżnicowanie wyników (długość tzw. „wąsów" pudełek). Największą ich rozpiętością cechują się drzewa decyzyjne. Wszystkie jego kwartyle są najniższe, mimo stosunkowo wysokiej wartości mediany (wyższej niż dla wszystkich metod z wyjątkiem DeepRules). Jeżeli chodzi o metody zaproponowane w pracy, największą rozpiętość między kwartylem górnym i dolnym wykazuje algorytm MofNRules w obydwu wariantach „przynajmniej M-z-N" oraz „dokładnie M-z-N". Warto zaznaczyć, że wyniki dla obydwu wariantów są do siebie bardzo zbliżone. Metoda MofNRules charakteryzuje się najwyższymi wartościami dolnego ogrodzenia oraz najniższymi górnego. Ich mediany również są niższe niż dla pozostałych metod proponowanych w niniejszej pracy. Sugeruje to, że rzadziej osiągają one wysokie zdolności klasyfikacyjne niż pozostałe metody. Algorytm ComplexConditions osiąga najniższą wartość dolnego ogrodzenia, przy jednocześnie wysokiej wartości średniej i mediany, co wskazuje na zróżnicowanie wyników.

Po przeprowadzeniu powyższej analizy sprawdzono, czy pomiędzy badanymi metodami występują statystycznie istotne różnice. W tym celu wykorzystano test Friedmana na poziomie istotności $\alpha = 0.05$. Jego statystyka wyniosła 8.471, a p-wartość

Tabela 7.15: Statystyki wskaźnika BAcc dla problemu klasyfikacji (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

Metoda	lf	uf	q1	q3	średnia	mediana
Drzewo decyzyjne	0.207	1.376	0.645	0.937	0.761	0.802
RuleKit	0.359	1.261	0.698	0.923	0.780	0.767
ComplexConditions	0.309	1.309	0.684	0.934	0.789	0.800
MofNRules („dokładnie M-z-N")	0.407	1.201	0.705	0.903	0.772	0.769
MofNRules („przynajmniej M-z-N")	0.391	1.216	0.700	0.906	0.771	0.765
DeepRules	0.365	1.271	0.705	0.931	0.805	0.828

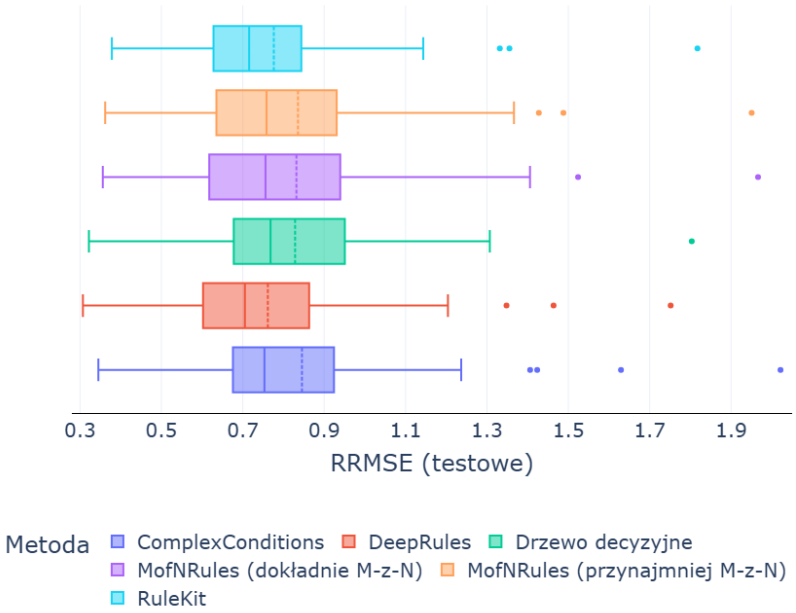
równa była 0.132. Jest ona wyższa niż założony poziom istotności, co potwierdza hipotezę zerową testu o braku istotnych statystycznie różnic pomiędzy wynikiem BAcc na zbiorze testowym dla badanych metod. Oznacza to, że wszystkie testowane metody osiągały podobne zdolności klasyfikacyjne. Wprowadzenie złożonych form warunków oraz wyrażeń logicznych do przesłanek reguł nie przyniosło tutaj znaczącej różnicy w wynikach.

4.2 Problem regresji

Zdolności predykcyjne algorytmów dla problemu regresji, ocenione zostały z użyciem metryki RRMSE. Jest on modyfikacją błędu średniokwadratowego, gdzie wynik jest dodatkowo dzielony przez sumę rzeczywistych wartości atrybutu decyzyjnego (patrz równanie 6.17). Dzięki takiemu zabiegowi uzyskiwane wartości nie zależą od przedziału rzeczywistych wartości etykiet. Pozwala to na porównanie błędów uzyskanych dla różnych zbiorów danych.

Rysunek 7.7 przedstawia wykres pudełkowy dla wartości wskaźnika RRMSE. Dokładne wartości prezentowanych wskaźników zostały dodatkowo zamieszczone w tabeli 7.16.

Na ich podstawie można zauważyć, że algorytm RuleKit osiągał najbardziej spójne wyniki, o czym świadczy najmniejsza odległość pomiędzy jego kwartylami ($IQR = 0.206$). Największy rozstęp pomiędzy kwartylami osiągnęła metoda MofNRules ($IQR = 0.302$ dla wariantu „dokładnie M-z-N" i $IQR = 0.279$ dla „przynajmniej M-z-N"). Cechują ją również najwyższe wartości górnego ogrodzenia sugerujące dużą zmienność uzyskiwanych wyników. Podobnie jak wcześniej, algorytm DeepRules wyróżnia się na tle innych proponowanych w pracy metod. Osiąga on najniższą spośród wszystkich ocenianych metod wartość dolnego ogrodzenia oraz średnią i medianę.



Rysunek 7.7: Wykres pudełkowy prezentujący zdolności predykcyjne (RRMSE) badanych algorytmów, dla problemu regresji.

Metoda ComplexConditions osiągnęła najwyższy średni RRMSE (0.845) spośród wszystkich algorytmów, co może wskazywać na nieco gorszą ogólną wydajność. Mediana (0.753) jest jednak zbliżona do metody MofNRules oraz nieznacznie niższa niż dla drzewa decyzyjnego. Podobnie jak w przypadku klasyfikacji, nie widać znaczącej różnicy w jakości predykcji pomiędzy dwoma wariantami metody MofNRules.

Na wykresie 7.7 widoczne są liczne wartości odstające wychodzące poza górne odgrodzenie. Dla drzew decyzyjnych widoczna jest jedna taka wartość odpowiadająca wynikom dla zbioru strike. Stanowił on outlier dla wszystkich badanych metod. Dla

Tabela 7.16: Statystyki wskaźnika RRMSE dla problemu regresji (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

Metoda	lf	uf	q1	q3	średnia	mediana
Drzewo decyzyjne	0.284	1.339	0.680	0.944	0.829	0.768
RuleKit	0.321	1.145	0.630	0.836	0.776	0.716
ComplexConditions	0.313	1.283	0.677	0.919	0.845	0.753
MofNRules („dokładnie M-z-N”)	0.170	1.376	0.622	0.924	0.832	0.756
MofNRules („przynajmniej M-z-N”)	0.217	1.332	0.635	0.914	0.836	0.758
DeepRules	0.229	1.229	0.604	0.854	0.762	0.705

algorytmów RuleKit, DeepRules oraz MofNRules w wariancie „dokładnie M-z-N", wartości odstające dotyczyły zbiorów: cpu, strike i veteran. Dla ComplexConditions był to dodatkowo zbiór servo. W przypadku metody MofNRules dla wariantu „przynajmniej M-z-N" były to cpu, machine, strike i veteran. Uśrednione wyniki dla wszystkich metod, dla zbiorów gdzie wystąpiły wartości odstające, znajdują się w tabeli 7.17. Dla każdego zbioru danych są prezentowane wyniki RRMSE zarówno na części treningowej, jak i testowej. Metody, dla których wynik stanowił wartość odstającą, zostały dodatkowo podkreślone.

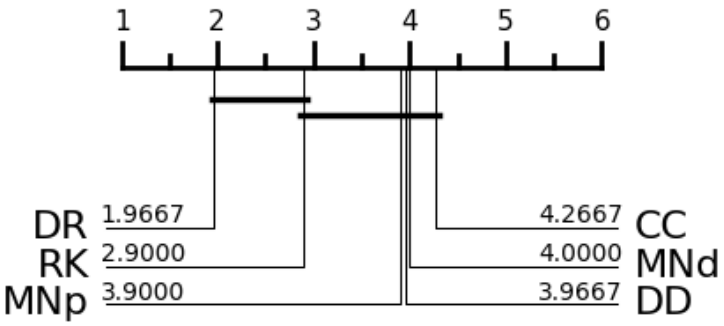
Patrząc na wyniki pierwszego zbioru w tabeli 7.17 o nazwie cpu, można zaobserwować, że wszystkie metody regułowe osiągały RRMSE treningowej powyżej 1 oraz testowe powyżej 1.3. Dla drzew decyzyjnych było to 0.0 dla zbioru treningowego i 1.053 dla testowego. Stanowi to zbliżoną wartość do pozostałych metod. Pokazuje to, że dla tego konkretnego zbioru, wartość RRMSE była stosunkowo wysoka dla wszystkich badanych algorytmów, również tych referencyjnych. Dla zbioru machine wynik na części testowej dla metody MofNRules wariantu „przynajmniej M-z-N" jest o około 0.1 gorszy od wyników drugiego wariantu oraz metody ComplexConditions. Dla wszystkich algorytmów regułowych (poza drzewami) RRMSE testowe jest stosunkowo wysokie (ponad 1.1). Podobna sytuacja występuje dla zbioru veteran, gdzie wszystkie algorytmy cechują się wysokim błędem predykcji (powyżej 1.3). W tym wypadku również dla drzewa jest on wysoki i wynosi 1.307. Podobnie dla zbioru strike można odnotować duże wartości RRMSE dla zbioru testowego (powyżej 1.7). Dla metod ComplexConditions oraz obu wariantów MofNRules jest ona bliższa 2. Metoda RuleKit, będąca metodą referencyjną, osiągnęła tutaj wynik 1.817. Ciekawa sytuacja występuje w przypadku zbioru servo, gdzie zaobserwować można dużą rozbieżność w wynikach metod. Najlepsze okazały się dla niego drzewa decyzyjne (RRMSE testowe równe 0.741). Nieco gorzej wypadły tutaj RuleKit i DeepRules (kolejno RRMSE testowe równe 0.812 i 0.863). Największą wartość błędu osiągały jednak metody ComplexConditions (0.1424) i MofNRules (1.375 dla wariantu „dokładnie M-z-N" i 1.321 dla „przynajmniej M-z-N"). Należy tutaj pamiętać, że wyniki dla tych metod są ze sobą powiązane. Metoda MofNRules wykorzystuje wewnętrznie algorytm ComplexConditions i wygenerowane przez niego reguły (patrz rozdział 6 sekcja 2). W skrajnym przypadku, dla zbioru nie uda się wygenerować warunków M-z-N spełniających określone przez metodę kryteria lub też nie pojawią się one w wynikowym zbiorze reguł. Wtedy może on być identyczny, jak ten początkowy uzyskany przez algorytm ComplexConditions. Tłumaczyć to może identyczne lub

Tabela 7.17: Wyniki RRMSE dla zbiorów, dla których wystąpiły wartości odstające.

Zbiór danych	Metoda	RRMSE (treningowe)	RRMSE (testowe)
cpu	ComplexConditions	1.354	1.629
	DeepRules	1.181	1.464
	Drzewo decyzyjne	0.000	1.053
	MofNRules („dokładnie M-z-N")	1.424	1.524
	MofNRules („przynajmniej M-z-N")	1.270	1.488
	RuleKit	1.030	1.355
machine	ComplexConditions	1.171	1.237
	DeepRules	0.807	1.204
	Drzewo decyzyjne	0.096	0.986
	MofNRules („dokładnie M-z-N")	1.217	1.264
	MofNRules („przynajmniej M-z-N")	1.368	1.366
	RuleKit	0.861	1.144
servo	ComplexConditions	1.342	1.424
	Drzewo decyzyjne	0.000	0.741
	DeepRules	0.753	0.863
	MofNRules („dokładnie M-z-N")	1.252	1.375
	MofNRules („przynajmniej M-z-N")	1.250	1.321
	RuleKit	0.846	0.812
strike	ComplexConditions	1.887	2.021
	DeepRules	1.720	1.752
	Drzewo decyzyjne	0.000	1.803
	MofNRules („dokładnie M-z-N")	1.906	1.966
	MofNRules („przynajmniej M-z-N")	1.891	1.951
	RuleKit	1.658	1.817
veteran	ComplexConditions	1.299	1.406
	DeepRules	1.134	1.348
	Drzewo decyzyjne	0.000	1.307
	MofNRules („dokładnie M-z-N")	1.275	1.406
	MofNRules („przynajmniej M-z-N")	1.199	1.428
	RuleKit	1.127	1.332

zbliżone wyniki dla obydwu metod. Przypadek tego zbioru został dokładniej zbadany, a uzyskane reguły nie wskazywały na wystąpienie błędu w implementacji algorytmów.

W dalszej części analizy wyników zostało zbadane, czy różnice pomiędzy ocenianymi metodami są istotne statystycznie. Ponownie przeprowadzono dla nich test Friedmana na poziomie istotności $\alpha = 0.05$. Wartość statystyki wyniosła 33.786, a p-wartość była równa 2.626E-6. Na tej podstawie można odrzucić hipotezę zerową, potwierdzając tym samym istnienie statystycznie istotnej różnicy pomiędzy wynikami poszczególnych algorytmów. Aby określić, pomiędzy którymi z nich jest ona obecna, przeprowadzono w pierwszej kolejności test post hoc Nemenyi dla $\alpha = 0.05$. Jego wyniki zostały przedstawione w postaci CD diagramu (z ang. critical difference



Rysunek 7.8: CD diagram dla wskaźnika RRMSE wyrysowany na podstawie wyników testu Nemenyi dla poziomu istotności $\alpha = 0.05$, wartość CD wyniosła 1.424.

diagram) reprezentującego różnice pomiędzy grupami pomiarów. Oś x reprezentuje rangi poszczególnych metod, gdzie najniższa ranga oznacza najlepszą z nich. Metody połączone ze sobą poziomą linią nie wykazują statystycznie istotnych różnic pomiędzy sobą. CD diagram dla wskaźnika RRMSE na zbiorze testowym widoczny jest na rysunku 7.8. Na jego podstawie zauważyć można brak istotnych różnic pomiędzy metodą DeepRules a RuleKit. Potwierdza on jednak istnienie istotnej różnicy pomiędzy algorytmem DeepRules a pozostałymi badanymi metodami. One same nie różnią się jednak między sobą w znaczący sposób.

Na koniec przeprowadzony został dodatkowo test Wilcoxona dla par obserwacji dla $\alpha = 0.05$, wraz z korektą p-wartości (patrz sekcja 1). Sprawdza on dodatkowo istotność różnic pomiędzy wszystkimi parami algorytmów. Jego wyniki zaprezentowane zostały w tabeli 7.18. Aby zachować czytelność tabeli, wprowadzono tutaj skrótowe oznaczenia poszczególnych algorytmów: CC - ComplexConditions, DR - DeepRules, DD - drzewo decyzyjne, RK - RuleKit, MNd - MofNRules wariant „dokładnie M-z-N" i MNp - MofNRules wariant „przynajmniej M-z-N". Wszystkie przecięcia wierszy i kolumn, dla których wartość jest mniejsza od $\alpha = 0.05$, oznaczają wystąpienie istotnej statystycznie różnicy w RRMSE testowym dla pary metod. Dla lepszej czytelności wartości takie zostały podkreślone.

Wyniki testów Wilcoxona potwierdzają wcześniejsze wnioski. Metoda DeepRules wykazuje istotne różnice względem wszystkich pozostałych, z wyjątkiem RuleKit. Dla RuleKit różnice występują jedynie względem wszystkich metod poza DeepRules

Tabela 7.18: P-wartości testów Wilcoxona porównujących wartość wskaźnika RRMSE ($\alpha = 0.05$, korekta FDR).

	CC	DR	DD	MNd	MNp	RK
CC	-	0.00	0.94	0.45	0.77	0.03
DR	0.00	-	0.04	0.00	0.00	0.27
DD	0.94	0.04	-	0.77	0.77	0.15
MNd	0.45	0.00	0.77	-	0.94	0.04
MNp	0.77	0.00	0.77	0.94	-	0.04
RK	0.03	0.27	0.15	0.04	0.04	-

Tabela 7.19: Statystyki wskaźnika IBS dla problemu analizy przeżycia (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

Metoda	lf	uf	q1	q3	średnia	mediana
Drzewo decyzyjne	0.078	0.392	0.196	0.274	0.230	0.226
RuleKit	0.096	0.267	0.160	0.203	0.180	0.185
ComplexConditions	0.120	0.261	0.173	0.208	0.186	0.189
MofNRules („dokładnie M-z-N")	0.074	0.293	0.156	0.211	0.187	0.193
MofNRules („przynajmniej M-z-N")	0.071	0.297	0.156	0.213	0.185	0.191
DeepRules	0.123	0.236	0.165	0.194	0.169	0.179

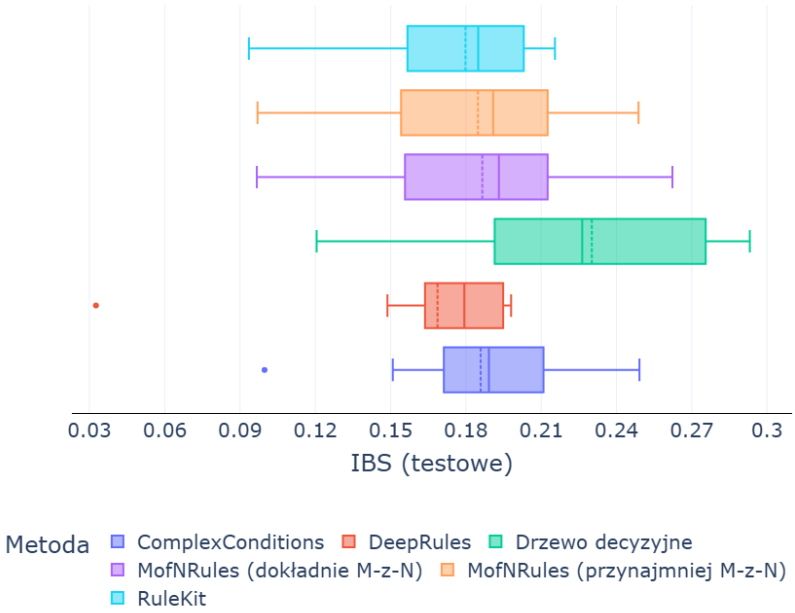
oraz drzewami decyzyjnymi. Pozostałe proponowane metody nie różnią się względem siebie w sposób istotny statystycznie.

4.3 Problem analizy przeżycia

Jak już wspomniano, do oceny zdolności predykcyjnych badanych metod, dla analizy przeżycia, użyty został całkowy wskaźnik Briera, w skrócie IBS (patrz równanie 6.18). Wartość IBS równa 0 odpowiada modelowi idealnemu, 0.25 oznacza z kolei model losowy, tj. taki, zwracający dla każdego czasu wartość prawdopodobieństwa równą 0.5. Podobnie jak w przypadku RRMSE, niższa wartość oznacza w tym wypadku lepszy wynik.

Analizę rozpoczęto, podobnie jak dla pozostałych dwóch typów problemów, od wyrysowania wykresu pudełkowego prezentującego wyniki dla wskaźnika IBS (patrz rysunek 7.9. Dokładne wartości prezentowanych na nim statystyk znajdują się w tabeli 7.19.

Na podstawie wykresu 7.9 oraz tabeli 7.19, można zaobserwować, że drzewa decyzyjne wyraźnie odstają od pozostałych metod. Osiągnęły one najwyższą medianę IBS (0.226) oraz średnią (0.23). Również odstęp pomiędzy kwartylem górnym i dolnym



Rysunek 7.9: Wykres pudełkowy prezentujący zdolności predykcyjne (IBS) badanych algorytmów, dla problemu analizy przeżycia.

jest w ich przypadku największy (IQR = 0.078), co wskazuje na dużą zmienność wyników. RuleKit, będący drugą metodą referencyjną, wypada znacznie lepiej od drzew, z medianą równą 0.185 oraz średnią 0.18. Jego IQR równe jest 0.043. Pokazuje to, że metody regułowe mogą stanowić użyteczną alternatywę dla drzew decyzyjnych w przypadku problemu analizy przeżycia, osiągając znacznie lepsze i bardziej stabilne wyniki.

Z metod proponowanych przez autora, najlepszym, pod względem wartości IBS na zbiorze testowym, okazał się algorytm DeepRules. Posiada on najniższą wartość średnią (0.169) i medianę (0.179) spośród wszystkich badanych metod. Cechuje go również najmniejszy kwartyl górny (Q3 = 0.194) oraz rozstęp międzykwartylowy (IQR = 0.029). Świadczy to o wysokiej spójności wyników. Potwierdza to również różnica pomiędzy dolnym i górnym ogrodzeniem wynosząca 0.113.

Obydwa warianty MofNRules osiągają bardzo podobne wyniki z medianami około 0.19 oraz średnimi około 0.186. Wyniki te są zbliżone do algorytmu ComplexConditions (średnia 0.186 i mediana równa 0.189), oraz nieco gorsze od referencyjnej metody RuleKit. MofNRules wykazują największy rozstęp między kwartylami ze wszystkich proponowanych metod (odpowiednio 0.055 dla wariantu „dokładnie M-z-N” oraz

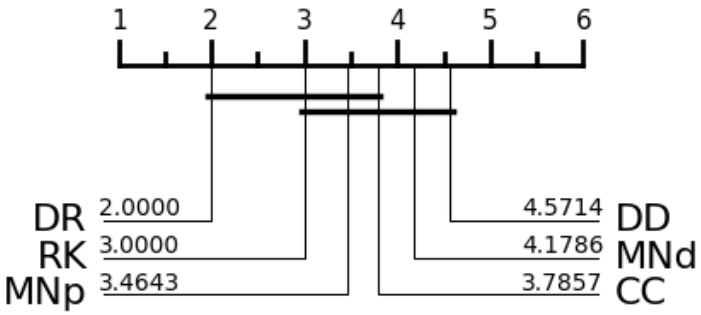
Tabela 7.20: Wyniki dla zbioru zinc.

Metoda	IBS (treningowy)	IBS (testowy)
Drzewo decyzyjne	0.034	0.232
RuleKit	0.084	0.096
ComplexConditions	0.086	0.100
MofNRules („dokładnie M-z-N")	0.091	0.097
MofNRules („przynajmniej M-z-N")	0.091	0.097
DeepRules	0.014	0.033

0.057 dla wariantu „przynajmniej M-z-N"). Dla ComplexConditions rozstęp pomiędzy kwartylem górnym i dolnym jest równy 0.035 i bliski temu osiąganemu przez metodę RuleKit.

Na wykresie 7.9 widoczne są pojedyncze wartości odstające dla metod DeepRules i ComplexConditions. Odpowiadają one wynikom dla zbioru zinc, gdzie obydwie metody osiągnęły bardzo niskie wartości IBS. Wciąż jednak IBS dla metody DeepRules (0.033) był tutaj niemal trzykrotnie niższy niż dla metody RuleKit (0.1). Wyniki dla wszystkich metod, na zbiorze zinc zostały zaprezentowane w tabeli 7.20. Dla algorytmu ComplexConditions wartość IBS dla zbioru testowego jest zbliżona do wyniku osiąganego przez RuleKit. W przypadku DeepRules wartość ta była znacząco niższa zarówno na zbiorze treningowym, jak i testowym niż dla pozostałych badanych metod.

Aby ocenić istotność zidentyfikowanych różnic, przeprowadzono test Friedmana dla $\alpha = 0.05$. Jego statystyka wyniosła 16.8 a p-wartość równa była 0.0049. Wynik ten świadczy o istnieniu statystycznie istotnej różnicy pomiędzy przynajmniej jedną z par badanych algorytmów. Aby zidentyfikować różniące się między sobą metody, przeprowadzono test Nemenyi dla $\alpha = 0.05$. Jego wyniki zostały przedstawione na CD diagramie widocznym na rysunku 7.10. Potwierdza on wnioski i widoczną przewagę algorytmu DeepRules nad pozostałymi metodami. Jest on w istotny statystycznie sposób lepszy od drzew decyzyjnych oraz metody MofNRules w wariancie „dokładnie M-z-N". W przypadku porównywania go z pozostałymi algorytmami test nie wykazał istotnych różnic. Porównując jednak rangi na wykresie, można zauważyć, że różnica pomiędzy rangą DeepRules a kolejnego po nim algorytmu RuleKit jest o wiele większa niż różnice pomiędzy kolejnymi występującymi na nim punktami. Sugeruje to, że uznawany za dość konserwatywny test Nemenyi mógł nie wykazać wszystkich różnic faktycznie występujących w danych.



Rysunek 7.10: CD diagram dla wskaźnika IBS wyrysowany na podstawie wyniku testu Nemenyi dla poziomu istotności $\alpha = 0.05$, wartość CD wyniosła 2.085.

Tabela 7.21: P-wartości testów Wilcoxona porównujących wartość wskaźnika IBS ($\alpha = 0.05$, korekta FDR).

	CC	DR	DD	MNd	MNp	RK
CC	–	0.04	0.08	0.30	0.86	0.09
DR	0.04	–	0.04	0.04	0.04	0.08
DD	0.08	0.04	–	0.09	0.08	0.06
MNd	0.30	0.04	0.09	–	0.37	0.06
MNp	0.86	0.04	0.08	0.37	–	0.31
RK	0.09	0.08	0.06	0.06	0.31	–

Aby to zweryfikować, wykonano dodatkowo test Wilcoxona dla par obserwacji, również dla $\alpha = 0.05$. Jego wyniki widoczne są w tabeli 7.21. Pokreślone zostały te komórki, dla których p-wartość wskazuje wystąpienie istotnej różnicy. Test ten wykazał różnice pomiędzy metodą DeepRules a wszystkimi pozostałymi z wyjątkiem algorytmu RuleKit. Pozostałe metody proponowane w pracy wykazują istotne różnice tylko względem DeepRules.

5 Badania złożoności zbiorów reguł uzyskiwanych z użyciem proponowanych metod

Wprowadzone w rozdziale 5 pojęcie głębokiego dyskretnego uczenia opierało się na hipotezie, że głębokie, bardziej złożone modele mogą reprezentować niektóre funkcje w wydajniejszy sposób niż modele płytkie i proste [13]. Aby zweryfikować, czy sytuacja taka wystąpiła również w przypadku zaproponowanych przez autora

metod, konieczna jest dalsza analiza wyników. Poprzednia sekcja skupiona była wyłącznie na ocenie zdolności predykcyjnych badanych algorytmów. W dalszej części autor podejmie próbę oceny, jak wprowadzenie złożonych form warunków i przesłanek wpłynęło na złożoność uzyskiwanych zbiorów reguł. W tym celu poddano analizie trzy wskaźniki: liczbę reguł, średnią liczbę warunków w przesłance oraz sumaryczną liczbę warunków we wszystkich regułach.

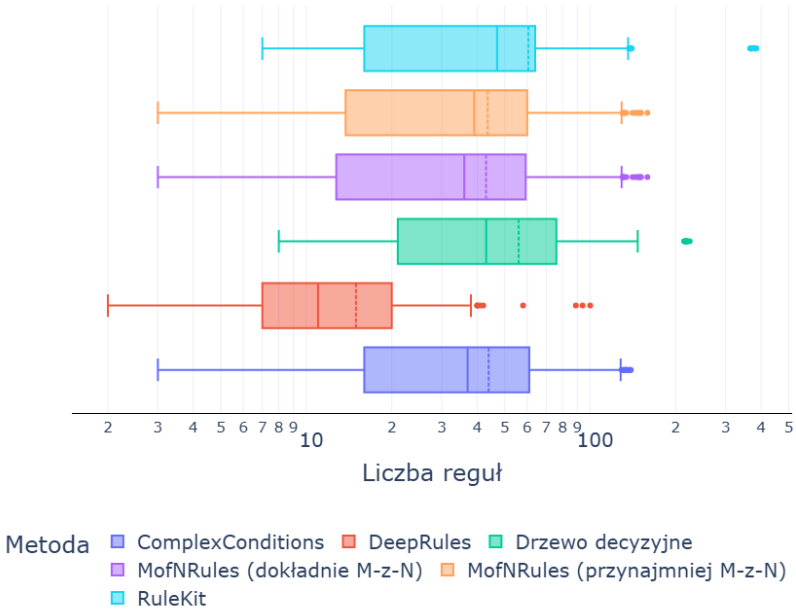
W niniejszej sekcji zastosowano odmienne podejście niż we wcześniejszych porównaniach. Badane są tutaj wszystkie wyniki, dla wszystkich dziesięciu iteracji (foldów) walidacji krzyżowej, a nie, jak miało to miejsce wcześniej, wartość uśredniona. Każdy przebieg walidacji krzyżowej odpowiada bowiem pojedynczemu zbiorowi reguł. W tym konkretnym przypadku uśrednienie wartości dla wszystkich z nich doprowadziłoby do utraty cennych informacji. Z takich danych nie sposób byłoby dowiedzieć się na przykład, jaka była maksymalna i minimalna liczba reguł uzyskana dla danego zbioru danych. Zabieg taki w oczywisty sposób zwiększa rozmiar badanych prób. Nie jest ona jak wcześniej równa liczbie zbiorów danych, lecz jest 10-krotnie większa. W przypadku metryki określającej liczbę warunków w regule odpowiada ona średniej wartości dla wszystkich reguł pojedynczego zbioru (foldu). W tekście bywa ona czasem nazywana długością reguły, dla krótszego zapisu. Możliwe byłoby tutaj również zbadanie liczby warunków w poszczególnych regułach z osobna. Takie podejście ma jednak pewne wady. Zbiory reguł wygenerowane poszczególnymi metodami nie muszą i zwykle nie zawierają tej samej liczby reguł. Sprawia to, że liczebność prób uzyskanych dla poszczególnych algorytmów nie byłaby równa. Utrudniłoby to lub uniemożliwiło zastosowanie wcześniej wspominanych testów statystycznych Friedmana i Wilcoxona.

Warto w tym miejscu wyjaśnić również, w jaki sposób obliczana jest sumaryczna liczba warunków. Uzyskiwana jest poprzez zsumowanie liczby warunków znajdujących się w przesłankach wszystkich reguł danego zbioru. Oznacza to, że ten sam warunek pojawiający się w kilku regułach policzony zostanie kilkukrotnie.

5.1 Problem klasyfikacji

Analiza liczby reguł

Analizę rozpoczęto od zbadania rozkładu liczby reguł w zbiorach wygenerowanych poszczególnymi metodami. Został on zwizualizowany na wykresie pudełkowym widocznym na rysunku 7.11. Dokładne wartości prezentowane na nim statystyk znajdują się w tabeli 7.22.

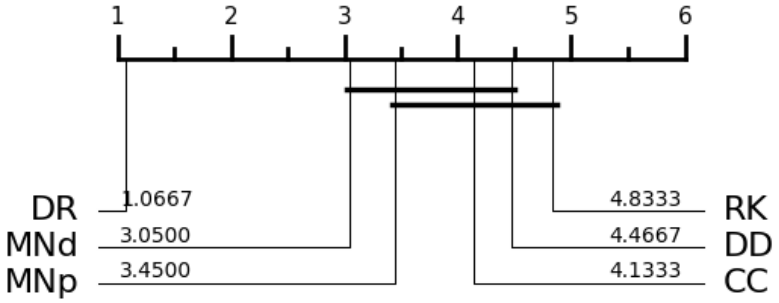


Rysunek 7.11: Wykres pudełkowy prezentujący liczbę reguł w wygenerowanych zbiorach, dla problemu klasyfikacji.

Na podstawie wyników ocenić można, że wszystkie z proponowanych przez autora metod (ComplexConditions, MofNRules oraz DeepRules) wykazują tendencję do generowania mniejszej liczby reguł niż metody referencyjne (RuleKit i drzewa decyzyjne). Widać to dobrze po wartościach średnich oraz medianach. Dla RuleKit średnia liczba reguł w zbiorach wynosi 60.6 a mediana blisko 47. Dla drzew decyzyjnych średnia i mediana były nieco niższe (kolejno 55.971 oraz 43). Dla ComplexConditions średnia wyniosła 43.9 a mediana 37. Metoda MofNRules w obu wariantach osiągnęła bardzo podobne wyniki. Średnia, w jej przypadku, zbliżona jest do 43, a mediana wynosi około 36. Najlepszy okazał się tutaj po raz kolejny algorytm DeepRules. Średnia liczba reguł wyniosła dla niego blisko 15 a mediana równa była 11. Jest to spora poprawa względem metody RuleKit. Wynika ona najprawdopodobniej ze specyfiki reguł generowanych przez algorytm. Wszystkie pozostałe metody generowały zbiory reguł z postaci DNF, gdzie przesłanka każdej z nich była koniunkcją warunków. Pewnym wyjątkiem były warunki wewnętrznych alternatyw generowane przez metodę ComplexConditions. Ich wyznaczanie podlegało jednak licznym ograniczeniom, co sprawiało, że pojawiały się one w regułach rzadko. W przypadku metody DeepRules każda z reguł może mieć postać DNF lub CNF, co umożliwia modelowanie bardziej

Tabela 7.22: Statystyki liczby reguł dla problemu klasyfikacji (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

Metoda	lf	uf	q1	q3	średnia	mediana
Drzewo decyzyjne	8.0	147.0	21.0	76.0	55.971	43.0
RuleKit	7.0	136.0	16.0	64.0	60.575	47.0
ComplexConditions	3.0	128.0	16.0	61.0	43.875	37.0
MofNRules („dokładnie M-z-N”)	3.0	129.0	13.0	59.0	42.956	36.0
MofNRules („przynajmniej M-z-N”)	3.0	129.0	14.0	60.0	43.538	39.0
DeepRules	2.0	38.0	7.0	20.0	14.96	11.0



Rysunek 7.12: CD diagram dla liczby reguł klasyfikacyjnych ($\alpha = 0.05$; CD = 1.424).

złożonych wyrażeń logicznych. W specyficznym przypadku pojedyncza tego typu reguła może opisać tę samą zależność co cały zbiór reguł w formacie DNF. Taki przypadek w oczywisty sposób jednak przekładałby się na długość takiej reguły. Między innymi z tego powodu konieczne jest przeprowadzanie dalszych analiz, wskaźników takich jak średnia liczba warunków w regule.

Test Friedmana przeprowadzony dla poziomu istotności $\alpha = 0.05$ wykazał istnienie statystycznie istotnych różnic (statystyka wyniosła 82.332, a p-wartość równa była 2.728e-16). W celu zidentyfikowania, dla których par metod zaistniały takowe różnice, wyrysowany został CD diagram na podstawie wyniku testu Nemenyi (patrz rysunek 7.12). Test wykonano dla tego samego poziomu istotności $\alpha = 0.05$.

Różnica w liczbie reguł generowanych przez metodę DeepRules względem pozostałych okazała się istotna statystycznie. Pozostałe metody proponowane w ramach pracy uzyskiwały wyniki lepsze od metod referencyjnych. Mimo to jedynie dla al-

Tabela 7.23: P-wartości testów Wilcoxona porównujących liczbę wygenerowanych reguł ($\alpha = 0.05$, korekta FDR).

	CC	DR	DD	MNd	MNp	RK
CC	–	0.0	0.0	0.0	0.0	0.0
DR	0.0	–	0.0	0.0	0.0	0.0
DD	0.0	0.0	–	0.0	0.0	0.10
MNd	0.0	0.0	0.0	–	0.03	0.0
MNp	0.0	0.0	0.0	0.03	–	0.0
RK	0.0	0.0	0.10	0.0	0.0	–

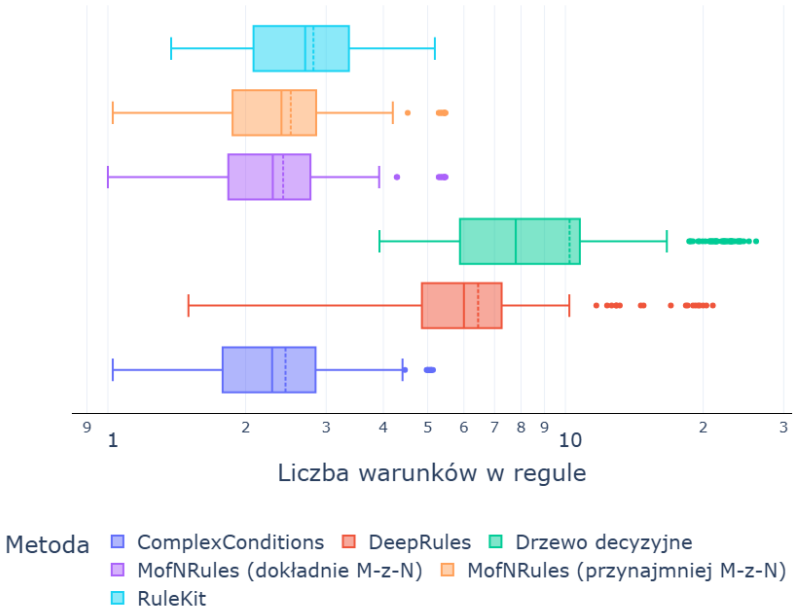
gorytmu MofNRules w wariancie „dokładnie M-z-N" różnica ta okazała się istotna statystycznie.

Na koniec przeprowadzony został test Wilcoxona z korektą FDR dla $\alpha = 0.05$. Jako test mniej konserwatywny niż wspomniany wcześniej test Nemenyi, wykazał on istnienie istotnych statystycznie różnic pomiędzy wszystkimi parami metod, z wyjątkiem RuleKit i drzew decyzyjnych.

Analiza średniej liczby warunków w regułach

Analizę rozpoczęto od wyrysowania wykresu pudełkowego dla średniej liczby warunków w regułach, dla każdej z badanych metod (patrz rysunek 7.13). Dokładne wartości przedstawionych na nim statystyk widoczne są w tabeli 7.24.

Na pierwszy rzut oka zauważyć można, że dwie spośród metod znacząco odstają od pozostałych, wykazując znacznie wyższą średnią liczbę warunków w regułach. Są to drzewa decyzyjne (średnia równa 10.2) oraz DeepRules (średnia równa 6.4). W przypadku drzew może to wynikać z ustawień parametrów. Liczba warunków w regule jest tutaj równa liczbie węzłów w ścieżce od korzenia do liścia, która zależy od maksymalnej głębokości drzewa. Ta z kolei jest często ograniczana poprzez odpowiednie ustawienie hiperparametrów algorytmu. W przypadku opisywanych badań maksymalna głębokość drzewa była jednak nieograniczona z góry. Miało to symulować zachowanie metody RuleKit, dla której domyślnie długość reguły nie ma ograniczeń (patrz sekcja 3). We wcześniejszej podsekcji zostało wykazane, że metoda DeepRules cechuje się silną tendencją do generowania mniejszej liczby reguł. Jak widać na podstawie wyników, są one jednak na ogół dłuższe i zawierają więcej warunków w przesłance. Może to wynikać ze specyficznej budowy ich przesłanek, które mogą przyjmować postać zarówno DNF, jak i CNF. Obydwa warianty metody MofNRules wykazały bardzo zbliżone wyniki, ze średnią około 2.5 i medianą około 2.3.



Rysunek 7.13: Wykres pudełkowy prezentujący średnią liczbę warunków w regułach dla problemu klasyfikacji.

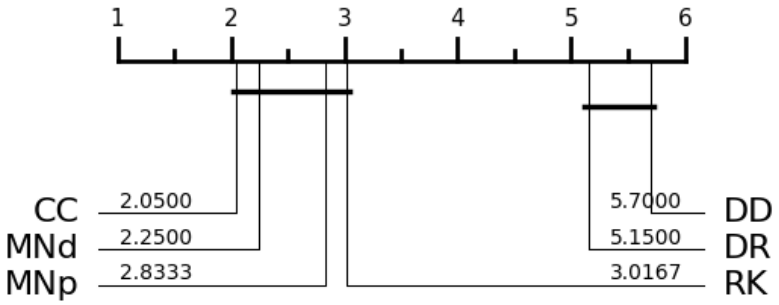
Zarówno metoda ComplexConditions (średnia równa 2.4 i mediana wynosząca 2.3) jak i MofNRules wykazały niższą średnią liczbę warunków w regule od referencyjnego algorytmu RuleKit (średnia równa 2.8 i mediana równa 2.7). Również wartość dolnego i górnego kwartyla jest w ich przypadku niższa. W połączeniu z obserwacjami i wnioskami z poprzedniej podsekcji można uznać, że metody ComplexConditions i MofNRules, dla problemu klasyfikacji, prowadzą na ogół do mniej złożonych zbiorów reguł, zarówno pod kątem liczby reguł, jak i ich długości rozumianej jako średnia liczba warunków w przesłankach.

Przeprowadzony w pierwszym kroku test Friedmana, dla poziomu istotności $\alpha = 0.05$, potwierdził istnienie statystycznie istotnych różnic między algorytmami (wartość statystyki testu wyniosła 104.74, a p-wartość równa była 5.286e-21). Jako kolejny wykonany został test Nemenyi, którego wyniki zwizualizowane zostały na CD diagramie (patrz rysunek 7.14). Został on obliczony dla tej samej wartości α co wcześniejszy test Friedmana, CD wyniosło 1.424.

Na podstawie CD diagramu z rysunku 7.14 stwierdzić można, że test Nemenyi nie wykazał statystycznie istotnej różnicy w wynikach pomiędzy algorytmanu RuleKit, ComplexConditions i MofNRules. Najlepszą spośród proponowanych metod

Tabela 7.24: Statystyki średniej liczby warunków w regule reguł dla problemu klasyfikacji (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

Metoda	lf	uf	q1	q3	średnia	mediana
Drzewo decyzyjne	3.923	16.66	5.882	10.725	10.211	7.789
RuleKit	1.375	5.184	2.083	3.364	2.814	2.699
ComplexConditions	1.025	4.405	1.783	2.842	2.445	2.286
MofNRules („dokładnie M-z-N”)	1.0	3.917	1.833	2.771	2.416	2.293
MofNRules („przynajmniej M-z-N”)	1.025	4.196	1.875	2.849	2.51	2.395
DeepRules	1.5	10.2	4.857	7.25	6.445	6.0



Rysunek 7.14: CD diagram dla średniej liczby warunków w regule dla problemu klasyfikacji ($\alpha = 0.05$; CD = 1.424).

Tabela 7.25: P-wartości testów Wilcoxona porównujących średnią liczbę warunków w regule ($\alpha = 0.05$, korekta FDR).

	CC	DR	DD	MNd	MNp	RK
CC	–	0.0	0.0	0.21	0.0	0.0
DR	0.0	–	0.0	0.0	0.0	0.0
DD	0.0	0.0	–	0.0	0.0	0.0
MNd	0.21	0.0	0.0	–	0.0	0.0
MNp	0.0	0.0	0.0	0.0	–	0.0
RK	0.0	0.0	0.0	0.0	0.0	–

pod kątem długości reguły okazał się ComplexConditions, a najgorszą DeepRules. Ten ostatni nie wykazał statystycznie istotnej różnicy względem drzew decyzyjnych. Obydwie te metody okazały się jednak prowadzić do istotnie dłuższych reguł niż pozostałe badane algorytmy. Warto jednak zwrócić tutaj uwagę na realne wartości statystyk zawarte w tabeli 7.24. Dla metody DeepRules mediana i górny kwartyl wyniosły kolejno 6 oraz 7.3. Oznacza to, że połowa reguł generowanych tą metodą posiada przesłanki zawierające 6 warunków lub mniej, a trzy czwarte z nich 7 warunków lub mniej. Wartości te, biorąc pod uwagę specyficzną budowę przesłanek budowanych przez ten algorytm, nie wydają się być przesadnie wysokie. Algorytm DeepRules w swych założeniach indukuje dwa rodzaje reguł. Pierwsze posiadają przesłankę w postaci DNF, a drugie CNF, gdzie literałami są warunki logiczne. Już dla koniunkcji trzech trójelementowych alternatyw dla postaci CNF liczba warunków w regule przekroczy górny kwartyl, wynosząc 9, pomimo faktu, że wyrażenie logiczne tego typu samo w sobie nie jest przesadnie skomplikowane. Co więcej, niska liczba warunków w regułach dla danego zbioru danych może świadczyć o tym, że algorytm nie odkrył w nim bardziej złożonych zależności lub że takie zależności po prostu tam nie występują. W skrajnym przypadku reguły DNF i CNF mogą składać się ze składników zawierających jeden warunek. Przesłanka reguł ma wtedy postać koniunkcji warunków (dla reguł CNF), lub alternatywy (dla reguł DNF).

Test Wilcoxona z korektą FDR, przeprowadzany dla $\alpha = 0.05$, wykazał istnienie statystycznie istotnych różnic pomiędzy większością testowanych algorytmów. Wyjątkiem okazała się jedynie para metod MofNRules (wariant „dokładnie M-z-N") i ComplexConditions.

Analiza sumarycznej liczby warunków w zbiorze reguł

Wcześniejsze analizy pozwoliły ocenić zaproponowane przez autora metody pod względem dwóch metryk: liczby generowanych reguł oraz długości ich przesłanek. Dla DeepRules wartość pierwszej z nich okazała się istotnie niższa, a drugiej wyższa niż dla pozostałych algorytmów. Innymi słowy, algorytm ten generuje na ogół zbiory o większej liczbie dłuższych reguł. Należy jednak pamiętać, że reguły indukowane metodą DeepRules mogą posiadać przesłanki w postaci DNF, która w teorii opisać może tę samą zależność co cały zbiór reguł tworzonych przez pozostałe metody. Rozważmy poniższy przykładowy zbiór reguł opisujący pojedynczą klasę.

$$\textbf{JEŻELI } a \ < 3 \wedge b \ = \{0\} \textbf{ TO } klasa = \{\mathbf{1}\}$$
$$\textbf{JEŻELI } a \ > 5 \wedge b \ = \{0\} \textbf{ TO } klasa = \{\mathbf{1}\}$$
$$\textbf{JEŻELI } c \ = \{1\} \wedge b = \{1\} \textbf{ TO } klasa = \{\mathbf{1}\}$$

Posiada on trzy reguły, z których każda zawiera dwa warunki w przesłance, co daje sumarycznie sześć warunków w całym zbiorze. Rozpatrzmy teraz zbiór reguł zawierający jedną regułę zaprezentowaną poniżej.

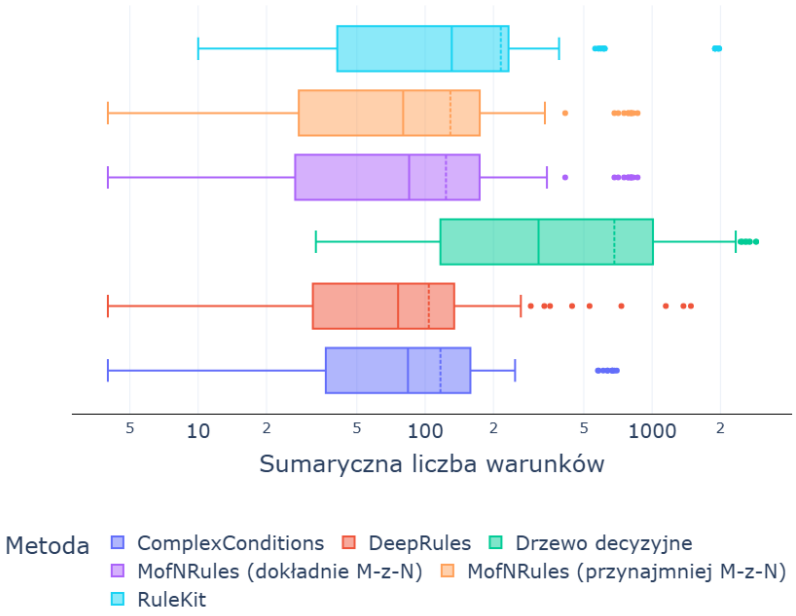
$$\textbf{JEŻELI } (a < 3 \wedge b \ = \{0\}) \vee (a > 5 \wedge b \ = \{0\}) \vee (c = \{1\} \wedge b \ = \{1\}) \textbf{ TO } \text{klasa} = \{\mathbf{1}\}$$

Mógłby on potencjalnie zostać wygenerowany przez algorytm DeepRules. Opisuje on tę samą relację w danych co wcześniejszy zbiór z trzema regułami, obserwujemy tutaj jednak spadek liczby reguł. Sama reguła jest jednak znacznie dłuższa pod kątem liczby warunków w przesłance, ponieważ jej przesłanka zawiera w sobie przesłanki wszystkich trzech reguł pierwszego zbioru. Oceniając w tym miejscu sumaryczną liczbę wszystkich warunków pojawiających się w całym zbiorze reguł, zobaczyć można, że jest ona identyczna dla obydwu zbiorów.

przesłanki

Analizę rozpoczęto od obliczenia statystyk dla sumarycznej liczby warunków w zbiorze dla wszystkich metod (patrz tabela 7.26). Zostały one, tak jak poprzednio, zwizualizowane z użyciem wykresu pudełkowego widocznego na rysunku 7.15.

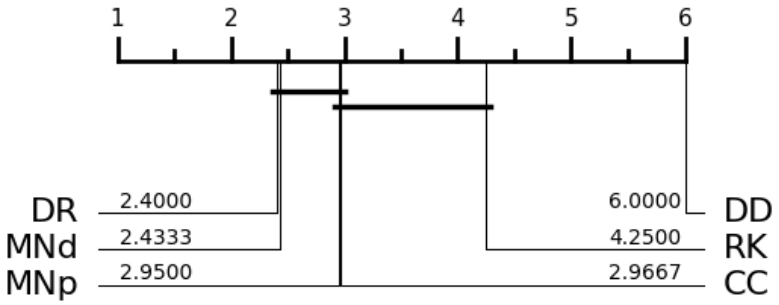
Na podstawie wykresu 7.15, stwierdzić można, że algorytm DeepRules osiąga najniższą wartość średnią (103.76) oraz medianę (76) ze wszystkich ocenianych algorytmów. Dolny i górny kwartyl (kolejno 32 i 134) są w jego przypadku również widocznie niższe niż w przypadku RuleKit (kolejno 41 i 232). Z proponowanych metod jedynie MofNRules osiągnęło niższą wartość dolnego kwartylu (27 dla wariantu



Rysunek 7.15: Wykres pudełkowy prezentujący sumaryczną liczbę warunków w zbiorach reguł, dla problemu klasyfikacji.

Tabela 7.26: Statystyki sumarycznej liczby warunków w zbiorze dla problemu klasyfikacji (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

Metoda	lf	uf	q1	q3	średnia	mediana
Drzewo decyzyjne	33.0	2336.0	117.0	1008.0	681.652	316.0
RuleKit	10.0	389.0	41.0	232.0	215.509	131.0
ComplexConditions	4.0	249.0	37.0	158.0	116.788	84.0
MofNRules („dokładnie M-z-N”)	4.0	344.0	27.0	173.0	123.538	85.0
MofNRules („przynajmniej M-z-N”)	4,.0	337.0	28.0	174.0	129.187	80.0
DeepRules	4.0	264.0	32.0	134.0	103.758	76.0



Rysunek 7.16: CD diagram dla sumarycznej liczby warunków w zbiorach reguł klasyfikacyjnych ($\alpha = 0.05$; CD = 1.424).

„dokładnie M-z-N" i 28 dla wariantu „przynajmniej M-z-N"). Dla wszystkich zaproponowanych metod zanotowano mniejszą sumaryczną liczbę warunków w zbiorze reguł niż dla obydwu metod referencyjnych. Potwierdza to wnioski z poprzednich podsekcji. Wyniki te sugerują również, że zbiory reguł generowane metodą DeepRules są realnie prostsze pod kątem sumarycznej liczby warunków niż te generowane innymi metodami, mimo że same reguły mogą zawierać więcej warunków w przesłankach.

Aby ocenić, na ile istotne są wykryte wcześniej różnice, przeprowadzono, tak jak poprzednio, test Friedmana dla $\alpha = 0.05$. Wartość statystyki testu wyniosła 86.0, a p-wartość równa była 4.634e-17. Potwierdza to istnienie istotnych statystycznie różnic pomiędzy przynajmniej jedną parą algorytmów. W celu zidentyfikowania tej pary lub par, przeprowadzono test Nemenyi (również dla $\alpha = 0.05$), którego wyniki zwizualizowano z użyciem CD diagramu (patrz rysunek 7.16).

Potwierdza on, że najlepszym algorytmem pod względem najmniejszej sumarycznej liczby warunków w zbiorze jest DeepRules. Nie wykazał on jednak istotnej różnicy względem metod ComplexConditions i MofNRules w obydwu wariantach. Okazał się on jednak w sposób istotny lepszy od metod referencyjnych. Najgorszym algorytmem spośród wszystkich badanych były drzewa decyzyjne, prowadząc do zbiorów o największej sumarycznej liczbie warunków. Jest to zgodne z obserwacjami z poprzednich analiz.

Na koniec przeprowadzony został test Wilcoxona dla poziomu istotności $\alpha = 0.05$. Jego wyniki zaprezentowano w tabeli 7.27. Wykazał on istotne różnice dla niemal wszystkich par algorytmów, z wyjątkiem RuleKit i drzew decyzyjnych.

Tabela 7.27: P-wartości testów Wilcoxona porównujących sumaryczną liczbę warunków w zbiorach reguł ($\alpha = 0.05$, korekta FDR).

	CC	DR	DD	MNd	MNp	RK
CC	–	0.0	0.0	0.0	0.0	0.0
DR	0.0	–	0.0	0.0	0.0	0.0
DD	0.0	0.0	–	0.0	0.0	0.1
MNd	0.0	0.0	0.0	–	0.03	0.0
MNp	0.0	0.0	0.0	0.03	–	0.0
RK	0.0	0.0	0.1	0.0	0.0	–

Tabela 7.28: Statystyki liczby reguł dla problemu regresji (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

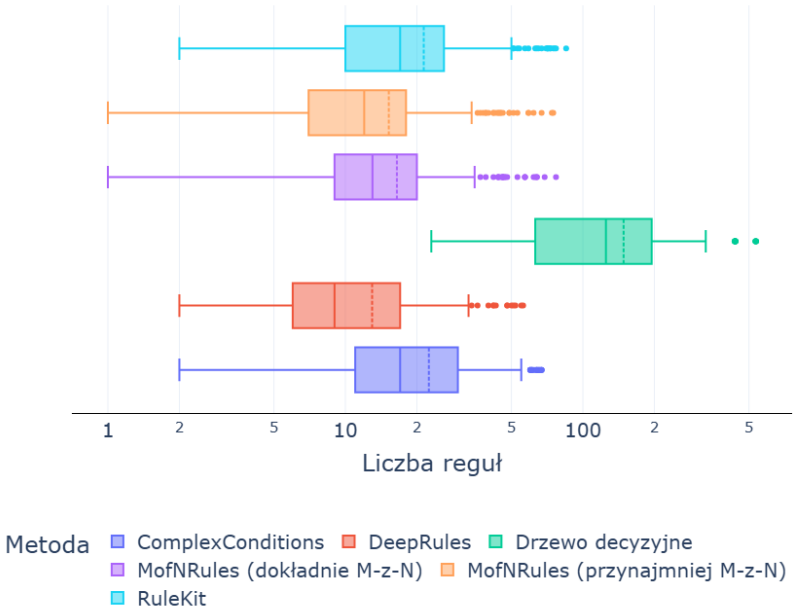
Metoda	lf	uf	q1	q3	średnia	mediana
Drzewo decyzyjne	23.0	329.0	63.0	194.5	148.329	125.0
RuleKit	2.0	50.0	10.0	26.0	21.373	17.0
ComplexConditions	2.0	55.0	11.0	29.5	22.461	17.0
MofNRules („dokładnie M-z-N”)	1.0	35.0	9.0	20.0	16.475	13.0
MofNRules („przynajmniej M-z-N”)	1.0	34.0	7.0	18.0	15.231	12.0
DeepRules	2.0	33.0	6.0	17.0	12.959	9.0

5.2 Problem regresji

Analiza liczby reguł

Analiza złożoności zbiorów reguł dla problemu regresji przeprowadzona została w sposób analogiczny jak miało to miejsce dla klasyfikacji. Pierwszym badanym wskaźnikiem była liczba reguł generowana przez poszczególne metody. Jej rozkład zwizualizowany został na wykresie pudełkowym widocznym na rysunku 7.17. Dokładne wartości statystyk, na podstawie których został wykreślony, podejrzeć można w tabeli 7.28.

Na ich podstawie zauważyć można, że podobnie jak w przypadku klasyfikacji, algorytm DeepRules generował zbiory o najmniejszej liczbie reguł. Widać to zarówno po najniższej średniej jak i medianie (kolejno 12.96 i 9), ale również po najniższym górnym i dolnym kwartylu (kolejno 6 i 17). Różnica ta jest dobrze widoczna, jednak mniej niż miało to miejsce w przypadku klasyfikacji. Należy tutaj zwrócić uwagę, że dla niemal wszystkich metod liczba reguł dla problemu regresji jest znacząco niższa, niż miało to miejsce w przypadku klasyfikacji. W przypadku RuleKit wartość średnia zmieniała się z 60.58 na 21.37, dla ComplexConditions była to zmiana z 43.86

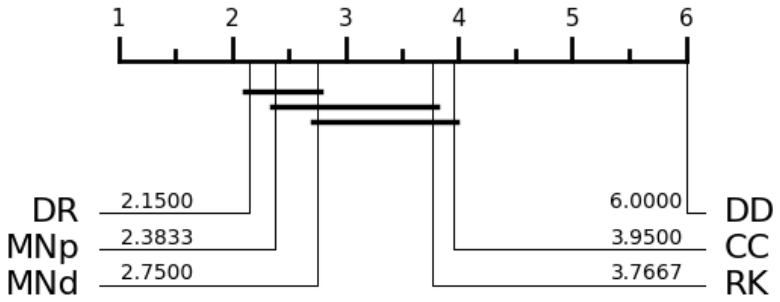


Rysunek 7.17: Wykres pudełkowy prezentujący liczbę reguł w wygenerowanych zbiorach, dla problemu regresji.

na 22.46, dla MofNRules w wariancie „dokładnie M-z-N" z 42.96 na 16.48, a dla „przynajmniej M-z-N" z 43.54 na 15.23. Mniejsza różnica wystąpiła w przypadku DeepRules, gdzie średnia spadła z 14.96 do wartości 12.96. Wyjątek stanowiły tutaj drzewa decyzyjne, dla których wartość średnia wzrosła ponad dwukrotnie z 55.97 na 148.33. Sugeruje to, że w przypadku zbiorów regresyjnych metody regułowe oparte o sekwencyjne pokrywanie zdolne są generować mniej złożone opisy danych, osiągając przy tym lepsze zdolności predykcyjne (patrz podsekcja 4.2).

Metoda MofNRules w obydwu wariantach okazała się lepsza od referencyjnego algorytmu RuleKit, osiągając zarówno niższą wartość średnią (16.48 dla wariantu „dokładnie M-z-N" i 15.23 dla wariantu „przynajmniej M-z-N"), jak i medianę oraz dolny i górny kwartyl. Spośród proponowanych w pracy metod, jedynie ComplexConditions okazał się nieznacznie gorszy (średnia równa 22.46 i górny kwartyl równy 29.5). Jego wyniki wydają się jednak bardzo zbliżone do tych osiąganych przez RuleKit.

Test Friedmana dla $\alpha = 0.05$ potwierdził istnienie statystycznie istotnych różnic pomiędzy przynajmniej jedną parą metod (wartość statystyki równa 88.997 i wartość p równa 1.091e-17). Aby określić, dla których, wykonano test Nemenyi dla tej samej



Rysunek 7.18: CD diagram dla liczby reguł regresyjnych ($\alpha = 0.05$; CD = 1.424).

Tabela 7.29: P-wartości testów Wilcoxona porównujących liczbę wygenerowanych reguł ($\alpha = 0.05$, korekta FDR).

	CC	DR	DD	MNd	MNp	RK
CC	–	0.0	0.0	0.0	0.0	0.20
DR	0.0	–	0.0	0.01	0.24	0.0
DD	0.0	0.0	–	0.0	0.0	0.0
MNd	0.0	0.01	0.0	–	0.0	0.0
MNp	0.0	0.24	0.0	0.0	–	0.0
RK	0.20	0.0	0.0	0.0	0.0	–

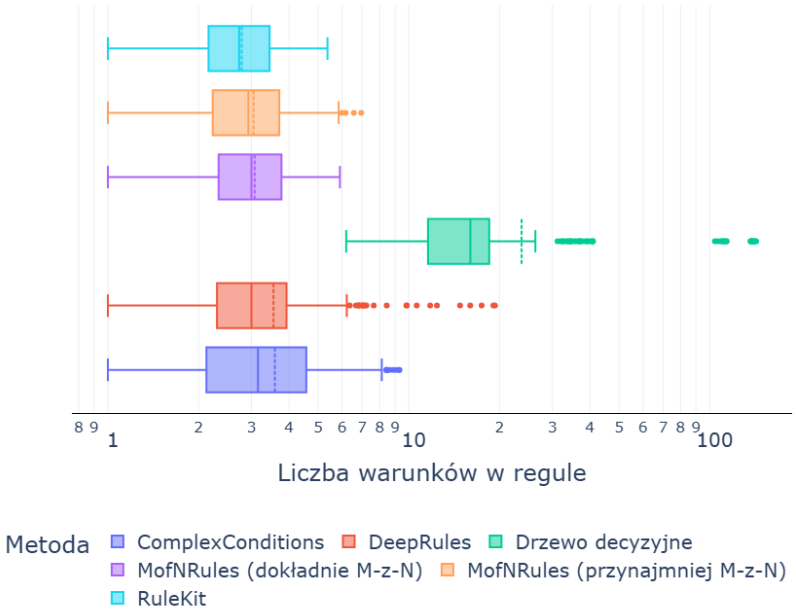
wartości α, którego wyniki zwizualizowano z użyciem CD diagramu (patrz rysunek 7.18).

Grupą metod wyznaczających najmniejszą liczbę reguł, nie różniącą się między sobą w istotny sposób, są w kolejności od najlepszego do najgorszego: DeepRules, MofNRues wariant „przynajmniej M-z-N" oraz wariant „dokładnie M-z-N". Test Nemenyi potwierdził, że drzewa decyzyjne generują na ogół znacząco większą liczbę reguł od pozostałych algorytmów. Metoda ComplexConditions okazała się tutaj gorsza od RuleKit, jednak różnica ta nie była istotna statystycznie.

Na koniec przeprowadzony został dodatkowo test Wilcoxona dla $\alpha = 0.05$, który sugeruje istnienie statystycznie istotnych różnic pomiędzy większością metod. Nie wystąpiła ona jedynie pomiędzy parami DeepRules i MofNRules (wariant „przynajmniej M-z-N"), oraz RuleKit i ComplexConditions.

Analiza średniej liczby warunków w regule

Rozkład średniej liczby warunków w regule dla problemu regresji, dla poszczególnych metod, przedstawiony został na wykresie pudełkowym widocznym na rysunku 7.19.



Rysunek 7.19: Wykres pudełkowy prezentujący średnią liczbę warunków w regułach, dla problemu regresji.

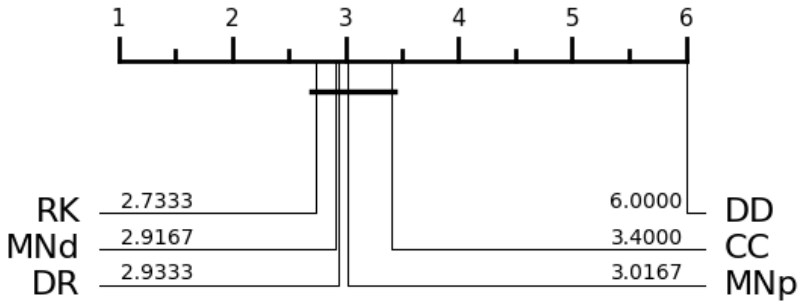
W tabeli 7.30 widoczne są dokładne wartości statystyk, na podstawie których został on wyrysowany.

Mamy w tym wypadku do czynienia z sytuacją odmienną niż miało to miejsce dla klasyfikacji. Żaden z proponowanych algorytmów nie osiągnął niższej wartości średniej i mediany niż referencyjny algorytm RuleKit, dla którego wyniosła ona kolejno (2.78 i 2.73). Możemy jednak zauważyć, że dla większości metod wyniki są do siebie zbliżone (dolny kwartyl wynosi około 2, a górny około 4). Wyjątkiem są tutaj drzewa decyzyjne, gdzie dolny i górny kwartyl są znacząco wyższe (kolejno 11.58 i 18.5). W przeciwieństwie do wyników dla problemu klasyfikacji, tutaj algorytm DeepRules nie wykazał silnej tendencji do generowania dłuższych reguł. Średnia wartość metryki wyniosła 3.5, gdy dla klasyfikacji była ona równa 6.45. Dla DeepRules, na wykresie pudełkowym widoczne są jednak liczne wartości odstające wykraczające poza zakres wartości pozostałych algorytmów (z wyłączeniem drzew decyzyjnych). Sugeruje to, że reguły uzyskiwane tą metodą mogą w pewnych rzadkich sytuacjach zawierać znacząco większą liczbę warunków.

Test Friedmana, przeprowadzony na poziomie istotności $\alpha = 0.05$, potwierdził istnienie istotnych statystycznie różnic pomiędzy badanymi metodami. Wartość

Tabela 7.30: Statystyki średniej liczby warunków w regule reguł dla problemu regresji (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

Metoda	lf	uf	q1	q3	średnia	mediana
Drzewo decyzyjne	6.194	26.34	11.583	18.495	23.7	16.019
RuleKit	1.0	5.368	2.162	3.44	2.784	2.73
ComplexConditions	1.0	8.132	2.131	4.558	3.589	3.154
MofNRules („dokładnie M-z-N”)	1.0	5.903	2.333	3.771	3.074	3.0
MofNRules („przynajmniej M-z-N”)	1.0	5.846	2.231	3.71	3.048	2.929
DeepRules	1.0	6.222	2.305	3.919	3.551	3.0



Rysunek 7.20: CD diagram dla średniej liczby warunków w regule dla problemu regresji ($\alpha = 0.05$; CD = 1.424).

statystyki testu wyniosła 55.94, a p-wartość równa była 4.433e-13. Aby zidentyfikować pary różniących się między sobą algorytmów, wykonano test Nemenyi dla tej samej wartości α. Jego wyniki zwizualizowano ponownie z użyciem CD diagramu na rysunku 7.20.

Potwierdza on wcześniejszą obserwację o podobieństwie pomiędzy wynikami zaproponowanych w pracy metod a tymi osiąganymi przez algorytm RuleKit. Test nie wykazał dla nich istotnych statystycznie różnic. Algorytmy uszeregowane od najlepszego do najgorszego to kolejno: RuleKit, MofNRules w wariancie „dokładnie M-z-N”, DeepRules, MofNRules wariant „przynajmniej M-z-N”, ComplexConditions i drzewa decyzyjne. Te ostatnie okazały się w sposób znaczący gorsze od pozostałych metod.

Dodatkowo wykonano test Wilcoxona dla par obserwacji, na poziomie istotności 0.05, w celu zidentyfikowania innych potencjalnie istotnych różnic, które mogły zostać pominięte w teście Nemenyi. Jego wyniki zaprezentowano w tabeli 7.31.

Tabela 7.31: P-wartości testów Wilcoxona porównujących średnią liczbę warunków w regule ($\alpha = 0.05$, korekta FDR).

	CC	DR	DD	MNd	MNp	RK
CC	–	0.0	0.0	0.0	0.0	0.0
DR	0.0	–	0.0	0.16	0.07	0.0
DD	0.0	0.0	–	0.0	0.0	0.0
MNd	0.0	0.16	0.0	–	0.49	0.0
MNp	0.0	0.07	0.0	0.49	–	0.0
RK	0.0	0.0	0.0	0.0	0.0	–

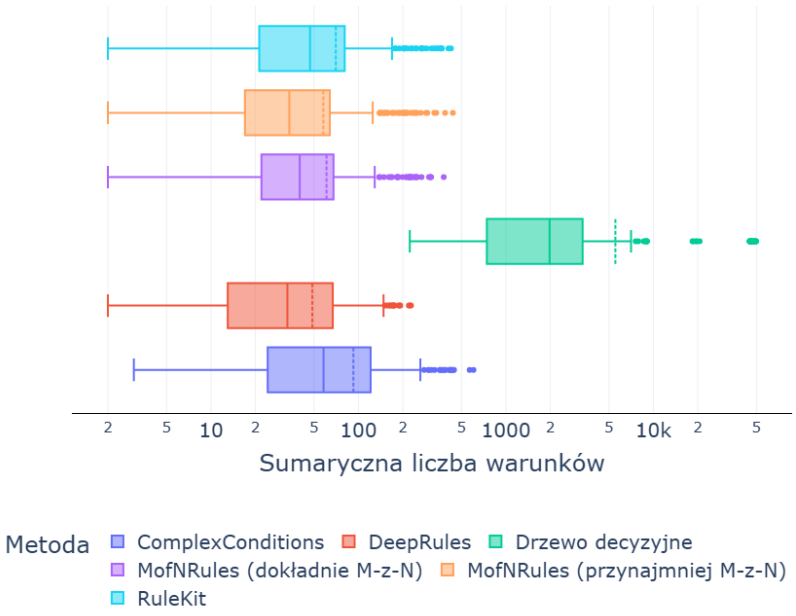
Wyniki testu sugerują istnienie istotnej statystycznie różnicy pomiędzy większością par algorytmów. Nie wykazano jej jednak pomiędzy DeepRules oraz obydwoma wariantami metody MofNRules. Dwa warianty nie różnią się także względem siebie w sposób istotny statystycznie.

Analiza sumarycznej liczby warunków w zbiorze reguł

Wcześniejsze analizy wykazały, że proponowane przez autora metody generują na ogół mniej reguł od referencyjnych algorytmów (RuleKit i drzewa decyzyjne). Same reguły posiadają jednak w przypadku tych pierwszych większą liczbę warunków w przesłankach. Na tej podstawie trudno rozstrzygnąć, czy opracowane algorytmy rzeczywiście doprowadziły do mniej złożonych zbiorów reguł. Aby to określić, pomocna może być analiza sumarycznej liczby warunków pojawiających się w całych zbiorach reguł.

W pierwszym kroku wyrysowany został wykres pudełkowy dla wartości sumarycznej liczby warunków w zbiorach dla wszystkich badanych metod. Widoczny jest on na rysunku 7.21. Dokładne wartości statystyk podejrzeć można w tabeli 7.32.

Jak można zauważyć na ich podstawie, drzewa decyzyjne okazały się prowadzić do zbiorów reguł o największej sumarycznej liczbie warunków (średnia równa 5526.46 i mediana równa 1978.0). Jest to widocznie wyższy wynik niż dla pozostałych badanych metod. Warto zwrócić w tym miejscu uwagę na fakt, że wartość średnia wykracza poza górny kwartyl, co sugeruje silnie prawostronną skośność rozkładu. Metoda MofNRules w wariancie „„przynajmniej M-z-N”” osiągnęła najniższą wartość górnego kwartyla (64). Jej średnia i mediana wyniosły, odpowiednio, 57,77 i 34. Drugi wariant tej metody odnotował stosunkowo podobne wyniki, ze średnią wynoszącą 60,76 oraz medianą równą 40. Algorytm DeepRules osiągnął najniższą wartość średniej (48.65), mediany (33), oraz dolnego kwartyla (13) spośród wszystkich badanych metod.



Rysunek 7.21: Wykres pudełkowy prezentujący sumaryczną liczbę warunków w zbiorach reguł, dla problemu regresji.

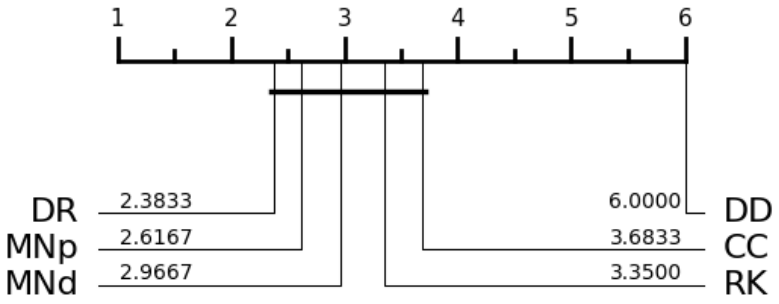
Cechuje się jednak wyższym rozstępem międzykwartylowym niż ma to miejsce dla RuleKit i MofNRules. Rozstęp ten jest na zbliżonym poziomie również w przypadku algorytmu ComplexConditions. Ten z kolei prowadził do najbardziej złożonych, ze względu na sumaryczną liczbę warunków, zbiorów reguł (wartość średnia równa 92.48 i mediana równa 58.0).

Wartość statystyki testu Friedmana, przeprowadzanego dla $\alpha = 0.05$, wyniosła 74.51, a p-wartość równa była 1.18e-14. Przemawia to za istnieniem istotnych statystycznie różnic pomiędzy wynikami osiąganymi przez poszczególne metody. Aby zidentyfikować różniące się między sobą algorytmy, wykonano test Nemenyi, również na poziomie istotności $\alpha = 0.05$. Jego wyniki zaprezentowano ponownie z użyciem CD diagramu na rysunku 7.22.

Po raz kolejny najwyższą sumaryczną liczbę warunków w zbiorach wykazały drzewa decyzyjne. Różnica ta okazała się istotna statystycznie względem reszty badanych metod. W przypadku pozostałych algorytmów test nie wykazał znaczących różnic pomiędzy nimi. Najlepszym z nich okazał się DeepRules, następny w kolejności był MofNRules w wariancie „przynajmniej M-z-N” oraz jego drugi wariant. Metoda

Tabela 7.32: Statystyki sumarycznej liczby warunków w zbiorze dla problemu regresji (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

Metoda	lf	uf	q1	q3	średnia	mediana
Drzewo decyzyjne	223.0	7057.0	745.5	3307.0	5526.464	1978.0
RuleKit	2.0	169.0	21.5	80.5	70.217	47.0
ComplexConditions	3.0	263.0	24.5	121.0	92.481	58.0
MofNRules („dokładnie M-z-N”)	2.0	129.0	22.0	67.5	60.756	40.0
MofNRules („przynajmniej M-z-N”)	2.0	125.0	17.0	64.0	57.773	34.0
DeepRules	2.0	148.0	13.0	67.0	48.654	33.0



Rysunek 7.22: CD diagram dla sumarycznej liczby warunków w zbiorach reguł regresyjnych ($\alpha = 0.05$; CD = 1.424).

Tabela 7.33: P-wartości testów Wilcoxona porównujących sumaryczną liczbę warunków w zbiorach reguł ($\alpha = 0.05$, korekta FDR).

	CC	DR	DD	MNd	MNp	RK
CC	–	0.0	0.0	0.0	0.0	0.2
DR	0.0	–	0.0	0.0	0.24	0.0
DD	0.0	0.0	–	0.0	0.0	0.0
MNd	0.0	0.0	0.0	–	0.0	0.0
MNp	0.0	0.24	0.0	0.0	–	0.0
RK	0.2	0.0	0.0	0.0	0.0	–

ComplexConditions osiągnęła gorsze rezultaty niż referencyjny RuleKit, co potwierdza poprzednie obserwacje.

W następnym kroku przeprowadzono test Wilcoxona dla par obserwacji dla tej samej wartości $\alpha = 0.05$, którego wyniki widoczne są w tabeli 7.33.

Wykazał on istotne różnice dla większości par algorytmów. Wyjątkami są tutaj pary: RuleKit i ComplexConditions oraz DeepRules i MofNRules w wariancie „przynajmniej M-z-N".

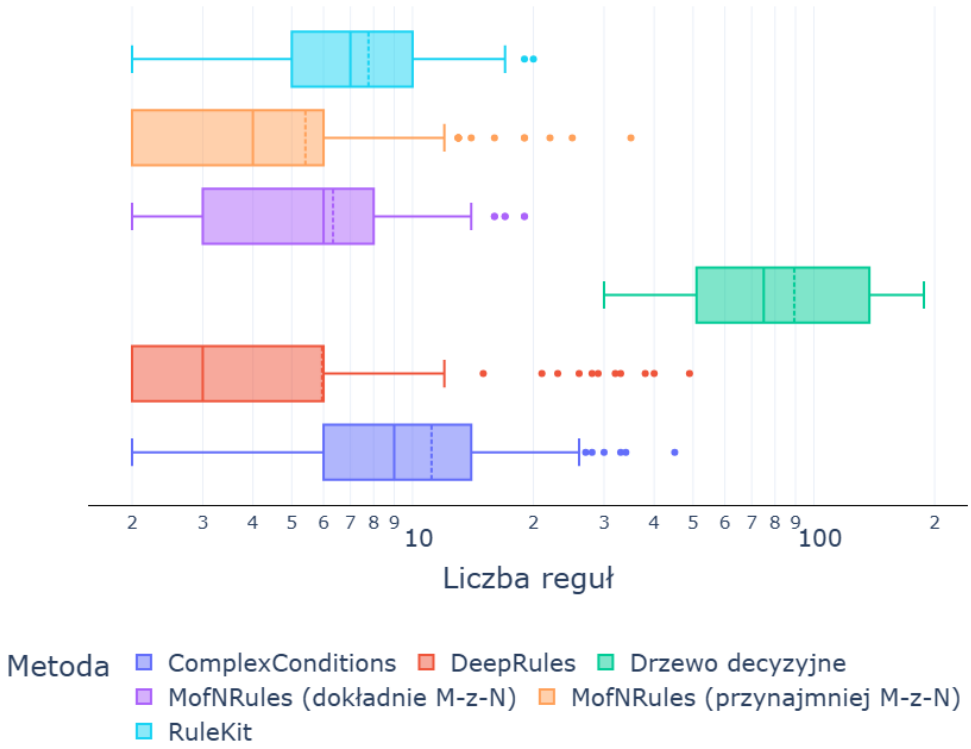
Powyższe wyniki i przeprowadzone analizy sugerują, że metody DeepRules i MofNRules mogą w przypadku problemu regresji prowadzić do zbiorów o mniejszej liczbie reguł oraz sumarycznej liczbie warunków niż ma to miejsce dla referencyjnego algorytmu RuleKit. Liczba warunków w przesłankach poszczególnych reguł pozostaje jednak na zbliżonym, nieco gorszym poziomie. Algorytm ComplexConditions okazał się w przypadku regresji generować na ogół większe zbiory reguł o dłuższych przesłankach, niż ma to miejsce dla RuleKit, MofNRules oraz DeepRules.

5.3 Problem analizy przeżycia

Analiza liczby reguł

Analizę złożoności zbiorów reguł dla problemu analizy przeżycia rozpoczęto tak jak poprzednio od zbadania, jak rozkładała się liczba reguł w zależności od badanej metody. W pierwszym kroku sporządzony został wykres pudełkowy widoczny na rysunku 7.23. Dokładne wartości statystyk, na podstawie których został sporządzony, znajdują się w tabeli 7.34.

Na ich podstawie, podobnie jak miało to miejsce w przypadku regresji, zauważyć można, że algorytm ComplexConditions (średnia równa 11.15 i mediana równa 9.0) prowadzi na ogół do zbiorów o większej liczbie reguł niż referencyjny RuleKit



Rysunek 7.23: Wykres pudełkowy prezentujący liczbę reguł w wygenerowanych zbiorach, dla problemu analizy przeżycia.

Tabela 7.34: Statystyki liczby reguł dla problemu analizy przeżycia (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

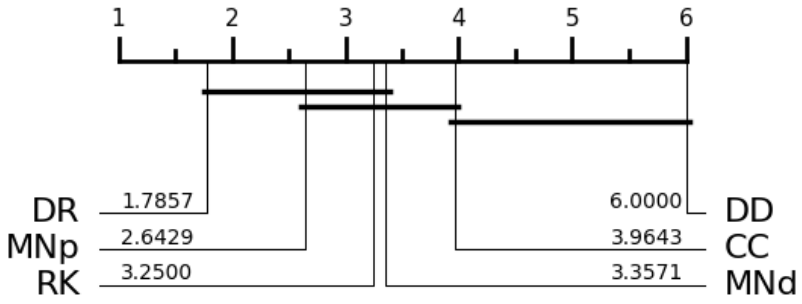
Metoda	lf	uf	q1	q3	średnia	mediana
Drzewo decyzyjne	30.0	188.0	51.0	137.25	89.336	75.0
RuleKit	2.0	17.0	5.0	10.0	7.764	7.0
ComplexConditions	2.0	26.0	6.0	14.0	11.15	9.0
MofNRules („dokładnie M-z-N”)	2.0	14.0	3.0	8.0	6.321	6.0
MofNRules („przynajmniej M-z-N”)	2.0	12.0	2.0	6.0	5.379	4.0
DeepRules	2.0	12.0	2.0	6.0	5.95	3.0

(średnia równa 7.76 i mediana równa 7.0). Drzewa decyzyjne po raz kolejny cechowały się największymi pod kątem liczby reguł zbiorami, daleko przewyższając pod tym względem pozostałe metody (dolny kwartyl równy 51 i górny wynoszący 137.25). Algorytmy DeepRules oraz MofNRules wykazały tendencje do generowania mniejszej liczby reguł. Co ciekawe, wariant MofNRules „przynajmniej M-z-N” (mediana równa 4.0, górny kwartyl równy 6.0) okazał się w tym wypadku widocznie lepszy od drugiego (mediana wynosząca 6.0 i górny kwartyl równy 8.0). Algorytm DeepRules osiągnął najniższą medianę (3) oraz górny kwartyl (6,0) spośród wszystkich badanych metod. Jego wartość średnia (5.95) była z kolei nieznacznie wyższa niż miało to miejsce dla MofNRules w wariancie „przynajmniej M-z-N” (5.38)

W celu oceny istotności statystycznej zaobserwowanych różnic pomiędzy metodami przeprowadzono test Friedmana dla $\alpha = 0.05$. Wartość statystyki testu wyniosła 42.268, a p-wartość równa była 5.199e-8. Wynik ten potwierdza istnienie istotnych statystycznie różnic pomiędzy przynajmniej jedną parą algorytmów. W następnym kroku przeprowadzono test post hoc Nemenyi, również dla $\alpha = 0.05$. Jego wyniki zobrazowane zostały z użyciem CD diagramu na rysunku 7.24.

Najlepszym algorytmem, z punktu widzenia najniższej liczby generowanych reguł, okazał się ponownie DeepRules. Następne w kolejności były: MofNRules w wariancie „przynajmniej M-z-N”, RuleKit, MofNRules w wariancie „dokładnie M-z-N”, ComplexConditions, oraz ostatnie drzewa decyzyjne. Test nie wykazał istotnych różnic pomiędzy czterema pierwszymi metodami. Potwierdzono jednak ich istnienie pomiędzy algorytmem DeepRules a ComplexConditions oraz drzewami. Te dwa ostatnie nie różniły się między sobą w istotny sposób.

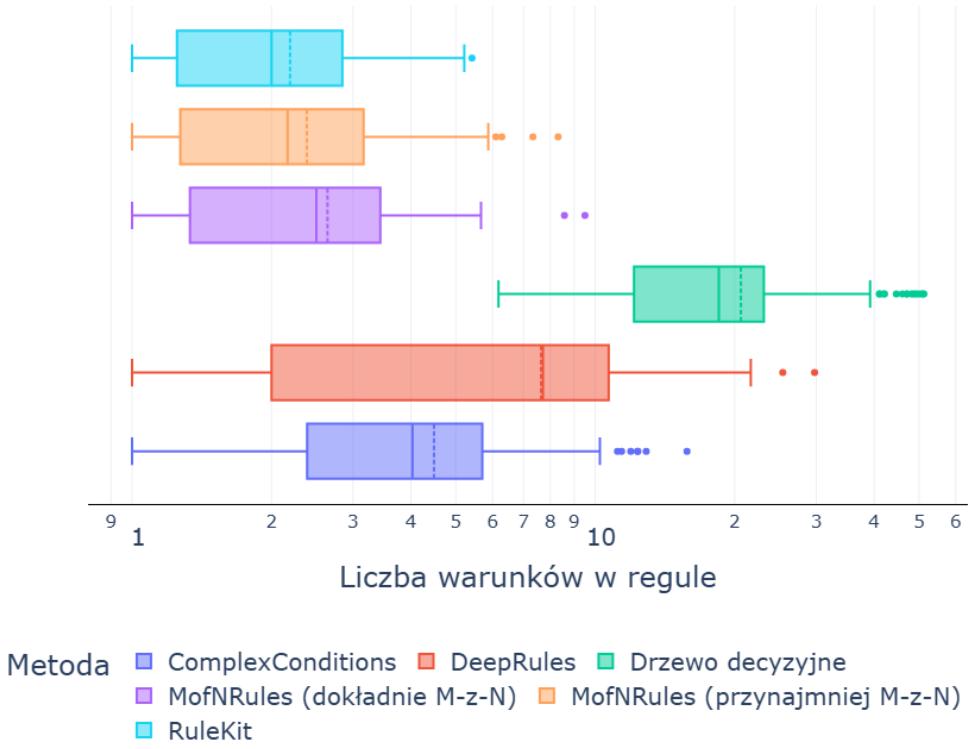
Na sam koniec przeprowadzono test Wilcoxona dla par obserwacji, na poziomie istotności $\alpha = 0.05$. Jego wyniki widoczne są w tabeli 7.35. Jako test mniej konserwa-



Rysunek 7.24: CD diagram dla liczby reguł przeżyciowych ($\alpha = 0.05$; CD = 2.085).

Tabela 7.35: P-wartości testów Wilcoxona porównujących liczbę wygenerowanych reguł ($\alpha = 0.05$, korekta FDR).

	CC	DR	DD	MNd	MNp	RK
CC	–	0.0	0.0	0.0	0.0	0.0
DR	0.0	–	0.0	0.0	0.04	0.0
DD	0.0	0.0	–	0.0	0.0	0.0
MNd	0.0	0.0	0.0	–	0.0	0.0
MNp	0.0	0.04	0.0	0.0	–	0.0
RK	0.0	0.0	0.0	0.0	0.0	–



Rysunek 7.25: Wykres pudełkowy prezentujący średnią liczbę warunków w regułach, dla problemu analizy przeżycia.

tywny niż test Nemenyi, wykazał on znaczące różnice pomiędzy wszystkimi parami badanych metod.

Analiza średniej liczby warunków w regule

Po zbadaniu rozkładu liczby reguł w zbiorach generowanych z użyciem poszczególnych metod przystąpiono do analizy średniej liczby warunków w przesłankach tych reguł. Rozpoczęto ją, tak jak wcześniej, od wyrysowania wykresu pudełkowego widocznego na rysunku 7.25. Dokładne wartości statystyk, które wizualizowano, można odczytać w tabeli 7.36.

Na ich podstawie można zauważyć, że po raz kolejny drzewa decyzyjne prowadziły do generowania najdłuższych reguł. Różnica ta jest wyraźnie widoczna po największej wartości dolnego kwartyla (22.95), znacznie wyższej niż dla pozostałych metod. Algorytm DeepRules cechuje się tutaj największym rozstępem międzykwartylowym, z górnym kwartylem równym 10.679 i dolnym wynoszącym 2.0. Zanotował on również

Tabela 7.36: Statystyki średniej liczby warunków w regule reguł dla problemu analizy przeżycia (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

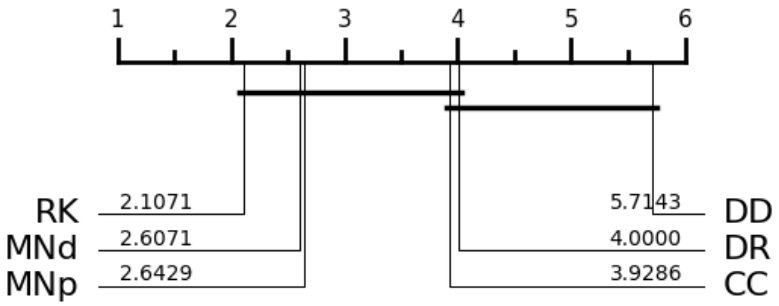
Metoda	lf	uf	q1	q3	średnia	mediana
Drzewo decyzyjne	6.182	39.203	12.15	22.953	20.626	18.483
RuleKit	1.0	5.214	1.25	2.822	2.193	2.0
ComplexConditions	1.0	10.233	2.394	5.698	4.486	4.031
MofNRules („dokładnie M-z-N")	1.0	5.667	1.333	3.418	2.642	2.5
MofNRules („przynajmniej M-z-N")	1.0	5.875	1.312	3.17	2.384	2.167
DeepRules	1.0	21.667	2.0	10.679	7.639	7.708

wyższą wartość średnią (7.64) i medianę (7.7), niż RuleKit oraz pozostałe metody zaproponowane w niniejszej pracy. Wyniki dla referencyjnego algorytmu RuleKit oraz MofNRules są do siebie stosunkowo zbliżone pod względem wartości górnego i dolnego kwartyla. Ten drugi osiągnął jednak nieco wyższą wartość średnią i medianę (kolejno 2.38 i 2.17 dla wariantu „przynajmniej M-z-N", oraz 3.42 i 2.64 dla „dokładnie M-z-N"). ComplexConditions okazał się tutaj widocznie gorszy od MofoNRules w dolnym kwartylem równym 2.39 i górnym wynoszącym 5.698.

Do zbadania czy między analizowanymi algorytmami występują istotne statystycznie różnice w średniej liczbie warunków w regule, przeprowadzony został test Friedmana dla $\alpha = 0.05$. Potwierdził on ich istnienie pomiędzy przynajmniej jedną parą metod (wartość statystyki testu wyniosła 35.82, p-wartość równa była 1.032e-6). W celu zidentyfikowania różniących się między sobą algorytmów wykonano test Nemenyi na poziomie istotności $\alpha = 0.05$. Jego wyniki zwizualizowano z użyciem CD diagramu, widocznego na rysunku 7.26.

Potwierdza on wcześniejsze obserwacje. Najlepszym algorytmem okazał się tutaj referencyjny RuleKit. Następna w kolejności była metoda MofNRules z nieznaczną przewagą wariantu „dokładnie M-z-N". Najgorszą metodą okazały się drzewa decyzyjne, lepsze od nich były kolejno DeepRules i ComplexConditions. Test Nemenyi wykazał dwie grupy metod nie różniące się między sobą w istotny sposób. Pierwszą z nich tworzą: RuleKit, MofNRules, ComplexConditions i DeepRules. Do drugiej należą: ComplexConditions, DeepRules i drzewa decyzyjne.

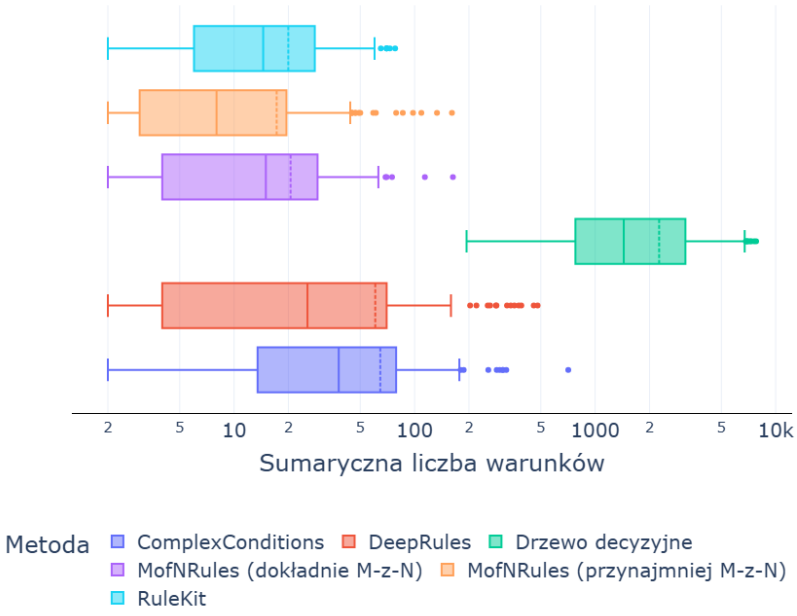
Na koniec przeprowadzony został test Wilcoxona dla par obserwacji, na poziomie istotności $\alpha = 0.05$. Potwierdził on istnienie istotnych różnic pomiędzy niemal wszyst-



Rysunek 7.26: CD diagram dla średniej liczby warunków w regule dla problemu analizy przeżycia ($\alpha = 0.05$; CD = 2.085).

Tabela 7.37: P-wartości testów Wilcoxona porównujących średnią liczbę warunków w regule ($\alpha = 0.05$, korekta FDR).

	CC	DR	DD	MNd	MNp	RK
CC	–	0.0	0.0	0.0	0.0	0.0
DR	0.0	–	0.0	0.0	0.0	0.0
DD	0.0	0.0	–	0.0	0.0	0.0
MNd	0.0	0.0	0.0	–	0.04	0.02
MNp	0.0	0.0	0.0	0.04	–	0.25
RK	0.0	0.0	0.0	0.02	0.25	–



Rysunek 7.27: Wykres pudełkowy prezentujący sumaryczną liczbę warunków w zbiorach reguł, dla problemu analizy przeżycia.

kimi parami metod. Wyjątkiem była tutaj para RuleKit i MofNRules w wariancie „przynajmniej M-z-N".

Analiza sumarycznej liczby warunków w zbiorze reguł

W przypadku problemu przeżycia ponownie wykazano, że niektóre z proponowanych w pracy metod generują na ogół mniejszą liczbę reguł zawierających większą liczbę warunków w przesłankach niż referencyjny algorytm RuleKit. Aby ocenić, czy faktycznie doszło w tym wypadku do uzyskania mniej złożonych zbiorów reguł, przeprowadzono dodatkowo analizę sumarycznej liczby wszystkich warunków. Jako iloczyn dwóch wcześniej badanych metryk pozwala ocenić całościowy stopień skomplikowania zbioru reguł.

W pierwszym kroku sporządzony został wykres pudełkowy badanej metryki widoczny na rysunku 7.27. Dokładne wartości prezentowanych na nim metryk widoczne są w tabeli 7.38.

Na ich podstawie można zaobserwować, że drzewa decyzyjne prowadzą do najbardziej złożonych zbiorów reguł pod względem sumarycznej liczby warunków. Widać to po wysokiej wartości dolnego kwartyla (810.75), daleko przewyższającego war-

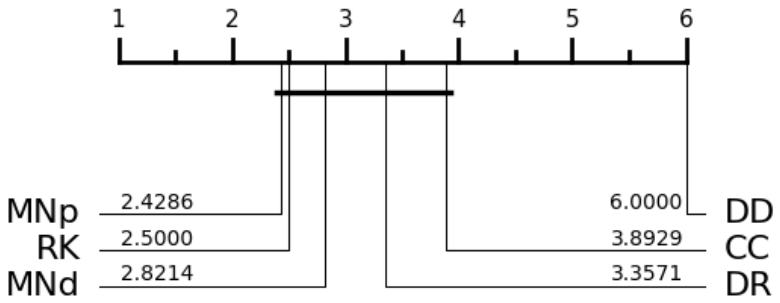
Tabela 7.38: Statystyki sumarycznej liczby warunków w zbiorze dla problemu analizy przeżycia (lf - dolne ogrodzenie, uf - górne ogrodzenie, q1 - kwartyl dolny, q3 - kwartyl górny).

Metoda	lf	uf	q1	q3	średnia	mediana
Drzewo decyzyjne	194.0	6734.0	810.75	3149.5	2261.779	1440.5
RuleKit	2.0	60.0	6.0	28.0	19.971	14.5
ComplexConditions	2.0	177.0	13.75	79.0	64.55	38.0
MofNRules („dokładnie M-z-N”)	2.0	63.0	4.0	29.0	20.557	15.0
MofNRules („przynajmniej M-z-N”)	2.0	44.0	3.0	19.25	17.214	8.0
DeepRules	2.0	159.0	4.0	69.5	60.55	25.5

tości górnych kwartyli pozostałych metod. Spośród wszystkich zaproponowanych w pracy metod, jedynie MofNRules, w wariancie „przynajmniej M-z-N”, osiągnął niższą wartość średnią (17.21) i medianę (8.0) niż referencyjny algorytm RuleKit (średnia równa 19.97 i mediana wynosząca 14.5). Co ciekawe, drugi wariant tej metody okazał się widocznie gorszy ze średnią wartością wynoszącą 20.56 i medianą równą 15.0. ComplexConditions cechował się tutaj dużo wyższą sumaryczną liczbą warunków. Jego górny kwartyl wyniósł 79.0, a mediana równa była 38.0. Algorytm DeepRules osiągnął ponownie najwyższy rozstęp międzykwartylowy spośród wszystkich zaproponowanych przez autora metod (kwartyl dolny równy 4.0 i górny równy 69.5). W jego wypadku mediana była na poziomie 25.5. Wszystko to sugeruje, że metoda DeepRules osiągała bardzo zróżnicowane wyniki, w pewnych sytuacjach prowadząc do prostszych zbiorów reguł niż referencyjny algorytm RuleKit. Na ogół jednak zbiory reguł generowane z jej użyciem zawierały większą sumaryczną liczbę warunków.

Test Friedmana przeprowadzony na poziomie istotności $\alpha = 0.05$ potwierdził istnienie istotnych statystycznie różnic pomiędzy przynajmniej jedną parą metod (wartość statystyki testu wyniosła 36.809, a p-value równa była 6.542e-7). Aby zidentyfikować różniące się między sobą algorytmy, wykonano następnie test post hoc Nemenyi. Poziom istotności wyniósł ponownie 0.05. Jego wyniki zwizualizowano z użyciem CD diagramu widocznego na rysunku 7.28.

Na jego podstawie najlepszym algorytmem okazał się MofNRules w wariancie „przynajmniej M-z-N”. Następne były kolejno: RuleKit, drugi wariant MofNRules, DeepRules, ComplexConditions oraz drzewa decyzyjne. Te ostatnie okazały się w znaczący sposób gorsze od pozostałych metod.



Rysunek 7.28: CD diagram dla sumarycznej liczby warunków w zbiorach reguł przeżyciowych ($\alpha = 0.05$; CD = 2.085).

Tabela 7.39: P-wartości testów Wilcoxona porównujących sumaryczną liczbę warunków w zbiorach reguł ($\alpha = 0.05$, korekta FDR).

	CC	DR	DD	MNd	MNp	RK
CC	–	0.0	0.0	0.0	0.0	0.0
DR	0.0	–	0.0	0.0	0.04	0.0
DD	0.0	0.0	–	0.0	0.0	0.0
MNd	0.0	0.0	0.0	–	0.0	0.0
MNp	0.0	0.04	0.0	0.0	–	0.0
RK	0.0	0.0	0.0	0.0	0.0	–

Mniej konserwatywny test Wilcoxona dla par obserwacji, przeprowadzony dla $\alpha = 0.05$, wykazał z kolei istnienie istotnych statystycznie różnic pomiędzy wszystkimi parami badanych metod. Jego wyniki widoczne są w tabeli 7.39

6 Podsumowanie eksperymentów

Podsumowując przeprowadzone badania eksperymentalne, zaobserwować można widoczne różnice w wynikach poszczególnych metod dla trzech badanych typów problemów. Przykładem jest większa liczba reguł w indukowanych zbiorach reguł dla większości z nich w przypadku problemu klasyfikacji w porównaniu do regresji i analizy przeżycia. Może to wynikać ze specyfiki działania poszczególnych algorytmów. W problemie klasyfikacji każdy z przykładów uczących musi przynależeć do jednego z określonego zestawu klas decyzyjnych. Dla tak określonego podziału algorytmy generują następnie reguły opisujące każdą z klas. W ten sposób, aby uzyskać opis dla nich wszystkich, liczba reguł nie może być mniejsza od liczby klas decyzyjnych. W przypadku regresji z kolei każda z reguł reprezentuje pewną grupę przykładów cechujących się podobnymi wartościami atrybutu decyzyjnego. Liczba takich grup nie jest tutaj jednak określona z góry, jak ma to miejsce dla klasyfikacji. Podobna sytuacja występuje w przypadku analizy przeżycia, gdzie reguły grupują przykłady o zbliżonej do siebie krzywej przeżycia. Wszystko to sprawia, że pozornie te same algorytmy mogą zachowywać się zupełnie inaczej dla każdego z typów problemów, co świetnie widać na omawianych wcześniej wynikach.

6.1 Problem klasyfikacji

Dla problemu klasyfikacji, ze wszystkich proponowanych w pracy metod, tylko DeepRules wykazywał tendencje do osiągania wyższych wartości BAcc niż referencyjne metody. Testy statystyczne nie wykazały jednak istotnych różnic pomiędzy wszystkimi badanymi algorytmami, sugerując bardzo zbliżone zdolności predykcyjne.Wszystkie metody zaproponowane w pracy doprowadziły do mniej złożonych zbiorów niż te referencyjne, jeśli chodzi o sumaryczną liczbę warunków. DeepRules indukował na ogół znacząco mniejszą liczbę dłuższych reguł. Ostatecznie zawierały one jednak najmniej warunków w stosunku do tych uzyskiwanych z użyciem pozostałych opracowanych metod. Dla MofNRules liczba uzyskanych reguł była mniejsza niż dla algorytmów referencyjnych, były one również krótsze niż w przypadku RuleKit.

ComplexConditions indukował z kolei podobną do RuleKit liczbę reguł. Cechowały się one jednak znacznie mniejszą liczbą warunków w przesłankach.

6.2 Problem regresji

W przypadku problemu regresji również tylko DeepRules osiągnął lepsze wyniki predykcyjne (niższą wartość RRMSE) niż referencyjny RuleKit. Podobnie jak w przypadku klasyfikacji, testy statystyczne nie potwierdziły jednak, by były to różnice znaczące. Reszta proponowanych metod wykazywała zbliżone wyniki predykcyjne do drzew decyzyjnych. Algorytm ComplexConditions, dla problemu regresji, wyznaczał zbiory reguł o podobnej złożoności co RuleKit pod względem sumarycznej liczby warunków. Metody MofNRules w wariancie „przynajmniej M-z-N" i DeepRules były pod tym względem znacząco lepsze. Uzyskiwane za ich pomocą zbiory reguł wykazywały zarówno niższą liczbę reguł, jak i mniejszą sumaryczną liczbę warunków. Zanotować można ciekawą różnicę w wynikach dla obydwu wariantów MofNRules. Uzyskały one niemal identycznie wyniki predykcyjne, jednak wariant „przynajmniej M-z-N" okazał się generować znacząco mniej złożone zbiory reguł. Widać to zarówno po liczbie reguł, jak i sumarycznej liczbie warunków w ich przesłankach.

6.3 Problem analizy przeżycia

Ze wszystkich proponowanych metod, jedynie DeepRules wykazywał lepsze wyniki predykcyjne dla problemu analizy przeżycia niż metody referencyjne. Dla pozostałych z nich były one statystycznie zbliżone do tych osiąganych przez RuleKit. Dla algorytmu ComplexConditions, w przypadku analizy przeżycia, widoczna jest tendencja do generowania większej liczby reguł. Ustępował on pod tym względem jedynie drzewom decyzyjnym. Pod względem sumarycznej liczby warunków, zbiory generowane z użyciem ComplexConditions nie były, w sposób statystycznie istotny, bardziej złożone od tych indukowanych z użyciem RuleKit. Algorytmy DeepRules i MofNRules generowały na ogół mniej reguł o dłuższych przesłankach niż pozostałe. Co ciekawe, dla analizy przeżycia to wariant „dokładnie M-z-N" algorytmu MofNRules okazał się prowadzić do mniej złożonych zbiorów. Było to widoczne zarówno pod względem liczby reguł, jak i sumarycznej liczby warunków. Wariant ten okazał się jako jedyny przewyższać pod tymi względami algorytm RuleKit.

8 Studia przypadków

Opisane w poprzednim rozdziale eksperymenty ujawniły zalety i słabości metod opracowanych w ramach doktoratu. Z uwagi na dużą liczbę przeprowadzonych eksperymentów oraz zastosowanie walidacji krzyżowej, wyniki analizowano z użyciem metod statystycznych, nie dając wglądu w poszczególne zbiory reguł generowane przez każdą z nich.

Zaprezentowane w niniejszym rozdziale przykłady pozwalają bliżej przyjrzeć się działaniu zaproponowanych w pracy algorytmów na wybranych zbiorach danych. Na początek analizie zostały poddane popularne zbiory dotyczące problemu klasyfikacji, są to: „Kwiaty irysa" oraz tzw. problemy mnicha (zbiory danych „MONK"). Pierwszy z nich jest niewielkim i prostym zbiorem, pojawiającym się bardzo często w literaturze, pozostałe to zbiory syntetyczne. Dzięki temu, że znane są dla tych zbiorów zależności opisujące poszczególne klasy, pozwalają one ocenić badane metody pod kątem zdolności odkrywania wiedzy.

Następnie pokazano trzy studia przypadków dotyczące rzeczywistych zbiorów danych, gdzie każdy z nich dotyczy innego typu problemu. Z uwagi na objętość pracy w każdym z przykładów poświęcono szczególną uwagę tylko jednej z metod. I tak dla problemu klasyfikacji wybrano zbiór danych „Heart Disease" oraz algorytm ComplexConditions, dla regresji zbiór „Ozone" oraz DeepRules, z kolei dla analizy przeżycia zbiór „BMT-Ch" oraz MofNRules.

1 Zbiory danych o znanych zależnościach opisujących poszczególne klasy

1.1 Kwiaty irysa

Zbiór danych „Kwiaty irysa" (z ang. iris) to niezwykle popularny w literaturze zbiór benchmarkowy dotyczący problemu klasyfikacji trójklasowej. Każdy z jego 150 przykładów opisuje jeden kwiat z użyciem czterech atrybutów numerycznych:

długości i szerokości ich płatków oraz działek kielicha (*petallength*, *petalwidth*, *sepallength* i *sepalwidth*). Równoliczne klasy decyzyjne reprezentują tutaj trzy gatunki kwiatów irysa: *setosa*, *versicolor* i *virginica*. Przed przejściem do uczenia modeli zbiór podzielono na część treningową i testową, gdzie pierwsza z nich liczyła 135 przykładów.

Tabela 8.1 prezentuje zbiór wygenerowany algorytmem referencyjnym RuleKit zawierający 9 reguł. Osiągnął on BAcc na zbiorze testowym równe 0.933. Można tutaj zauważyć, że druga oraz trzecia reguła pokrywają bardzo niewiele przykładów.

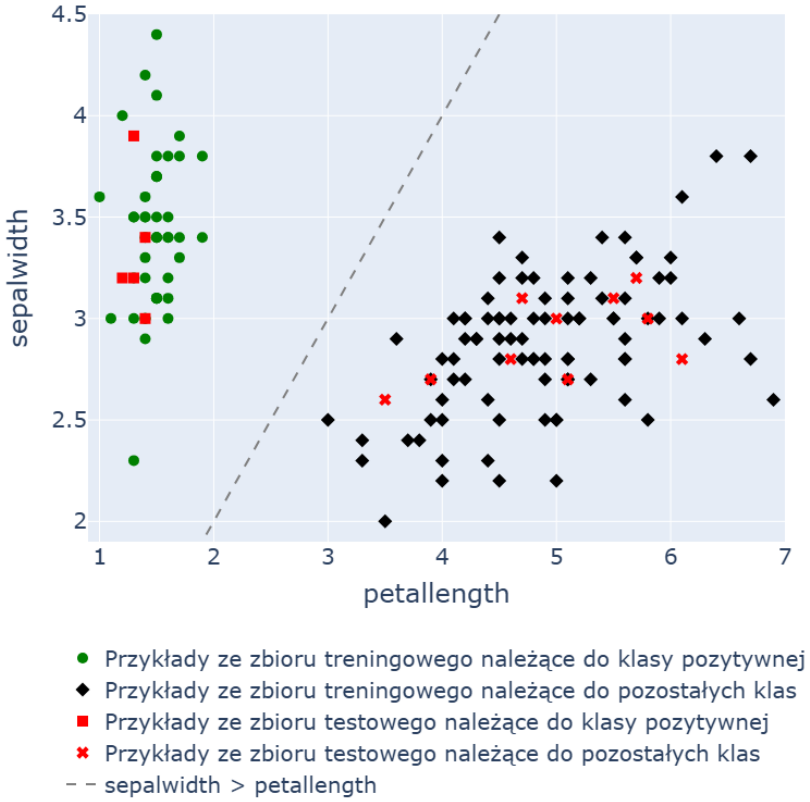
$\mathbf{r_{k1}}$	IF petallength $<$ 2.45 THEN class = setosa (p=45, n=0)
$\mathbf{r_{k2}}$	IF sepalwidth $\in$ [2.35, 2.45) THEN class = versicolor (p = 3, n = 0)
$\mathbf{r_{k3}}$	IF sepallength $>$ 6.1 AND sepalwidth $<$ 2.45 THEN class = versicolor (p = 2, n = 0)
$\mathbf{r_{k4}}$	IF petallength $\in$ [2.3, 4.75) THEN class = versicolor (p = 40, n = 1)
$\mathbf{r_{k5}}$	IF petalwidth $<$ 1.65 AND petallength $\in$ [2.2, 4.95) THEN class = versicolor (p = 43, n = 0)
$\mathbf{r_{k6}}$	IF petalwidth $<$ 1.85 AND sepallength $>$ 4.95 AND petallength $\in$ [2.45, 5.15) THEN class = versicolor (p = 44, n = 7)
$\mathbf{r_{k7}}$	IF petalwidth $>$ 1.65 AND petallength $>$ 4.85 THEN class = virginica (p = 38, n = 0)
$\mathbf{r_{k8}}$	IF petalwidth $>$ 1.65 THEN class = virginica (p = 41, n = 1)
$\mathbf{r_{k9}}$	IF petallength $>$ 4.95 THEN class = virginica (p = 39, n = 1)

Tabela 8.1: Reguły wygenerowane algorytmem RuleKit

W tabeli 8.2 widoczny jest z kolei zbiór reguł uzyskany metodą ComplexConditions. Zawiera ona o dwie reguły mniej niż tabela poprzednia. Każda z reguł cechuje się stosunkowo dużą liczbą pokrytych przykładów. Może to sugerować, że reprezentowane przez nie zależności lepiej opisują poszczególne klasy. Zbiór ten osiąga tę samą wartość BAcc wynoszącą 0.933.

$\mathbf{r_{c1}}$	IF sepalwidth $>$ petallength THEN class = setosa (p = 45, n = 0)
$\mathbf{r_{c2}}$	IF petalwidth $<$ 1.65 AND petallength $\in$ [2.45, 4.8) THEN class = versicolor
$\mathbf{r_{c3}}$	IF sepallength $>$ 4.95 AND petallength $\in$ [2.45, 4.9) THEN class = versicolor (p = 41, n = 2)
$\mathbf{r_{c4}}$	IF petalwidth $\in$ [0.8, 1.7) AND petallength $<$ 4.95 THEN class = versicolor (p = 43, n = 0)
$\mathbf{r_{c5}}$	IF petalwidth $>$ 1.65 AND petallength $>$ 4.85 THEN class = virginica (p = 38, n = 0)
$\mathbf{r_{c6}}$	IF petalwidth $>$ 1.65 THEN class = virginica (p = 41, n = 1)
$\mathbf{r_{c7}}$	IF petallength $>$ 4.95 THEN class = virginica (p = 39, n = 1)

Tabela 8.2: Reguły wygenerowane algorytmem ComplexConditions



Rysunek 8.1: Wizualizacja pokrycia reguły r_{c1} z tabeli 8.2. Linia przerywana reprezentuje przesłankę reguły.

Warto w tym miejscu zwrócić uwagę na pierwszą regułę r_{c1}. Jej konkluzja wskazuje na klasę *setosa*, będącą liniowo separowalną od pozostałych. Można ją opisać z użyciem reguły zawierającej pojedynczy warunek $petallenght < 2.45$. Reguła taka pojawia się w zbiorze wygenerowanym algorytmem RuleKit (r_{k1}). Co ciekawe jednak, algorytm ComplexConditions opisał tę klasę w inny, alternatywny sposób z użyciem warunku $sepalwidth > petallength$ (reguła r_{c1}). Na rysunku 8.1 podjęto próbę zwizualizowania pokrycia tegoż warunku. Zielone punkty oraz czerwone kwadraty odpowiadają kolejno przykładom pozytywnym (należącym do klasy *setosa*) zbioru treningowego i testowego. Czarne kwadraty oraz czerwone krzyżyki reprezentują kolejno negatywne przykłady treningowe i testowe.

Patrząc na rysunek 8.1 można zobaczyć, że przesłanka reguły r_{c1} idealnie oddziela klasę *setosa* od pozostałych. Na podstawie danych nie sposób stwierdzić wyższości takiego opisu nad tym reprezentowanym przez regułę r_{k1}. Pokazuje to jednak,

w jaki sposób wykorzystanie złożonych warunków w przesłankach może pomagać w odkrywaniu nowych zależności, które inaczej mogłyby zostać niezauważone.

W tabelach 8.4 i 8.3 zaprezentowano zbiory reguł wygenerowane metodą MofNRules kolejno w wariantach „przynajmniej M-z-N" oraz „dokładnie M-z-N". Obydwa osiągnęły na części testowej tę samą wartość BAcc wynoszącą, podobnie jak dla algorytmu ComplexConditions, 0.933.

$\mathbf{r_{md1}}$	IF sepalwidth > petallength THEN class = setosa (p = 45, n = 0)
$\mathbf{r_{md2}}$	IF 2-of-3 (petallength < 4.95, sepalwidth > petallength, petalwidth < 1.65) THEN class = versicolor (p = 43, n = 0)
$\mathbf{r_{md3}}$	IF sepalwidth < petallength THEN class = versicolor (p = 45, n = 45)
$\mathbf{r_{md4}}$	IF ¬ 2-of-3 (sepalwidth > petallength, petalwidth ∈ [0.80, 1.70), petallength < 4.95) THEN class = virginica (p = 45, n = 2)

Tabela 8.3: Reguły wygenerowane algorytmem MofNRules w wariancie „dokładnie M-z-N"

Zarówno w pierwszym, jak i drugim zbiorze uzyskanym metodą MofNRules, klasa *virginica* opisana jest pojedynczą regułą zawierającą warunek 2-z-3. Pokrywa ona wszystkie przykłady pozytywne oraz dwa negatywne. Wcześniej omawiane algorytmy potrzebowały kilku reguł do rozdzielenia przykładów tej klasy. Warunek M-z-N zawarty w regułach r_{mp5} oraz r_{md4} zawiera w sobie warunek złożony *sepalwidth* > *petallength* analizowany wcześniej na rysunku 8.1. Sugeruje to, że dzięki wykorzystaniu deskryptorów M-z-N algorytm MofNRules zdołał odkryć nową zależność, która stosunkowo precyzyjnie opisuje klasę *virginica*.

$\mathbf{r_{mp1}}$	IF sepalwidth > petallength THEN class = setosa (p = 45, n = 0)
$\mathbf{r_{mp2}}$	IF petallength ∈ [2.45, 4.80) AND petalwidth < 1.65 THEN class = versicolor (p = 40, n = 0)
$\mathbf{r_{mp3}}$	IF petallength ∈ [2.45, 4.90) AND 2-of-3 (petallength < 4.95, petalwidth < 1.65, sepallength > 4.95) THEN class = versicolor (p = 42, n = 2)
$\mathbf{r_{mp4}}$	IF petalwidth ∈ [0.80, 1.70) AND petallength < 4.95 THEN class = versicolor (p = 43, n = 0)
$\mathbf{r_{mp5}}$	IF ¬ 2-of-3 (petallength < 4.95, sepalwidth > petallength, petalwidth < 1.65) class = virginica (p = 45, n = 2)

Tabela 8.4: Reguły wygenerowane algorytmem MofNRules w wariancie „przynajmniej M-z-N"

Jako ostatni zostanie zaprezentowany zbiór reguł wygenerowany metodą DeepRules, widoczny w tabeli 8.5. Zawiera on 5 reguł, uzyskując przy tym najwyższe BAcc

na zbiorze testowym równe 0.944. Warto w tym miejscu zwrócić uwagę na regułę r_{d3}, opisującą klasę *versicolor*. Pokrywa ona wszystkie przykłady pozytywne i tylko jeden negatywny. Inne metody nie uzyskały dla tej klasy reguły o równie wysokim wsparciu i precyzji.

$\mathbf{r_{d1}}$	IF petallength $\leqslant$ 2.45 THEN class = setosa (p=45, n=0)
$\mathbf{r_{d2}}$	IF (petalwidth $\leqslant$ 1.6 $\vee$ petallength $\leqslant$ 4.8) $\wedge$ (petalwidth $\leqslant$ 1.6 $\vee$ sepalwidth $\geqslant$ 3.05) $\wedge$ (petallength $\leqslant$ 4.85 $\vee$ sepalwidth $\leqslant$ 2.5) $\wedge$ petallength $\in$ [2.45, 4.9] THEN class = versicolor (p=43, n=0)
$\mathbf{r_{d3}}$	IF petallength $\in$ [2.45, 5.1] $\wedge$ (sepallength $\leqslant$ 6.10 $\vee$ petallength $\leqslant$ 4.9) $\wedge$ (petalwidth $\leqslant$ 1.6 $\vee$ sepalwidth $\geqslant$ 3.05) $\wedge$ (petalwidth $\leqslant$ 1.65 $\vee$ sepalwidth $\geqslant$ 2.8) THEN class = versicolor (p=45, n=1)
$\mathbf{r_{d4}}$	IF petalwidth $\geqslant$ 1.8 THEN class = virginica (p=40, n=1)
$\mathbf{r_{d5}}$	IF petallength $\geqslant$ 4.85 THEN class = virginica (p=42, n=3)

Tabela 8.5: Reguły wygenerowane algorytmem DeepRules

Metoda DeepRules, jak zostało to opisane w rozdziale 6 w sekcji 3, jest teoretycznie w stanie znajdować wyrażenia DNF odpowiadające warunkom M-z-N. W tym wypadku jednak tak się nie stało i żadna z reguł w zbiorze nie może zostać zapisana w taki sposób. Należy pamiętać, że wszystkie opisywane tutaj metody, w tym DeepRules, bazują na heurystykach. Tym samym nie przeszukują one pełnej przestrzeni możliwych rozwiązań, przez co niektóre z nich nie są brane pod uwagę. Z tego powodu, pomimo pewnych podobieństw, mogą one prowadzić do uzyskania całkowicie różnych zbiorów reguł.

1.2 Problemy „MONK”

Tak zwane Problemy „MONK” (z ang. MONK’s Problems) to nazwa trzech popularnych syntetycznych zbiorów benchmarkowych, dotyczących klasyfikacji binarnej. Są one powszechnie używane do testowania różnych algorytmów uczenia maszynowego i odkrywania wiedzy. Każdy z nich zawiera sześć atrybutów nominalnych ($a1, \ldots, a6$). Celem analizy jest odkrycie zależności opisującej klasę pozytywną. Jest ona zdefiniowana następująco dla każdego z trzech zbiorów [186]:

- Dla MONK-1: klasa = 1, jeśli $a1 = a2 \vee a5 = 1$.
- Dla MONK-2: klasa = 1, jeśli dokładnie 2 z atrybutów $a1, \ldots, a6$ są równe 1.
- Dla MONK-3: klasa = 1, jeśli (a5 = 3 $\wedge$ a4 = 1) $\vee$ (a5 $\neq$ 4 $\wedge$ a2 $\neq$ 3).

W zbiorze MONK-3, dla atrybutu decyzyjnego, zostało dodatkowo dodane 5% szumu utrudniającego klasyfikację.

MONK-1

Zbiór danych MONK-1 zawiera 324 przykłady, z czego 124 stanowią część treningową (w tym 62 należące do klasy pozytywnej), a pozostałe testową.

W tabeli 8.6 zostały przedstawione reguły wskazujące na klasę pozytywną wygenerowane przez referencyjny algorytm RuleKit. Jak widać, nie zdołał on odkryć oryginalnej zależności, która ją opisuje. Pomimo, że zbiór ten nie zawiera szumu, uzyskane reguły nie osiągnęły na danych testowych BAcc równego 1, co pokazuje, że nie wszystkie przykłady zostały poprawnie sklasyfikowane.

$\mathbf{r_{k1}}$	IF a5 = 1 THEN klasa = 1 (p = 29, n = 0)
$\mathbf{r_{k2}}$	IF a1 = 3 AND a2 = 3 THEN klasa = 1 (p = 17, n = 0)
$\mathbf{r_{k3}}$	IF a3 = 2 AND a4 = 1 AND a6 = 1 THEN klasa = 1 (p = 7, n = 1)
$\mathbf{r_{k4}}$	IF a1 = 2 AND a2 = 2 THEN klasa = 1 (p = 15, n = 0)
$\mathbf{r_{k5}}$	IF a1 = 1 AND a2 = 1 THEN klasa = 1 (p = 9, n = 0)

Tabela 8.6: Reguły wygenerowane algorytmem RuleKit, na zbiorze „MONK-1”

Przejdźmy teraz do analizy reguł uzyskanych z użyciem zaproponowanych przez autora metod. Na początek zostanie omówiony zbiór wygenerowany przez ComplexConditions, widoczny w tabeli 8.7. Zawiera on jedynie dwie reguły osiągające BAcc równe 1.0. Patrząc na ich przesłanki, widać, że algorytm odnalazł zależność opisującą klasę pozytywną. Zawiera ona w sobie jednak operator logicznej alternatywy, którego metoda ComplexConditions nie jest w stanie zastosować w tym konkretnym kontekście. Z tego powodu do opisu klasy pozytywnej potrzebne były dwie reguły.

Zbiory uzyskane metodą MofNRules były w tym wypadku identyczne jak te zaprezentowane w tabeli 8.7, co jest zgodne z oczekiwaniami. Dla tego konkretnego zbioru danych, biorąc pod uwagę zależność, jaką opisuje, użycie warunków M-z-N nie mogło przynieść korzyści ani pod kątem dokładności predykcji, ani liczby reguł.

$\mathbf{r_{c1}}$	IF a1 = a2 THEN klasa = 1 (p = 41, n = 0)
$\mathbf{r_{c2}}$	IF a5 = 1 THEN klasa = 1 (p = 29, n = 0)

Tabela 8.7: Reguły wygenerowane algorytmem ComplexConditions, na zbiorze „MONK-1"

Na koniec zostanie pokazany zbiór reguł wygenerowany algorytmem DeepRules. Ten z kolei, w przeciwieństwie do ComplexConditions i MofNRules, jest w stanie uzyskiwać reguły, gdzie warunki w przesłance łączone są operatorem alternatywy logicznej. Można więc spodziewać się, że zdoła on opisać klasę pozytywną z użyciem jednej reguły. Patrząc na tabelę 8.8 można zobaczyć, że faktycznie tak się stało.

$\mathbf{r_{d1}}$	IF a1 = a2 $\vee$ a5 = 1 THEN klasa = 1 (p = 62, n = 0)

Tabela 8.8: Reguła wygenerowana przez algorytm DeepRules, na zbiorze „MONK-1"

MONK-2

Zbiór danych MONK-2 posiada 369 przykładów. Spośród nich 169 stanowi część treningową, zawierającą 105 przykładów należących do klasy pozytywnej. Ta z kolei opisana jest pojedynczym warunkiem dokładnie 2-z-6.

Zbiór reguł wygenerowany z użyciem algorytmu referencyjnego RuleKit zawierał aż 27 reguł wskazujących na klasę pozytywną. Ze względu na ich liczbę nie zostały one przedstawione w tej pracy. Uzyskany zbiór osiągnął BAcc na danych testowych równe 0.698. Wiele spośród tych reguł posiadało bardzo niskie wsparcie, a aż 14 z nich pokrywało mniej niż 3 przykłady. Cały zbiór uzyskał stosunkowo wysokie BAcc na danych treningowych, wynoszące 0.921, które spadło jednak drastycznie w przypadku danych testowych. Wszystko to sugeruje, że algorytm RuleKit, używający w przesłankach reguł jedynie warunki proste, nie zdołał w sposób precyzyjny opisać klasy pozytywnej.

$\mathbf{r_{mp1}}$	IF $\neg$ 2-of-6 (a1 = 1, a2 = 1, a3 = 2, a4 = 1, a5 = 1, a6 = 1) THEN class = 1 (p = 33, n = 17)
$\mathbf{r_{mp2}}$	IF $\neg$ 2-of-6 (a1 = 1, a2 = 1, a3 = 1, a4 = 1, a5 = 1, a6 = 2) THEN class = 1 (p = 28, n = 20)
$\mathbf{r_{mp3}}$	IF 2-of-6 (a1 = 1, a2 = 1, a3 = 1, a4 = 1, a5 = 1, a6 = 1) THEN class = 1 (p = 64, n = 64)

Tabela 8.9: Reguły wygenerowane algorytmem MofNRules w wariancie przynajmniej 2-z-6, na zbiorze „MONK-2"

Metoda ComplexConditions wygenerowała w tym przypadku mniej, bo jedynie 21 reguł. Ze względu na objętość pracy one również nie zostaną w niej przedstawione. Algorytm ten osiągnął wyższe, choć wciąż niezbyt wysokie BAcc na danych testowych równe 0.819. Poszczególne reguły cechowały się także wyższym wsparciem niż te wygenerowane przez RuleKit. Wciąż jednak były wśród nich reguły, pokrywające jedynie kilka przykładów. Wskazuje to, że zastosowanie warunków złożonych w przesłankach reguł, przez algorytm ComplexConditions, pozwala na uzyskanie bardziej dokładnego i zwięzłego opisu danych. Wciąż jednak nie zdołano odkryć oryginalnej zależności reprezentującej klasę pozytywną.

W kolejnym kroku wygenerowano zbióry reguł z użyciem metody MofNRules. Zostały one przedstawione w tabelach 8.9 (wariant „przynajmniej M-z-N") oraz 8.10 (wariant „dokładnie M-z-N"). Rezultat jest tutaj zgodny z oczekiwaniami. Wariant „dokładnie M-z-N", dla M równego 2 i N równego 6, odkrył oryginalną zależność opisującą klasę pozytywną. Reprezentująca ją reguła r_{md1} pokrywa wszystkie należące do niej przykłady, uzyskując na danych testowych BAcc równe 1. Co ciekawe, dla klasy negatywnej została wygenerowana analogiczna reguła zawierająca ten sam warunek M-z-N w zanegowanej formie.

BAcc na danych testowych, dla wariantu przynajmniej 2-z-6, wyniosło 0.764, a więc mniej niż w przypadku ComplexConditions. Liczba reguł jest tutaj jednak znacznie mniejsza (3). W przypadku obydwu zbiorów uzyskanych z użyciem MofNRules, wszystkie występujące w nich reguły posiadały stosunkowo wysokie wsparcie.

$\mathbf{r_{md1}}$	IF 2-of-6 (a1 = 1, a2 = 1, a3 = 1, a4 = 1, a5 = 1, a6 = 1) THEN class = 1 (p = 64, n = 0)

Tabela 8.10: Reguła wygenerowana przez algorytm MofNRules w wariancie dokładnie 2-z-6, na zbiorze „MONK-2"

Przykład ten pokazuje, jak wybór odpowiedniego wariantu metody MofNRules potrafi warunkować liczbę oraz jakość indukowanych reguł. Widać tutaj również, jak warunki M-z-N mogą pomóc w uzyskiwaniu krótszych i mniej złożonych opisów, a także w odkrywaniu zależności trudnych do przedstawienia z użyciem innych rodzajów deskryptorów.

Algorytm DeepRules wygenerował dla zbioru MONK-2 aż 7 reguł. Pomimo że warunek dokładnie 2-z-6 zawarty w regule r_{md1} można zapisać w postaci formuły DNF, która mogłaby teoretycznie zostać przez niego odkryta, tak się nie stało. Żadna z uzyskanych z jego pomocą reguł nie zawierała oryginalnej zależności opisującej

klasę pozytywną. Sam zbiór osiągnął z kolei na danych testowych BAcc wynoszący 0.82. Z uwagi na poziom skomplikowania uzyskanych reguł, w tabeli 8.11 zostanie zaprezentowana tylko jedna z nich. Jak można zauważyć, jej przesłanka jest na tyle rozbudowana, że jej zrozumienie staje się bardzo trudne.

$\mathbf{r_{d1}}$	IF $(a3 \neq a6 \vee a2 = a4 \vee a5 = 1) \wedge (a1 \neq a2 \vee a1 = 2 \vee a5 = 1) \wedge (a4 \neq 1 \vee a5 \neq 1) \wedge (a1 = 2, 3 \vee a5 \neq 1) \wedge (a1 \neq a2 \vee a5 \neq 2) \wedge (a1 \neq a4 \vee a5 \neq 4) \wedge (a1 = 2, 3 \vee a1 \neq a2) \wedge (a3 \neq a6 \vee a1 \neq a2) \wedge (a1 \neq a4 \vee a1 \neq 1) \wedge (a3 \neq a6 \vee a4 \neq 2) \wedge (a3 \neq a6 \vee a5 \neq 4) \wedge (a3 \neq a6 \vee a5 \neq 3 \vee a4 = 1) \wedge (a2 = a4 \vee a5 \neq 1 \vee a3 = 2) \wedge (a1 \neq a2 \vee a5 \neq 3) \wedge (a4 \neq 1 \vee a2 \neq 1 \vee a6 = 2) \wedge (a1 \neq a2 \vee a5 = 1 \vee a4 = 1) \wedge (a3 \neq a6 \vee a2 \neq 2)$ THEN klasa = 1 (p=48, n=7)

Tabela 8.11: Przykładowa reguła wygenerowana przez algorytm DeepRules, na zbiorze „MONK-2”

Na przykładzie zbioru MONK-2 możemy zauważyć, że zdolność algorytmu DeepRules do generowania reguł zawierających złożone formuły logiczne CNF i DNF w przesłankach może w pewnych sytuacjach prowadzić do uzyskania nadmiernie długich i trudnych w interpretacji reguł.

MONK-3

Ostatni ze zbiorów, opisujący trzeci problem „MONK”, zawiera 554 przykłady, z których 122 stanowi zbiór treningowy, gdzie 60 z nich należy do klasy pozytywnej i 62 do negatywnej. W przeciwieństwie do dwóch poprzednich, zbiór ten posiada 5-procentowy szum dodany dla atrybutu decyzyjnego, który dodatkowo utrudnia precyzyjną klasyfikację.

Zbiór reguł wygenerowany algorytmem RuleKit zawierał w tym wypadku 13 reguł, osiągając na danych testowych BAcc równe 0.913. Tak jak w poprzednim przykładzie, tutaj również część z nich posiadała stosunkowo niskie wsparcie, a trzy pokrywały mniej niż 10 przykładów.

Zbiór uzyskany algorytmem ComplexConditions zaprezentowano w tabeli 8.12. Zawiera on jedynie dwie reguły, osiągając BAcc równe 0.991. Na tym przykładzie można zauważyć, jak wykorzystanie warunków wewnętrznej alternatywy dla atrybutów nominalnych skutkuje krótszym i bardziej zwięzłym opisem danych, zwiększając przy tym jakość klasyfikacji. Każda z reguł w zbiorze cechuje się także stosunkowo wysoką wartością wsparcia.

$\mathbf{r_{c1}}$	IF a2 ∈ 1, 2 AND a5 ∈ {1, 2, 3} THEN klasa = 1 (p = 57, n = 5)
$\mathbf{r_{c2}}$	IF a1 ≠ a4 AND a5 ∈ {1, 2, 3} THEN klasa = 1 (p = 42, n = 16)

Tabela 8.12: Reguły wygenerowane algorytmem ComplexConditions, na zbiorze „MONK-3"

Zbiory reguł uzyskane metodą MofNRules, dla M równego 2 i N równego 3, widoczne są w tabelach 8.13 (wariant „przynajmniej M-z-N") i 8.14 (wariant „dokładnie M-z-N"). Obydwa uzyskały BAcc na zbiorze testowym równe 0.974. Wszystkie obecne w nich reguły posiadają stosunkowo wysokie wsparcie.

$\mathbf{r_{mp1}}$	IF a2 ≠ 3 AND a5 ≠ 4 THEN klasa = 1 (p = 57, n = 5)
$\mathbf{r_{mp2}}$	IF a1 ≠ a4 AND ¬ 2-of-3 (a5 = 4, a2 = 3, a3 ≠ 2) THEN klasa = 1 (p = 42, n = 16)

Tabela 8.13: Reguły wygenerowane algorytmem MofNRules w wariancie „przynajmniej M-z-N", na zbiorze „MONK-3"

$\mathbf{r_{md1}}$	IF a2 ≠ 3 AND a5 ≠ 4 THEN klasa = 1 (p = 57, n = 5)
$\mathbf{r_{md2}}$	IF a5 ≠ 4 AND a1 ≠ a4 AND ¬ 2-of-3 (a5 = 4, a2 = 3, a3 ≠ 2) THEN klasa = 1 (p = 42, n = 7)

Tabela 8.14: Reguły wygenerowane algorytmem MofNRules w wariancie „dokładnie M-z-N", na zbiorze „MONK-3"

Patrząc na reguły wygenerowane metodą ComplexConditions, możemy zauważyć, że pierwsza reguła r_{c1} odpowiada drugiej części wyrażenia opisującego klasę pozytywną zbioru MONK-3 (a5 ≠ 4 ∧ a2 ≠ 3). Jest tak, ponieważ domeną wartości atrybutu a2 jest zbiór {1, 2, 3}, a atrybutu a5 zbiór {1, 2, 3, 4}. Druga reguła r_{c2} z tabeli 8.12 nie odpowiada jednak oryginalnemu opisowi klasy pozytywnej. Podobna sytuacja występuje w przypadku zbiorów uzyskanych metodą MofNRules. Przesłanka pierwszej reguły jest tutaj alternatywnym, krótszym zapisem drugiej części formuły opisującej klasę pozytywną. Pierwsza część tej formuły (a5 = 3 ∧ a4 = 1) pozostała jednak nieodkryta przez żaden z wyżej wspominanych algorytmów.

$\mathbf{r_{md1}}$	IF (a2 ≠ 3 ∨ a5 = 3) ∧ a5 ≠ 4 AND (a2 ≠ 3 ∨ a4 = 1) THEN klasa = True (p=59, n=5)

Tabela 8.15: Reguła wygenerowana przez algorytm DeepRules, na zbiorze „MONK-3"

Na koniec został zaprezentowany jednoelementowy zbiór reguł wygenerowany z użyciem algorytmu DeepRules, osiągający na danych testowych BAcc równe 1 (patrz tabela 8.15). Przypadek ten jest szczególnie ciekawy, ponieważ reguła ta nie odpowiada oryginalnemu wyrażeniu definiującemu klasę pozytywną. Opisuje ją jednak z równą skutecznością. W celach dalszej analizy utworzono regułę r', opartą na oryginalnym opisie problemu MONK-3, oraz obliczono jej pokrycie. Została ona przedstawiona poniżej.

$\mathbf{r'}$: IF (a5 = 3 ∧ a4=1) ∨ (a5 ≠ 4 ∧ a2 ≠ 3) THEN klasa =1 (p=59, n=5)

Jak można zaobserwować, pokrycie to jest identyczne jak w przypadku reguły r_{d1} uzyskanej algorytmem DeepRules. Zweryfikowano także, że obydwie reguły pokrywają dokładnie ten sam zestaw przykładów zarówno pozytywnych, jak i negatywnych. Poniżej zostało dowiedzione, że przesłanki obydwu reguł r' i r_{d1} są tożsame logicznie.

Stwierdzenie 1. *Formuły*

- $F_1 = (a5 = 3 \wedge a4 = 1) \vee (a5 \neq 4 \wedge a2 \neq 3)$ *i*
- $F_2 (a2 \neq 3 \vee a5 = 3) \wedge a5 \neq 4 \wedge (a2 \neq 3 \vee a4 = 1)$

są tożsame logicznie, zakładając, że $a2 \in \{1, 2, 3\}$, $a4 \in \{1, 2, 3\}$ *oraz* $a5 \in \{1, 2, 3, 4\}$.

Dowód. Aby udowodnić prawdziwość stwierdzenia 1 zostanie dokonane przekształcenie formuły F_2 do postaci F_1 z użyciem praw rachunku zdań.

W pierwszym kroku, dla F_2, zostanie zastosowane prawo rozdzielności alternatywy względem koniunkcji: $(A \vee B) \wedge (A \vee C) \Leftrightarrow A \vee (B \wedge C)$, gdzie:

- $A = (a_2 \neq 3)$
- $B = (a5 = 3)$
- $C = (a4 = 1)$

Po jego użyciu otrzymujemy:

$$F_2 = [a2 \neq 3 \vee (a5 = 3 \wedge a4 = 1)] \wedge a5 \neq 4 \tag{8.1}$$

Następnie wykorzystane zostanie prawo rozdzielności koniunkcji względem alternatywy: $(A \vee B) \wedge C \Leftrightarrow (A \wedge C) \vee (B \wedge C)$, gdzie:

- $A = (a_2 \neq 3)$
- $B = (a5 = 3) \wedge (a4 = 1)$
- $C = (a5 \neq 4)$

Otrzymujemy w ten sposób następującą postać:

$$F_2 = (a2 \neq 3 \wedge a5 \neq 4) \vee (a5 = 3 \wedge a4 = 1 \wedge a5 \neq 4) \tag{8.2}$$

Przyjrzyjmy się teraz prawemu członowi powyższej alternatywy:

$$(a5 = 3 \wedge a4 = 1 \wedge a5 \neq 4)$$

Atrybut a5 przyjmuje wartości z domeny $\{1, 2, 3, 4\}$, przez co warunek $a5 = 3$ implikuje tutaj $a5 \neq 4$. Dlatego też koniunkcję $(a5 = 3 \wedge a5 \neq 4)$ można w tym wypadku uprościć do samego $a5 = 3$. W ten sposób prawy człon alternatywy sprowadza się do poniższej postaci $(a5 = 3 \wedge a4 = 1)$.

Podstawiając uproszczoną postać prawego członu alternatywy do aktualnej postaci formuły F_2 otrzymujemy:

$$F_2 = (a2 \neq 3 \wedge a5 \neq 4) \vee (a5 = 3 \wedge a4 = 1) \tag{8.3}$$

Zamieniając kolejność literatów alternatywy oraz koniunkcji, korzystając z praw przemienności, otrzymujemy w końcu:

$$F_2 = (a5 = 3 \wedge a4 = 1) \vee (a5 \neq 4 \wedge a2 \neq 3) \tag{8.4}$$

Uzyskana w ten sposób formuła F_2 jest równa formule F_1, co było do udowodnienia.

□

Można więc powiedzieć, że algorytm DeepRules, jako jedyny w badanych metodach, odkrył dla zbioru MONK-3 zależność opisującą klasę pozytywną. Choć jej zapis nie jest identyczny z oczekiwanym, to jak dowiedziono, obie formuły są tożsame logicznie.

2 Zbiór danych dotyczący choroby wieńcowej „Heart Disease”

Zbiór danych „Heart Disease” został opisany po raz pierwszy przez Detrano i innych w pracy z 1989 roku zatytułowanej „International application of a new probability

algorithm for the diagnosis of coronary artery disease". Dane zostały zebrane przez Centrum Medyczne Weteranów, Long Beach oraz Fundację Kliniki Cleveland. Dotyczą one problemu klasyfikacji dwuklasowej, gdzie atrybut decyzyjny określa wystąpienie choroby serca, a poszczególne przykłady w zbiorze reprezentują pacjentów. Pierwotnie zbiór ten posiadał 303 wiersze i aż 76 atrybutów, jednak w większości badań, jak i w tym przykładzie, wykorzystywano tylko niektóre z nich. Są to kolejno:

- **age**: Wiek pacjenta (atrybut numeryczny).
- **sex**: Płeć pacjenta (atrybut binarny: 0 = kobieta, 1 = mężczyzna).
- **cp**: Typ bólu w klatce piersiowej, atrybut kategoryczny z czterema wartościami:
  - *asympt* - dławica asymptomatyczna,
  - *non_anginal* - ból niedławicowy,
  - *typ_angina* - typowa dławica piersiowa,
  - *atyp_angina* - dławica atypowa;
- **trestbps**: Spoczynkowe ciśnienie krwi (atrybut numeryczny).
- **chol**: Poziom cholesterolu w surowicy (atrybut numeryczny).
- **thalach**: Maksymalne tętno osiągnięte podczas wysiłku (atrybut numeryczny).
- **exang**: Dławica piersiowa wywołana wysiłkiem (atrybut binarny: 0 = brak, 1 = obecność).
- **oldpeak**: Obniżenie odcinka ST wywołane wysiłkiem fizycznym (atrybut numeryczny).
- **ca**: Liczba zwężonych lub zablokowanych głównych naczyń krwionośnych (atrybut kategoryczny: 0-3).
- **thal**: Wynik testu na talasemię, atrybut kategoryczny z trzema wartościami:
  - *reversable_defect* - defekt odwracalny,
  - *fixed_defect* - defekt uleczony,
  - *normal* - wynik ujemny;

Tabela 8.16: Wyniki badanych metod ocenianych na części testowej zbioru danych „Heart Disease”

Metoda	BAcc (test)	Liczba reguł	Średnia liczba warunków w regule	Sumaryczna liczba warunków	Średnia jakość
Drzewa decyzyjne	0.594	7	3.571	25	–
RuleKit	0.953	52	4.308	224	0.614
ComplexConditions	0.943	34	3.882	132	0.619
MofNRules („dokładnie M-z-N”)	0.882	62	2.806	174	0.484
MofNRules („przynajmniej M-z-N”)	0.919	63	3.429	216	0.493
DeepRules	0.882	18	8.278	149	0.597

- **num**: Atrybut decyzyjny określający stan choroby wieńcowej przyjmujący dwie wartości: 0 = zwężenie tętnic wieńcowych poniżej 50%, 1 = zwężenie tętnic wieńcowych o 50% lub więcej sugerujące chorobę wieńcową.

Na potrzeby poniższego przykładu zbiór danych został podzielony na część treningową i testową w proporcjach około 70% i 30%. Podziału dokonano w sposób stratyfikowany względem atrybutu decyzyjnego w celu osiągnięcia możliwie podobnego rozkładu jego wartości w obydwu podzbiorach. Następnie wygenerowano dla niego zbiory reguł z użyciem wszystkich metod zaproponowanych przez autora w rozdziale 6 oraz referencyjnych algorytmów: drzew decyzyjnych i RuleKit. Użyto tutaj tych samych wartości parametrów jak podczas wcześniejszych eksperymentów (patrz rozdział 7 sekcja 3). Wygenerowane zbiory reguł zostały następnie ocenione pod kątem jakości klasyfikacji (wskaźnik BAcc), liczby reguł, średniej liczby warunków w regule, sumarycznej liczby warunków w całym zbiorze oraz średniej jakości reguły. Dla spójności i możliwości porównania, jakość reguł była tutaj obliczona z wykorzystaniem jednej miary C2 (patrz wzór 6.2) dla wszystkich metod, na podstawie pokrycia na zbiorze treningowym. Metryka ta została jednak pominięta dla drzew decyzyjnych, ze względu na ich odmienny sposób działania oraz użycie innych miar jakości w czasie procesu uczenia (w tym wypadku indeksu Giniego).

Wyniki dla poszczególnych metod przedstawiono w tabeli 8.16. Na jej podstawie można zauważyć, że najlepsze zdolności klasyfikacyjne osiągnął tutaj algorytm referencyjny RuleKit (0.953). Najgorszy wynik BAcc na zbiorze testowym osiągnęły drzewa decyzyjne, pomimo najmniejszej złożoności modelu (liczby reguł i warunków). Z zaproponowanych w pracy metod najniższymi wynikami predykcyjnymi wykazały się DeepRules oraz MofNRules w wariancie „dokładnie M-z-N” (BAcc równe 0.882).

Najlepszą z proponowanych metod okazał się algorytm ComplexConditions z BAcc wynoszącym 0.943. Osiągnął on również drugą najniższą (po drzewach decyzyjnych) sumaryczną liczbę warunków w zbiorze, wynoszącą 132. Jest to redukcja o około 40% względem zbioru reguł uzyskanego z użyciem RuleKit.

Z wyżej wspomnianych powodów oraz z uwagi na obszerność niniejszej pracy, w ramach tego przykładu uwagę skupiono głównie na metodzie ComplexConditions. Na jej przykładzie autor podejmie próbę zaprezentowania, w jaki sposób użycie złożonych warunków może prowadzić do zwiększenia mocy opisowej uzyskiwanych reguł, prowadząc do odkrywania nowych zależności w danych. Pokazane zostanie także, jak w pewnych sytuacjach może wiązać się to z uzyskiwaniem mniej złożonych, a zatem potencjalnie łatwiejszych w interpretacji zbiorów reguł. W tym celu zamiast prezentować pełne zbiory wygenerowane przez każdą z metod, wybrana zostanie pojedyncza reguła uzyskana metodą ComplexConditions, cechująca się stosunkowo wysoką wartością miary jakości C2 (0.703). Reguła ta, oznaczona jako r_c, została przedstawiona poniżej.

$$\mathbf{r_c}: \quad \text{IF } (\text{oldpeak} \geqslant 0.55 \ \vee \ \text{oldpeak} < 0.15) \ \wedge \ \text{thal} \neq \text{normal} \ \wedge \ \text{cp} = \text{asympt} \ \wedge \\ \text{age} < 65.5 \ \wedge \ \text{trestbps} \geqslant 112 \ \textbf{THEN} \ \text{class} = 1 \ (p = 51, n = 2)$$

Pokrywa ona 51 przykładów klasy pozytywnej, co stanowi ponad połowę jej liczebności, oraz 2 przykłady klasy negatywnej liczącej 97 przykładów. Klasa negatywna liczyła z kolei 115 wierszy. Wartości te sugerują, że reguła r_c może reprezentować silną relację w danych.

Patrząc na jej przesłankę można zauważyć, że zawiera ona w sobie różne rodzaje warunków. Pierwszy z nich (oldpeak $\geqslant 0.55 \vee$ oldpeak < 0.15) jest warunkiem wewnętrznej alternatywy opartym na atrybucie opisującym obniżenie odcinka ST, wywołane wysiłkiem fizycznym. Warunek taki spełniony jest zarówno dla wartości atrybutu poniżej 0.15, jak i dla takich większych lub równych 0.55. Sugeruje to, że nie tylko wysokie wartości obniżenia, ale również te niższe mogą być niepokojące i sugerować wystąpienie choroby wieńcowej u pacjenta. Drugi warunek (thal $\neq$ *normal*) stanowi negację warunku prostego opartego na atrybucie thal, reprezentującym wynik testu na talasemię. Kolejny warunek (cp = *asympt*) jest z kolei warunkiem prostym. Atrybut cp opisuje tutaj typ bólu w klatce piersiowej. Kolejne dwa warunki również są warunkami prostymi, gdzie pierwszy z nich (age < 65.5) dotyczy kolumny reprezentującej wiek pacjenta, a drugi (trestbps $\geqslant 112$) spoczynkowego ciśnienia krwi.

Na tej podstawie regułę r_c można więc całościowo zinterpretować w następujący sposób: wystarczająco niskie lub wysokie obniżenie odcinka ST u pacjenta z wiekiem poniżej 65.5 lat, w połączeniu z asymptotycznym bólem w klatce, wysokim ciśnieniem spoczynkowym oraz pozytywnym wynikiem testu na talasemię, może sugerować chorobę wieńcową.

Zasadność owej reguły pod kątem medycznym należałoby oczywiście zweryfikować z ekspertem dziedzinowym. Autor niniejszej pracy, ze względu na brak odpowiednich kompetencji, nie odważy się dokonać takiej oceny. Przykład ten służy jednak zaprezentowaniu zdolności metody ComplexConditions do wykrywania bardziej złożonych relacji w danych.

Sprawdzono również, czy referencyjny algorytm RuleKit odkrył te zależności. Znaleziono w tym celu, w wygenerowanym przez niego zbiorze, reguły zawierające warunki zbliżone do tych pojawiających się w regule r_c. Zostały one przedstawione poniżej.

$\mathbf{r_{k1}}$: **IF** oldpeak $\geqslant 2.45 \wedge$ cp = asympt $\wedge$ trestbps $\geqslant 107$ **THEN** class $= 1 (p = 27, n = 0)$

$\mathbf{r_{k2}}$: **IF** thal = reversable_defect $\wedge$ cp = asympt $\wedge$ oldpeak $\geqslant 0.65$ **THEN** class $= 1 (p = 45, n = 0)$

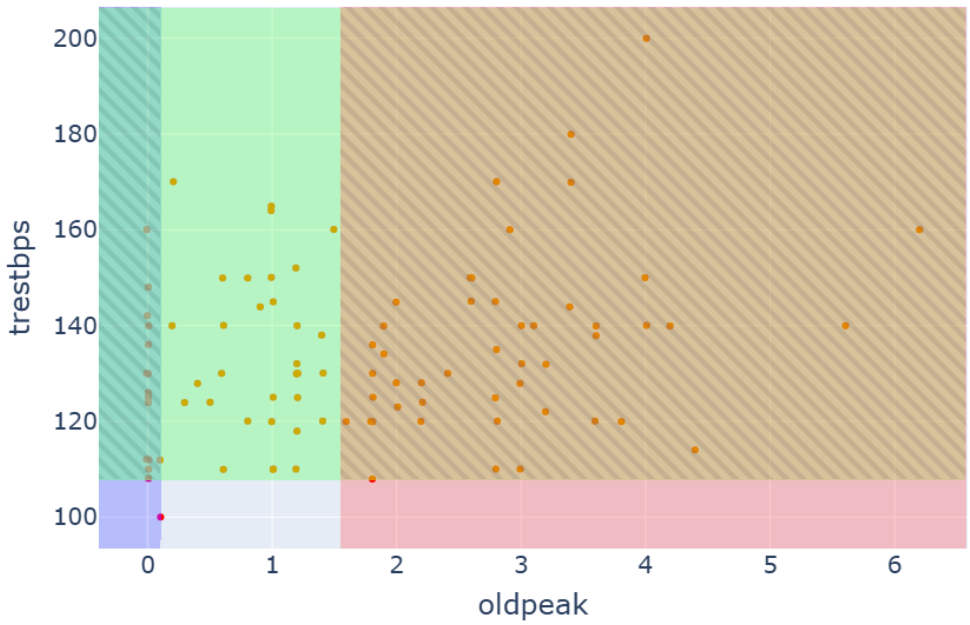
$\mathbf{r_{k3}}$: **IF** thal = reversable_defect $\wedge$ cp = asympt $\wedge$ trestbps $\geqslant 113.5 \wedge$ age < 63.5 **THEN** class $= 1 (p = 44, n = 3)$

Pierwsza z nich definiuje podobny próg dla atrybutu trestbps (107), zawierając również w swej przesłance warunek $cp = asymp$. Ostatni z warunków oparty o ciśnienie spoczynkowe jest jednak całkowicie inny. Zarówno w regule r_{k2}, jak i r_{k3}, występuje deskryptor $thal = reversable\_defect$. W ostatniej z nich obecny jest również warunek $age < 63.5$, będący zbliżonym do warunku wykorzystującego ten sam atrybut w regule r_c.

Reguły r_{k1}, r_{k2} i r_{k3} pomimo, że zawierają podobne deskryptory jak w przypadku r_c, nie opisują tej samej relacji w danych. Pokrywają one także mniejszą liczbę przykładów pozytywnych (kolejno 27, 45 i 44). Świadczy to o tym, że algorytm ComplexConditions wykrył potencjalnie nową zależność, opisującą klasę pozytywną.

Aby zwizualizować pokrycie reguły r_c, bazującej na aż pięciu atrybutach, zdecydowano się na uproszczenie. Skupiono się wyłącznie na dwóch atrybutach: oldpeak oraz trestbps. W tym celu usunięto z reguły r_c wszystkie warunki, które nie są na nich oparte. Uzyskana w ten sposób reguła $r_{c'}$ jest widoczna poniżej. Poszczególne warunki składowe zaznaczono dodatkowo kolorami, które zostały następnie wykorzystane do wizualizacji.

$\mathbf{r_{c'}}$: IF (oldpeak $\geqslant 0.55 \vee$ oldpeak < 0.15) $\wedge$ trestbps $\geqslant 112$ **THEN** class $= 1$ $(p = 82, n = 82)$



Rysunek 8.2: Wykres punktowy przedstawiający przykłady klasy pozytywnej zbioru „Heart Disease" z zaznaczonym na nim pokryciem reguły $r_{c'}$.

Jak widać po pokryciu, usunięcie pozostałych warunków znacząco pogorszyło jakość reguły. Na potrzeby poniższej wizualizacji skupiono się jednak na graficznym pokazaniu obszaru pokrywanego przez regułę rc'.

Rysunek 8.2 przedstawia wykres punktowy przykładów należących do klasy pozytywnej zbioru „Heart Disease". Na osi x znajdują się wartości atrybutu oldpeak a na y atrybutu trestbps. Pokrycia poszczególnych warunków składowych reguły zostały zaznaczone kolorowymi prostokątami. Ich kolory odpowiadają fragmentom przesłanki reguły $r_{c'}$ o tym samym kolorze. Obszar zakreskowany reprezentuje przykłady pokryte przez całą regułę $r_{c'}$. Zauważyć można, że składa się on z dwóch prostokątów w obu górnych rogach wykresu. Do pokrycia takiego zbioru przykładów potrzeba dwóch reguł z warunkami prostymi, jako że każda z nich jest w stanie pokryć jedynie jeden z prostokątów. Nieokrojona postać reguły r_c zawiera jednak dodatkowo warunek $thal \neq normal$. Jako że atrybut thal posiada trzy możliwe wartości, jego odpowiednikiem jest alternatywa dwóch warunków prostych: $thal = reversable\_defect$ oraz $thal = fixed\_defect$. Rozważmy jednak w tym miejscu regułę, która zawiera w przesłance zarówno warunek $(oldpeak \geqslant 0.55 \vee oldpeak < 0.15)$, jak i $thal \neq normal$. Aby pokryć wszystkie pokrywane przez nią przykłady z użyciem warunków prostych

używanych przez RuleKit, konieczne jest użycie aż 4 reguł. Zostały one przedstawione poniżej.

$$\mathbf{r_{k1'}}: \textbf{IF } \text{oldpeak} \geqslant 0.55 \ \wedge \ thal = reversable\_defect \textbf{ THEN } \text{class} = 1$$
$$\mathbf{r_{k2'}}: \textbf{IF } \text{oldpeak} \geqslant 0.55 \ \wedge \ thal = fixed\_defect \textbf{ THEN } \text{class} = 1$$
$$\mathbf{r_{k3'}}: \textbf{IF } \text{oldpeak} < 0.15 \ \wedge \ thal = reversable\_defect \textbf{ THEN } \text{class} = 1$$
$$\mathbf{r_{k4'}}: \textbf{IF } \text{oldpeak} < 0.15 \ \wedge \ thal = fixed\_defect \textbf{ THEN } \text{class} = 1$$

Zapis taki mógłby zostać wygenerowany przez algorytm RuleKit, jednak w praktyce tak się nie stało. Pokazuje to, że choć teoretycznie relacje opisane z użyciem wspominanych warunków złożonych można opisać również z wykorzystaniem warunków prostych, to nie znaczy to, że faktycznie taki opis zostanie uzyskany przez algorytm. Metoda ComplexConditions, pomimo dużego podobieństwa do metody RuleKit, znalazła w tym miejscu całkowicie nową zależność w danych. Należy również pamiętać, że taki alternatywny opis, z użyciem samych warunków prostych, byłby znacząco bardziej skomplikowany, a zatem potencjalnie trudniejszy w interpretacji i zrozumieniu.

Co ciekawe, metody DeepRules or MofNRules nie odnalazły podobnej reguły, pomimo że również wykorzystują one warunki złożone. Osiągnęły także widocznie gorsze wyniki predykcyjne oraz średnią jakość reguły. Pokazuje to, że pomimo podobieństw, każda z metod działa odmiennie, mogąc generować różne od siebie opisy. W dużej mierze od samych danych zależy to, która okaże się dawać najlepsze rezultaty. Metody DeepRules i MofNConditions potrafią wyszukiwać bardziej skomplikowane opisy niż algorytm ComplexConditions. Ich słabsze wyniki mogą sugerować, że takie zależności zwyczajnie nie występują w omawianym tutaj zbiorze danych.

3 Zbiór danych dotyczący zanieczyszczenia ozonem w Los Angeles „Ozone”

Drugi przykład zostanie poświęcony zbiorowi danych dotyczącemu zanieczyszczenia powietrza ozonem w Los Angeles w roku 1976. Został on po raz pierwszy opisany przez Breiman i Friedman w roku 1985 w pracy zatytułowanej „Estimating Optimal Transformations for Multiple Regression and Correlation”. W tym miejscu została wykorzystana jego wersja pochodząca z pakietu języka R o nazwie mlbench. Zbiór ten można potraktować zarówno jako szereg czasowy, jak i dane regresyjne. W tym przykładzie skupiono się jednak na problemie regresji, stąd też zostały pominięte

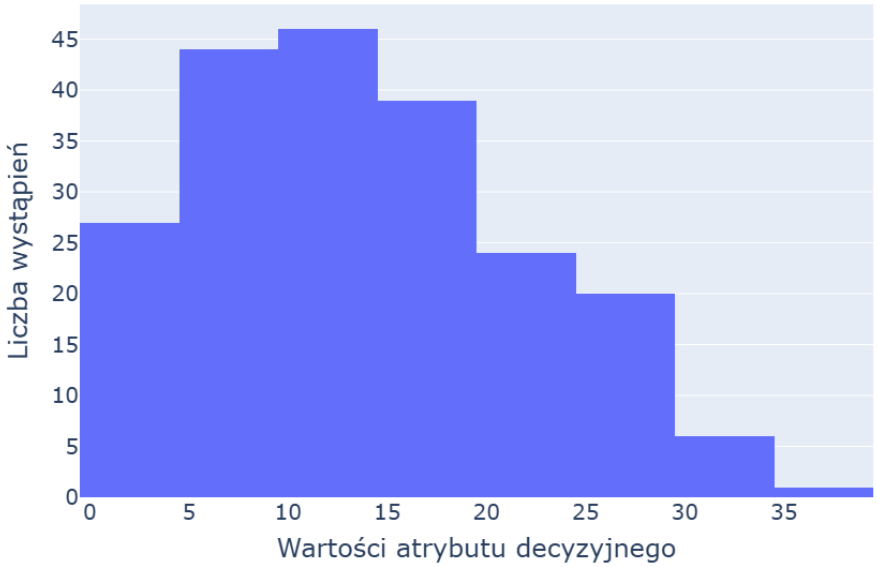
atrybuty dotyczące czasu, takie jak: miesiąc, dzień miesiąca oraz tygodnia. Pozostałe z nich, które zostały użyte, wymieniono i opisano poniżej:

- **vdb500ht** - Wysokość, na jakiej temperatura powietrza osiąga 5500 milibarów w metrach.
- **wdsp** - Prędkość wiatru w milach na godzinę.
- **hmdty** - Wilgotność potwietrza w procentach.
- **sdafbtmp** - Temperatura powietrza w stopnia Fahrenheita mierzona w szkole średniej Sandburg w Los Angeles.
- **invht** - Wysokość inwersji mierzona w stopach (z ang. inversion height).
- **dagpg** - Gradient ciśnienia geopotencjalnego (mm Hg).
- **invtmp** - Temperatura inwersji w stopniach Fahrenheita (z ang. inversion height).
- **vzblty** - Widoczność w milach.
- **target** - Dzienny maksymalny poziom ozonu w powietrzu (numeryczny atrybut decyzyjny).

Problem regresyjny polega tutaj na prognozie maksymalnego dziennego poziomu ozonu w powietrzu na podstawie wartości pozostałych atrybutów. Przed przejściem do indukcji reguł, zbiór danych został podzielony na część treningową i testową w proporcjach 70 i 30 procent. Następnie zbadano rozkład atrybutu decyzyjnego w pierwszej z nich. Jego wartość minimalna wyniosła 2, a maksymalna 38. Dolny kwartyl był równy 7, górny 19, a mediana 13. Dla lepszego zobrazowania rozkładu na rysunku 8.3 przedstawiono jego histogram.

Na jego podstawie widać, że rozkład ten przypomina rozkład Gaussa. Najczęściej pojawiającymi się wartościami atrybutu decyzyjnego są te z przedziału 5 do 14. Pozostałe wartości występują tym rzadziej, im bardziej odstają od tego zakresu.

Następnie zostały wygenerowane zbiory reguł z użyciem wszystkich zaproponowanych w pracy metod oraz dwóch algorytmów referencyjnych: RuleKit oraz drzew decyzyjnych. Zastosowano te same wartości parametrów, jak w przypadku wcześniejszych eksperymentów opisanych w rozdziale 7 w sekcji 3. Tabela 8.17 prezentuje



Rysunek 8.3: Histogram wartości atrybutu decyzyjnego zbioru dotyczącego zanieczyszczenia powietrza ozonem.

wyniki. Oceniane było RRMSE na zbiorze testowym, liczba reguł, średnia liczba warunków w regule, sumaryczna liczba warunków w całym zbiorze oraz średnia jakość reguł oceniana miarą C2.

Algorytmem osiągającym najgorszy wynik predykcji, a więc największą wartość RRMSE, był tym razem ComplexConditions (0.235). Nieco lepszy okazał się tutaj RuleKit z wartością 0.214. Zbliżony wynik (0.203) osiągnęła metoda MofNRules w wariancie „dokładnie M-z-N". Drugi z wariantów okazał się bliższy drzewom

Tabela 8.17: Wyniki badanych metod ocenianych na części testowej zbioru danych dotyczącego zanieczyszczenia ozonem w Los Angeles w roku 1976

Metoda	RRMSE (test)	Liczba reguł	Średnia liczba warunków w regule	Sumaryczna liczba warunków	Średnia jakość
Drzewa decyzyjne	0.177	236	24.292	5733	–
RuleKit	0.214	33	3.788	125	0.469
ComplexConditions	0.235	37	7.54	279	0.503
MofNRules („dokładnie M-z-N")	0.203	23	4.304	99	0.463
MofNRules („przynajmniej M-z-N")	0.18	27	5.185	140	0.472
DeepRules	0.169	6	7	42	0.646

decyzyjnym (0.177) z RRMSE wynoszącym 0.18. Najlepszym algorytmem w tym konkretnym przykładzie był DeepRules z RRMSE równym 0.169. Doprowadził on także do najmniej skomplikowanego zbioru reguł, patrząc na ich liczbę (6) oraz sumaryczną liczbę warunków (42). Zarówno pierwsza, jak i druga wartość są widocznie niższe niż te osiągane przez pozostałe metody. Dla porównania drugi w kolejności zbiór reguł o najmniejszej sumarycznej liczbie warunków, uzyskany z użyciem MofNRules w wariancie „dokładnie M-z-N", posiadał ich 99, czyli ponad dwukrotnie więcej. Patrząc na średnią liczbę warunków w regule, można zauważyć, że największą wartość osiągnęły tutaj drzewa decyzyjne (24.292). Najkrótsze reguły generował z kolei RuleKit (średnio 3.788 warunków w przesłance). W przypadku algorytmu DeepRules było to średnio 7 warunków, co jest wynikiem wyższym niż w przypadku metody MofNRules (4.304 dla wariantu „dokładnie M-z-N" i 5.185 dla wariantu „przynajmniej M-z-N"). ComplexConditions uzyskał wynik na poziomie 7.54, generując najdłuższe reguły spośród wszystkich zaproponowanych w tej pracy metod. Wygenerowany z jego użyciem zbiór reguł był również najbardziej liczny (37 reguł).

Wyniki te potwierdzają oczywisty, choć wart odnotowania wniosek: żadna z zaproponowanych metod nie osiąga jednocześnie najlepszych rezultatów we wszystkich przypadkach. Ich skuteczność, zarówno pod kątem jakości predykcji, jak i złożoności otrzymywanych zbiorów reguł, zależy od specyfiki danych i występujących w nich zależności. Wcześniejszy rozdział 7 skupiał się na przetestowaniu algorytmów na wielu zbiorach w celu wyciągnięcia pewnych ogólnych wniosków. Należy jednak pamiętać, że każdy zbiór danych należy traktować indywidualnie. Warto przy tym, w miarę możliwości, wypróbować wiele różnych algorytmów, aby wybrać najbardziej odpowiedni do danego zadania.

W ramach tego przykładu szczególną uwagę poświęcono metodzie DeepRules ze względu na najmniejszą wartość RRMSE oraz liczbę wygenerowanych przez nią reguł. Jako że było ich jedynie 6, w przeciwieństwie do poprzedniego przykładu, zaprezentowany zostanie cały ich zbiór (patrz tabela 8.18).

Nawiasy kwadratowe znajdujące się obok konkluzji definiują granicę przedziału ufności reguły. Na jego podstawie były obliczane wartości p , n , P oraz N . Są one jednak wyznaczane inaczej niż we wcześniejszym przykładzie dotyczącym problemu klasyfikacji. Wartości P i N oznaczają liczbę przykładów ze zbioru treningowego, dla których wartość atrybutu decyzyjnego odpowiednio należy lub nie należy do przedziału ufności. Natomiast p i n określają, ile z tych przykładów wpadających do przedziału ufności zostało lub nie zostało pokrytych przez przesłankę reguły.

$\mathbf{r_{d1}}$	IF sdafbtmp $\leqslant 68 \vee$ invht $\geqslant 3,415 \vee$ vdb500ht $\leqslant 5,700 \vee$ vzblty $\leqslant 2$ THEN target $= 8.92\,[3.9, 13.95]$ $(p = 83, n = 33, P = 94, N = 113)$
$\mathbf{r_{d2}}$	IF invtmp $\geqslant 90.5$ THEN target $= 32.5\,[27, 38]$ $(p = 2, n = 0, P = 17, N = 190)$
$\mathbf{r_{d3}}$	IF (dagpg $\leqslant -9 \wedge$ sdafbtmp $\leqslant 75 \wedge$ vzblty $\geqslant 3) \vee$ (invtmp $\leqslant 53 \wedge$ sdafbtmp $\leqslant 58 \wedge$ hmdty $\leqslant 77) \vee$ (invht $\geqslant 3,624.5 \wedge$ wdsp $\geqslant 6.5) \vee$ sdafbtmp $\leqslant 35.5 \vee$ wdsp $\leqslant 0 \vee$ invht $<$ vzblty THEN target $= 6.67\,[1.53, 11.82]$ $(p = 62, n = 5, P = 87, N = 120)$
$\mathbf{r_{d4}}$	IF (dagpg $\leqslant -9 \wedge$ vzblty $\geqslant 35) \vee$ (invtmp $\leqslant 53 \wedge$ sdafbtmp $\leqslant 58 \wedge$ hmdty $\leqslant 78.5) \vee$ (invht $\geqslant 3,624.5 \wedge$ wdsp $\geqslant 6.5) \vee$ sdafbtmp $\leqslant 35.5 \vee$ dagpg $\leqslant -68 \vee$ invht $<$ vzblty THEN target $= 6.96\,[1.58, 12.33]$ $(p = 63, n = 5, P = 100, N = 107)$
$\mathbf{r_{d5}}$	IF hmdty $\leqslant 52.5 \vee$ (invtmp $\leqslant 61.5 \wedge$ vzblty $\geqslant 45 \wedge$ dagpg $=< 21.5, 38 >) \vee$ (dagpg $\leqslant -10 \wedge$ hmdty $\geqslant 55.5) \vee$ invtmp $\leqslant 48 \vee$ (vzblty $\leqslant 7 \wedge$ hmdty $\geqslant 54) \vee$ (invht $\geqslant 3,624.5 \wedge$ sdafbtmp $\leqslant 65) \vee$ vdb500ht $\leqslant 5,670$ THEN target $= 8.34\,[2.43, 14.26]$ $(p = 75, n = 15, P = 112, N = 95)$
$\mathbf{r_{d6}}$	IF sdafbtmp $\geqslant 66.5 \wedge$ vzblty $\geqslant 8.5 \wedge$ invht $\geqslant 348.5 \wedge$ dagpg $\geqslant -39$ THEN target $= 19.91\,[13.45, 26.36]$ $(p = 67, n = 29, P = 81, N = 126)$

Tabela 8.18: Zbiór reguł wygenerowany na zbiorze „Ozone" z użyciem algorytmu DeepRules

W pierwszym kroku zostaną przeanalizowane konkluzje reguł, a więc prognozowany przez nie poziom zanieczyszczenia ozonem. Wartości te zestawimy w tym miejscu z rozkładem wartości atrybutu decyzyjnego. Na podstawie histogramu można określić, że wartości pojawiające się w konkluzjach reguł r_{d1} oraz r_{d5} (kolejno 8.92 i 8.34) występują w danych stosunkowo często. Można więc powiedzieć, że opisują one sytuacje, dla których poziom zanieczyszczeń nie odbiega zbytnio od średnich wartości. Z punktu widzenia tego przykładu ciekawsze wydają się jednak pozostałe reguły. W przypadku r_{d2} oraz r_{d6} wartości w konkluzjach były wyższe niż górny kwartyl (kolejno 32.5 oraz 19.91). Reguły te opisują, dla jakich warunków poziom zanieczyszczenia powietrza jest wyższy niż zazwyczaj. Z kolei dla reguł r_d3, r_{d4} konkluzje wskazują na wartości mniejsze od dolnego kwartyla (kolejno 6.67 oraz 6.96). Dają więc one informacje o tym, w jakich sytuacjach poziom ozonu w powietrzu jest szczególnie niski.

Przyjrzyjmy się teraz samym przesłankom reguł. Pokazują one, jak różnorodne opisy danych mogą zostać uzyskane z użyciem algorytmu DeeRules. Przykładowo reguła r_{d1} posiada przesłankę w postaci alternatywy warunków elementarnych. Sprawia to jednak, że sama w sobie nie jest zbyt precyzyjna. Reguła r_{d4} stanowi z kolei dobry przykład rozbudowanej reguły DNF. Posiada ona sześć składników, z czego ostatnie trzy są jednoelementowe. Reguła r_d6 posiada w swojej przesłance jedynie koniunkcje warunków prostych i jako taka mogłaby zostać wygenerowana przez

algorytm RuleKit. W rzeczywistości jednak tak się nie stało. W regułach uzyskanych przez RuleKit nie istnieje taka, która wykazywałaby podobieństwo do r_{d6}. Żadna z nich nie zawiera także ani jednego z warunków, który pojawia się w r_{d6}. Sugeruje to, że obydwie metody odkryły w tym wypadku inne zależności.

Sprawdzono w tym miejscu także, jak wyglądały reguły uzyskane algorytmem RuleKit, których konkluzje wskazywały na podobne wartości atrybutu decyzyjnego, jak ma to miejsce w r_{d6}. Zostały one przedstawione poniżej.

$\mathbf{r_{k1}}$	IF vdb500ht $\in [5,885, 5,895) \wedge$ vzblty $\in [55, 75)$ THEN class $= 19\,[19, 19]$ $(p = 2, n = 0, P = 10, N = 287)$
$\mathbf{r_{k2}}$	IF invtmp $\in [82, 84.5) \wedge$ vdb500ht $< 5,875 \wedge$ hmdty < 84 THEN class $= 20\,[15.45, 24.55]$ $(p = 1, n = 2, P = 61, N = 236)$
$\mathbf{r_{k3}}$	IF wdsp $\geqslant 3.5 \wedge$ vdb500ht $\geqslant 5,770 \wedge$ invtmp $< 76.5 \wedge$ invht $< 2,887 \wedge$ vzblty $=< 23.5, 80) \wedge$ dagpg $\in [25, 56.5)$ THEN class $= 20.17\,[14.13, 26.21]$ $(p = 4, n = 2, P = 78, N = 219)$

Tabela 8.19: Reguły wygenerowane algorytmem RuleKit, których konkluzje wskazują na podobne wartości jak w przypadku konkluzji reguły r_{d6}

Jak widać, nie mają one wiele wspólnego z opisem uzyskanym przez r_{d6}. W przypadku r_{k1} zakres wartości atrybutu vzblty (widoczność) jest całkowicie inny. Reguła r_{k2} bazuje z kolei na zupełnie innym zestawie atrybutów. Ostatnie z nich r_{k3} również stosuje odmienne deskryptory. Mają one czasem pewną część wspólną z tymi użytymi w r_{d6}, mimo to sama reguła jest zupełnie inna. Warto w tym miejscu zwrócić uwagę na niskie wartości p i n reguł r_{k1}, r_{k2} i r_{k3}. Pokrywają one znacznie niższą liczbę przykładów niż r_{d6}. Po przeanalizowaniu zbiorów reguł uzyskanych metodami ComplexConditions oraz MofNRules okazało się, że również w nich nie odnaleziono podobnego opisu.

Powyższy przykład prezentuje różnorodność opisów, jakie jest w stanie generować i odkrywać algorytm DeepRules. W tym konkretnym wypadku uzyskał on zbiór o małej liczbie reguł. One same nie zawsze są jednak proste w zrozumieniu i interpretacji. Poświęćmy w tym miejscu chwilę na próbę opisu słownego relacji definiowanej przez regułę r_d3. Da to pogląd na to, jak skomplikowane relacje w danych jest w stanie odkrywać DeepRules. Regułę tę można zinterpretować następująco:

Stosunkowo niski poziom zanieczyszczenia powietrza ozonem (około 6.67) można odnotować, gdy występuje co najmniej jeden z poniższych przypadków:

- gradient ciśnienia geopotencjalnego jest niższy niż -9, temperatura powietrza jest wyższa niż 75 stopni °F oraz widoczność jest wyższa lub równa 3 mile;

- temperatura inwersji jest niższa lub równa 53 °F, temperatura powietrza jest mniejsza lub równa 58 °F oraz wilgotność jest niższa lub równa 77%;
- wysokość inwersji jest wyższa lub równa 3,624.5 stopy oraz prędkość wiatru jest wyższa lub równa 6.5 m/h;
- temperatura powietrza jest niższa lub równa 35.5 °F;
- prędkość wiatru jest niższa lub równa 0;
- wysokość inwersji jest mniejsza od widoczności.

W powyższym opisie uwagę przyciągają dwa ostatnie przypadki. Pierwszy z nich dotyczy sytuacji, gdzie prędkość wiatru jest mniejsza lub równa 0 m/h. Na pierwszy rzut oka taki warunek wydaje się niezbyt sensowny. W rzeczywistości jednak zbiór posiada sumarycznie aż 15 wierszy z zerową wartością prędkości wiatru. Oczywiście prędkość ta nigdy nie była mniejsza od zera, co byłoby niemożliwe. Warunek ten równie dobrze można by więc zamienić na warunek $wdsp = 0$. Takie deskryptory nie są jednak brane przez algorytm w przypadku atrybutów numerycznych.

Drugi wspomniany przypadek występuje wtedy, gdy wysokość inwersji jest mniejsza od widoczności. Zasadność takiego opisu wydaje się już na pierwszy rzut oka wątpliwa. Pamiętamy wszak, że pierwszy z atrybutów mierzony jest w stopach, a drugi w milach. Należy w tym miejscu pamiętać, że odkryta w sposób automatyczny wiedza nie zawsze pokrywa się z wiedzą dziedzinową, a czasem i zdrowym rozsądkiem, nawet jeżeli znajduje potwierdzenie w danych. Oczywiście ryzyko pojawiania się takich reguł można by łatwo mitygować poprzez ręczne zdefiniowanie, które atrybuty mogą być ze sobą porównywane z użyciem warunków relacji atrybutów. Aktualnie opracowane przez autora metody nie posiadają parametrów umożliwiających taką konfigurację, w przyszłości jest jednak planowane ich dodanie. Warto jednak zauważyć i podkreślić fakt, że w tym konkretnym przypadku, mając do czynienia z interpretowalnym modelem regułowym, taka sytuacja została łatwo wyłapana. Stosując nieinterpretowalne metody uczenia maszynowego, byłoby to o wiele trudniejsze, ponieważ zwykle niewiele wiadomo o tym, na podstawie jakich zależności w danych podejmują one decyzje.

4 Zbiór danych dotyczący przeszczepu szpiku kostnego „BMT-Ch”

Trzecie i ostatnie studium przypadku poświęcono problemowi analizy przeżycia. Wykorzystany w nim zbiór danych opisuje 187 pacjentów poddanych zabiegowi przeszczepu szpiku kostnego. Spośród nich 75 to kobiety, a 112 mężczyźni, w wieku od 7 miesięcy do 20 lat i 2 miesięcy. Mediana wieku wynosi 9 lat i 7 miesięcy. Dane zostały zebrane przez Klinikę Transplantacji Szpiku, Onkologii i Hematologii Dziecięcej należącą do Uniwersyteckiego Szpitala Klinicznego we Wrocławiu. Wśród wszystkich pacjentów aż u 155 występowały różne rodzaje nowotworów złośliwych, w tym 67 przypadków ostrej białaczki limfoblastycznej, 33 ostrej białaczki szpikowej, 25 przewlekłej białaczki szpikowej i 18 zespołu mielodysplastycznego. U pozostałych 32 pacjentów odnotowano choroby niezłośliwe. Spośród nich u 13 wystąpiła ciężka niedokrwistość aplastyczna, u 5 niedokrwistość Fanconiego a u 4 adrenoleukodystrofia sprzężona z chromosomem X . We wspomnianym zbiorze danych za zdarzenie uznano wystąpienie zgonu pacjenta.

U każdego z pacjentów został przeprowadzony allogeniczny przeszczep komórek macierzystych krwiotwórczych od niespokrewnionego dawcy, wykonany zgodnie z europejskimi protokołami i wytycznymi Europejskiej Grupy Roboczej ds. Wrodzonych Wad Krwi i Szpiku Kostnego. W zbiorze występuje 37 atrybutów, poniżej wymienione i opisane zostały wybrane z nich.

- **donor_age** - Wiek dawcy.
- **acute_GvHD_II_III_IV** - Atrybut binarny określający czy wystapił rozwój ostrej choroby przeszczep przeciwko gospodarzowi w stadium II, III lub IV (Tak – 1, Nie – 0).
- **recipient_age** - Wiek biorcy.
- **recipient_age_below_10** - Atrybut binarny określający czy biorca w chwili przeszczepu miał skończone 10 lat lub więcej.
- **relapse** - Atrybut binarny określający czy wystąpił nawrót choroby.
- **acute_GvHD_III_IV** - Atrybut binarny określający czy wystapił rozwój ostrej choroby przeszczep przeciwko gospodarzowi w stadium III lub IV (Tak – 0, Nie – 1).

- **CD34_x1e6_per_kg** - Dawka komórek CD34+ na kilogram masy ciała biorcy.
- **CD3_to_CD34_ratio** - Stosunek komórek CD3+ do komórek CD34+.
- **recipient_body_mass** - Masa ciała biorcy w kilogramach.
- **time_to_acute_GvHD_III_IV** - Czas do rozwoju ostrej choroby przeszczep przeciwko gospodarzowi w stadium III lub IV.
- **survival_time** - Czas obserwacji (jeśli pacjent żyje) lub czas, który upłynął do wystąpienia zgonu w dniach.
- **survival_status** - Atrybut binarny określający czy pacjent przeżył czy też nie.

Zdefiniowany tutaj problem analizy przeżycia polega na prognozie krzywej Kaplana-Meiera przeżycia danego pacjenta. Krzywa ta jest funkcją, określającą prawdopodobieństwo przeżycia w danej chwili czasu po przeprowadzeniu zabiegu. Zbiór danych podzielono na część treningową i testową, liczące kolejno 168 i 19 przykładów. Następnie wygenerowane zostały zbiory reguł z użyciem wszystkich badanych metod. Ponownie użyto tych samych wartości hiperparametrów, jak miało to miejsce w eksperymentach opisanych w rozdziale 7. Każdy z uzyskanych zbiorów reguł oceniono następnie na danych testowych, a uzyskane w ten sposób wyniki zaprezentowano w tabeli 8.20. Jakość reguły, w przeciwieństwie do wcześniejszych przykładów, obliczona została na podstawie p-wartości testu logrank jako 1 − p-wartość.

Tabela 8.20: Wyniki badanych metod na części testowej zbioru danych „BMT-Ch”

Metoda	IBS (test)	Liczba reguł	Średnia liczba warunków w regule	Sumaryczna liczba warunków	Średnia jakość
Drzewa decyzyjne	0.0985	31	6.484	201	–
RuleKit	0.161	14	3.214	45	0.96
ComplexConditions	0.202	12	3.25	39	0.986
MofNRules („dokładnie M-z-N”)	0.173	4	1.75	7	0.998
MofNRules („przynajmniej M-z-N”)	0.177	3	1.667	5	0.991
DeepRules	0.194	8	15.75	126	1.0

Na podstawie tabeli 8.20 zauważyć można, że najlepszą metodą, pod kątem wyników predykcji (najniższego IBS), jest drzewo przeżyciowe (0.0985). Zamieniając

je na reguły, odpowiada ono zbiorowi 31 reguł, zawierając w swych przesłankach średnio 6.5 warunków. RuleKit osiągnął w tym przypadku IBS równy 0.161, generując 14 reguł. Wszystkie z proponowanych metod uzyskały w tym wypadku gorsze wyniki predykcyjne (wyższe wartości IBS) od algorytmów referencyjnych. Najgorszy spośród nich (ComplexCondition), z IBS równym 0.202, doprowadził do zbioru o 12 regułach, zawierających średnio 3.25 warunków w przesłance. Algorytm DeepRules wygenerował tutaj mniej, bo jedynie 8 reguł. Były one jednak znacząco dłuższe (średnio 15.8 warunków w przesłance). Najmniejszą liczbę, najkrótszych reguł, uzyskały tutaj obydwa warianty metody MofNRules. Dla wariantu „dokładnie M-z-N" było to 4 reguły, a dla drugiego 3. Wyniki predykcyjne obydwu z nich były do siebie zbliżone (0.173 dla wariantu „dokładnie M-z-N" i 0.177 dla „przynajmniej M-z-N").

W przypadku rozważanego zbioru danych, reguły uzyskane metodą MofNRules doprowadziły do znacznie mniejszej liczby reguł niż pozostałe, powodując jednocześnie stosunkowo niewielkie pogorszenie wyniku IBS względem referencyjnego algorytmu RuleKit. Z tego powodu, w niniejszym przykładzie, to właśnie jej poświęcona zostanie szczególna uwaga.

W tabeli 8.21 widoczne są wszystkie 4 reguły wygenerowane algorytmem MofNRules, w wariancie „dokładnie M-z-N". Część konkluzji została tutaj pominięta, ponieważ odpowiada estymatorowi Kaplana-Meiera, który trudno byłoby przedstawić w formie tekstowej. Zamiast tego podane zostało wsparcie reguły, obliczone jako stosunek liczby pokrytych przez przesłankę przykładów do liczby wszystkich przykładów w zbiorze treningowym.

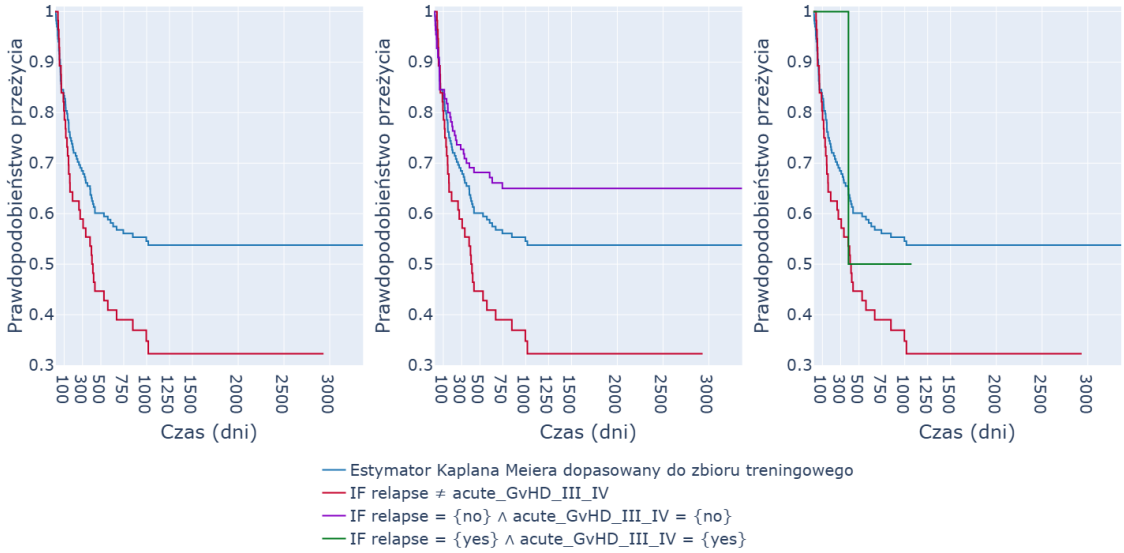
$\mathbf{r_{md1}}$	IF $\neg$2-z-3(donor_age $\notin [21.4, 21.97)$, CD34_x1e6_per_kg $\notin [1.35, 1.99)$, recipient_body_mass $\notin [33.5, 35.5)$) $\wedge$ relapse = acute_GvHD_III_IV (wsparcie = 0.613)
$\mathbf{r_{md2}}$	IF $\neg$2-z-3(CD3_to_CD34_ratio $\notin [2.43, 2.50)$, CD34_x1e6_per_kg $\notin [9.88, 10.96)$, donor_age $\neq< 21.40, 21.97)$) $\wedge$ relapse = acute_GvHD_III_IV (wsparcie = 0.601)
$\mathbf{r_{md3}}$	IF recipient_age_below_10 $\neq$ acute_GvHD_II_III_IV $\wedge$ time_to_acute_GvHD_III_IV $<$ 19.50 (wsparcie = 0.036)
$\mathbf{r_{md4}}$	IF relapse $\neq$ acute_GvHD_III_IV (wsparcie = 0.339)

Tabela 8.21: Zbiór reguł wygenerowany na zbiorze „BMT-Ch" z użyciem algorytmu MofNRules w wariancie „dokładnie M-z-N"

W tabeli 8.22 zaprezentowano zbiór trzech reguł uzyskanych z wykorzystaniem wariantu „przynajmniej M-z-N".

$\mathbf{r_{mp1}}$	IF 2-z-3(CD34_x1e6_per_kg $\notin [4.51, 5.08)$, donor_age < 44.06, recipient_body_mass $\notin [33.5, 35.5)) \wedge$ relapse = acute_GvHD_III_IV (wsparcie = 0.655)
$\mathbf{r_{mp2}}$	IF recipient_age_below_10 $\neq$ acute_GvHD_II_III_IV $\wedge$ time_to_acute_GvHD_III_IV > 19.5 (wsparcie = 0.036)
$\mathbf{r_{mp3}}$	IF relapse $\neq$ acute_GvHD_III_IV (wsparcie = 0.339)

Tabela 8.22: Zbiór reguł wygenerowany na zbiorze „BMT-Ch” z użyciem algorytmu MofNRules w wariancie „przynajmniej M-z-N”



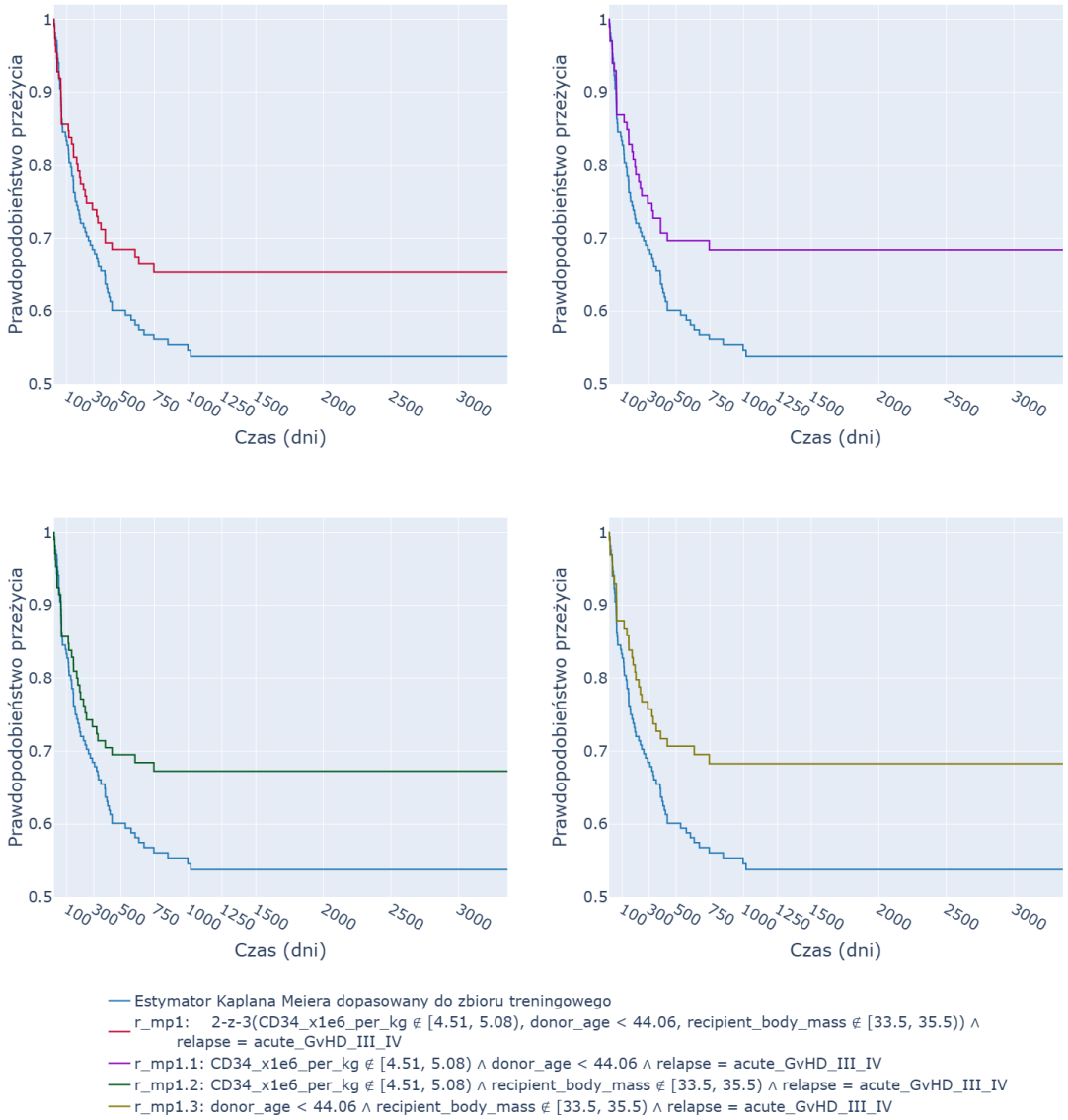
Rysunek 8.4: Wizualizacja konkluzji reguł r_{md4} i r_{mp3}

W zbiorze uzyskanym z użyciem wariantu „dokładnie M-z-N” obecne są dwie reguły z pojedynczym zanegowanym warunkiem 2-z-3. W przypadku „przynajmniej M-z-N”, tego typu warunek pojawił się tylko raz. Wszystkie reguły obydwu zbiorów zawierające ten rodzaj deskryptora cechują się widocznie wyższym wsparciem (powyżej 60%). Każda z nich zawiera oprócz warunku M-z-N, również ten sam warunek relacji atrybutów $relapse = acute\_GvHD\_III\_IV$. Negacja tego warunku, oznaczona dalej jako c_p, występuje także w przesłance ostatniej z reguł obydwu zbiorów. Rysunek 8.4 prezentuje krzywą Kaplana-Meiera przykładów pokrytych przez c_p, porównując ją z krzywą całego zbioru treningowego.

Analizując wykresy widoczne na rysunku 8.4, można zauważyć, że obecność jednego z czynników, takich jak rozwój ostrej choroby przeszczep przeciwko gospodarzowi (aGvHD) lub nawrót choroby, ma negatywny wpływ na szanse przedłużenia życia pacjenta. Taką sytuację reprezentuje czerwona krzywa na wykresie. W jej przypadku

wartość prawdopodobieństwa spada wraz z czasem szybciej niż ma to miejsce dla estymatora Kaplana-Meiera całego zbioru (krzywa niebieska). Sytuacja, gdy obydwa czynniki nie występują, zwiększa z kolei szanse pacjenta na przeżycie (krzywa koloru fioletowego). Kolorem zielonym oznaczono krzywą przeżycia dla przypadku, gdy u pacjenta wystąpi zarówno nawrót choroby, jak i rozwój ostrej choroby przeszczep przeciwko gospodarzowi. Niestety, zawiera ona bardzo niewiele punktów, dając niepełny obraz sytuacji. Dzieje się tak, ponieważ w zbiorze treningowym sytuacja taka występuje tylko dla dwóch takich przykładów, z których jeden powiązany jest z wystąpieniem zgonu. Stąd też najniższa wartość prawdopodobieństwa zielonej krzywej równa jest 0.5.

Przyjrzyjmy się teraz bliżej regule zawierającej warunek M-z-N, na przykładzie reguły r_{mp1} wygenerowanej z użyciem MofNRules w wariancie „przynajmniej M-z-N". Jej przesłankę można przedstawić w postaci alternatywy trzech koniunkcji. Wszystkie z nich potraktowane zostały jako osobne reguły oznaczone kolejno jako: $r_{mp1.1}$, ..., $r_{mp1.3}$. Na podstawie takiej interpretacji sporządzony został rysunek 8.5. Prezentuje on krzywe przeżycia przykładów pokrywanych przez regułę r_{mp1} oraz reguły $r_{mp1.1}$,...,$r_{mp1.3}$, na tle krzywej całego zbioru treningowego.



Rysunek 8.5: Wizualizacja konkluzji reguły r_{mp1}

Patrząc na ów rysunek, można zauważyć, że krzywe $r_{mp1.1}$, $r_{mp1.2}$,$r_{mp1.3}$ r_{mp1} są do siebie bardzo zbliżone. Wartość wsparcia r_{mp1}, wynosząca 65%, jest jednak znacznie wyższa niż dla pozostałych trzech reguł, gdzie nie przekracza ono 59%. Warto zaznaczyć w tym miejscu, że zarówno algorytm RuleKit, jak i ComplexConditions oraz DeepRules nie zdołały wygenerować żadnej z reguł $r_{mp1},\ldots,r_{mp3}$. Sugeruje to, że metoda MofNRules odkryła w tym wypadku zupełnie nową zależność, która pozwoliła jej w znacznie bardziej zwięzły i kompleksowy sposób opisać dane treningowe. W tym konkretnym przypadku było to jednak związane z pewnym pogorszeniem wyników predykcyjnych.

9 Wdrożenie

1 Wstęp

Niniejsza dysertacja podsumowuje czteroletnie prace prowadzone przez autora w ramach piątej edycji programu „Doktorat wdrożeniowy". Wcześniejsza jej część była poświęcona aspektom naukowym i badawczym. Specyfika wspomnianego programu wymaga jednak, by jednym z wymiernych efektów końcowych było faktyczne wdrożenie wyników badań przez podmiot zatrudniający autora. W tym wypadku jest to Instytut Sztucznej Inteligencji i Cyberbezpieczeństwa należący do Sieci Badawczej Łukasiewicz, z siedzibą w Katowicach (ozn. Ł-AI), wcześniej znany jako Instytut Technik Innowacyjnych EMAG. Niniejszy rozdział ma za zadanie przybliżyć czytelnikowi aspekt wdrożeniowy prowadzonych przez autora prac.

2 Platforma RuleMiner

Równolegle z prowadzonymi badaniami, przez cały okres trwania studiów doktoranckich autora, w Ł-AI trwały intensywne prace nad budową internetowej platformy analitycznej o nazwie RuleMiner. Były one realizowane w ramach projektu „Rozwój interinstytucjonalnych technologii agregacji, przetwarzania i wizualizacji danych statystycznych instytucji publicznych." Powstanie wspomnianej platformy było jednym z planowanych efektów projektu.

Platforma RuleMiner jest obecnie ogólnodostępna, również w ramach darmowego planu, pod adresem: `https://app.ruleminer.ai/` (dostęp 16.09.2025). Jej głównymi zastosowaniami są analiza danych, automatyczne odkrywanie wiedzy oraz budowanie systemów predykcyjnych. Konkurencyjne platformy typu BI w większości nastawione są na wspomaganie użytkownika w procesie analizy danych poprzez oferowanie mu pewnego zestawu narzędzi statystycznych oraz metod uczenia maszynowego, a także związanych z nimi wizualizacji. Nie posiadają one jednak zwykle mechanizmów automatycznego odkrywania zależności i relacji w danych. Użytkownik

ma zwykle możliwość jedynie weryfikowania swoich własnych hipotez oraz sprawdzania ich prawdziwości. Inna grupa produktów jest nastawiona z kolei na trenowanie modeli predykcyjnych z nastawieniem na wysoką dokładność prognoz. Zwykle jednak nie adresują lub nie przykładają szczególnej uwagi do problemu wyjaśnialności podejmowanych przez nie decyzji.

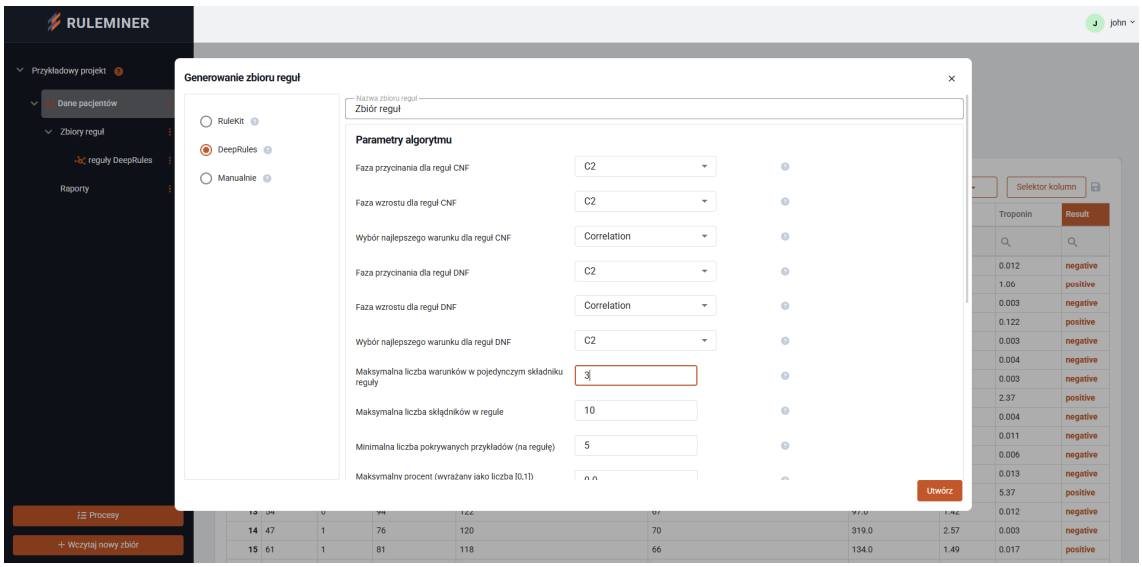
RuleMiner oferuje unikalne podejście, skupiając się na interpretowalnych algorytmach uczenia maszynowego, ze szczególnym uwzględnieniem metod regułowych, automatyzując, do pewnego stopnia, proces analizy danych oraz odkrywania zawartej w nich wiedzy.

RuleMiner pozwala użytkownikowi na wczytanie swoich własnych zbiorów danych tabelarycznych dotyczących problemów klasyfikacji, regresji lub analizy przeżycia. Następnie, po ich uprzednim przygotowaniu, umożliwia wykonanie na nich szeregu przydatnych analiz i wizualizacji. Użytkownik może także, co najważniejsze, wygenerować na ich podstawie zbiory reguł z użyciem wybranego, skonfigurowanego przez siebie algorytmu regułowego. Uzyskane w ten sposób opisy mogą być następnie eksplorowane, modyfikowane i wizualizowane w łatwy, graficzny sposób, w celu uzyskania odpowiedzi na nurtujące go pytania.

Dla mniej zaawansowanych użytkowników platforma oferuje funkcjonalność automatycznego doboru wartości hiperparametrów algorytmów w oparciu o jego odpowiedzi udzielone w krótkiej ankiecie. Dzięki temu każda, nawet niezaznajomiona z uczeniem maszynowym osoba, jest w stanie wygenerować czytelne i interpretowalne opisy przesłanych przez siebie danych.

3 Wykonane prace wdrożeniowe

Autor niniejszej pracy brał czynny udział w projektowaniu i implementacji całego systemu jako jeden z głównych wykonawców. Prowadzone przez niego badania, opisane we wcześniejszej części pracy, miały służyć zwiększeniu konkurencyjności platformy względem innych dostępnych na rynku rozwiązań. Docelowo miało to zostać osiągnięte poprzez wdrożenie jednej z opracowanych przez autora metod głębokiej indukcji reguł do systemu i udostępnienie jej użytkownikom. Dzięki temu zyskali by oni nowe, unikalne narzędzie do eksploracji danych i odkrywania wiedzy. Jak zostało to już wcześniej wspomniane w rozdziale 4, w literaturze niemalże nieobecne są metody pozwalające na odkrywanie tak złożonych wzorców jak te zaproponowane przez autora. Co więcej, niewiele algorytmów opisanych w przeglądzie literatury



Rysunek 9.1: Zrzut ekranu formularza konfiguracyjnego algorytmu DeepRules.

oferuje solidną i łatwą w uruchomieniu implementację. Wszystko to pozwoliłoby wyróżnić produkt podmiotu zatrudniającego na tle innych istniejących rozwiązań, zwiększając jego atrakcyjność.

Na podstawie przeprowadzonych i opisanych w rozdziale 7 badań, podjęto decyzję o wdrożeniu do platformy RuleMiner algorytmu DeepRules. Decyzja ta podyktowana była również jego mniejszą złożonością obliczeniową względem pozostałych metod (patrz rozdział 6 podsekcja 3.5).

W pierwszym kroku został opracowany pakiet języka Python oferujący implementację metody DeepRules[1]. Wykorzystuje ona wewnętrznie bibliotekę NumPy[2], napisaną w głównej mierze w języku programowania C , w celu przyspieszenia obliczeń.

Następnie, pakiet ten został zintegrowany z resztą systemu RuleMiner, tak by użytkownik był w stanie generować zbiory reguł z jego użyciem, z poziomu wygodnego i intuicyjnego interfejsu graficznego aplikacji webowej. Wymagało to zarówno prac związanych z graficznym interfejsem użytkownika (frontend), jak i częścią obliczeniową (backend). Autor niniejszej dysertacji był zaangażowany w obydwa te rodzaje prac związanych z wdrożeniem algorytmu DeepRules do platformy RuleMiner. Były one prowadzone głównie w czwartym, ostatnim roku studiów doktoranckich. Zakończone zostały sukcesem i w aktualnej wersji 1.5 platformy algorytm DeepRules

[1]Link do repozytorium pakietu deeprules: `https://github.com/ruleminer/deeprules`, [dostęp 16.09.2025].

[2]Link do oficjalnej strony biblioteki NumPy: `https://numpy.org/`, [dostęp 16.09.2025].

jest dostępny dla wszystkich użytkowników. Aktualnie trwają dalsze prace rozwojowe aplikacji RuleMiner. W ramach nich planowane jest wdrożenie metody MofNRules. Wiąże się to jednak ze znacznie większymi i bardziej poważnymi zmianami w kodzie źródłowym platformy.

10 Podsumowanie i wnioski

Niniejsza praca dotyczy tematyki uczenia maszynowego, opartego na regułach oraz eksploracji danych, ze szczególnym uwzględnieniem indukcji złożonych zależności, wyrażanych w języku deskryptorów (warunków elementarnych). Głównym celem było odkrycie głębszych relacji, których standardowe algorytmy indukcji reguł, oparte na heurystykach, nie są w stanie wykryć. To z kolei może w pewnych sytuacjach przyczyniać się do poprawy interpretowalności uzyskiwanych modeli regułowych, umożliwiając uzyskanie bardziej zwięzłych i zrozumiałych opisów danych. Autor prezentuje w pracy trzy metody indukcji reguł, które umożliwiają odkrywanie złożonych zależności w danych. Każda z nich obsługuje aż trzy rodzaje danych: klasyfikacyjne, regresyjne i przeżyciowe. Opisane metody wpisują się w koncepcję głębokiego dyskretnego uczenia, opisaną w rozdziale 5. W przeciwieństwie do popularnych obecnie algorytmów subsymbolicznych, takich jak np. głębokie sieci neuronowe, te zaproponowane w pracy koncentrują się na tworzeniu zrozumiałych dla człowieka opisów danych w postaci zbiorów reguł decyzyjnych. Każda z trzech metod – ComplexConditions, MofNRules oraz DeepRules – prezentuje odmienne podejście do zagadnienia. Wszystkie jednak skupiają się na poszukiwaniu skomplikowanych zależności w danych, których zapis z użyciem prostych warunków (np. wiek $>$ 30 lub kolor = czerwony), stosowanych przez tradycyjne algorytmy regułowe, byłby nadmiernie rozbudowany i mało czytelny. Cechą wspólną wszystkich opracowanych metod jest działanie w oparciu o popularną strategię sekwencyjnego pokrywania, gdzie każda nowa reguła indukowana jest na zbiorze niepokrytych dotąd przykładów. Minimalizuje to redundancję, pozwalając na tworzenie zwięzłych zbiorów reguł, gdzie każda z nich wnosi unikalną wartość.

Algorytm ComplexConditions stanowi odświeżone podejście do idei konstruktywnej indukcji sterowanej danymi. Bazując na dobrze opisanej w literaturze metodzie RuleKit [74] dodatkowo rozszerza ją o szeroki wachlarz warunków złożonych, takich jak warunki relacji atrybutów (np. wzrost $>$ waga) lub wewnętrznych alternatyw (np. kolor $\in$ {czerwony, zielony, żółty}). Umożliwia to tworzenie bardziej zwięzłych

opisów danych w porównaniu do powszechnie stosowanych algorytmów regułowych. W efekcie może to prowadzić do zredukowania złożoności wynikowego modelu, ułatwiając tym samym jego interpretację. Jak zostało to zaprezentowane na przykładach, zastosowanie tych złożonych form warunków pozwala również opisywać relacje w danych, których opis za pomocą warunków prostych byłby niemożliwy do uzyskania lub nadmiernie skomplikowany i rozbudowany. Metoda ComplexConditions oraz wyniki badań jej zdolności klasyfikacyjnych zostały opisane przez autora w pracy zbiorowej zatytułowanej *„New Approach to Constructive Induction—Towards Deep Discrete Learning"*. Praca ta została przedstawiona na międzynarodowej konferencji DepCoS-RELCOMEX w 2023 roku.

Druga z metod, MofNRules, stanowi rozwinięcie pierwszej, wprowadzając do niej nowy rodzaj warunków: M-z-N. Pozwalają one opisywać zależności w danych, do których opisu konieczne byłoby użycie wielu reguł z warunkami prostymi. Algorytm ten działa zgodnie z ideą konstruktywnej indukcji sterowanej hipotezami, generując reguły w procesie dwóch następujących po sobie przebiegów indukcji, rozszerzając przy tym pierwotną przestrzeń cech. Wyjątkową cechą metody MofNRules jest także to, że jako jedna z nielicznych obecnych w literaturze, umożliwia tworzenie reguł zawierających w przesłankach, obok warunków M-z-N, także inne rodzaje warunków. Algorytm ten został po raz pierwszy opisany przez autora w pracy zbiorowej zatytułowanej *„Classification, regression, and survival rule induction with complex and M-of-N elementary conditions"* w 2024 roku [113]. Przedstawione w niej badania nie były jednak równie wnikliwe i szerokie jak te zaprezentowane w niniejszej dysertacji.

Trzecia i ostatnia z opisywanych metod, DeepRules, prezentuje odmienne spojrzenie na problem. Zamiast wyczerpująco przeglądać przestrzeń warunków złożonych, tak jak miało to miejsce w przypadku ComplexConditions, algorytm ten buduje rozbudowane wyrażenia logiczne w postaci CNF (koniunkcja alternatyw) i DNF (alternatywa koniunkcji). Wyrażenia te mogą być następnie upraszczane, zastępując ich poszczególne fragmenty równoważnymi logicznie warunkami złożonymi. Algorytm umożliwia w ten sposób uzyskiwanie reguł, zawierających wszystkie rodzaje deskryptorów stosowanych w poprzednich dwóch metodach (z pewnymi ograniczeniami opisanymi dokładniej w rozdziale 6 w sekcji 3), oferując przy tym mniejszą złożoność obliczeniową.

Jedną z unikalnych i istotnych cech wszystkich trzech opracowanych algorytmów jest ich zdolność do radzenia sobie z trzema popularnymi typami problemów, spoty-

kanymi w uczeniu maszynowym: klasyfikacją, regresją i analizą przeżycia. Należy zaznaczyć, że jest to niezwykle rzadkie, a w literaturze praktycznie nie występują metody o podobnych własnościach. Dodatkowo każdy z opisywanych algorytmów zapewnia obsługę wartości brakujących oraz umożliwia operowanie na danych, zawierających zarówno atrybuty nominalne, jak i numeryczne, bez konieczności stosowania ich kodowania lub dyskretyzacji. Zdolność do wykrywania złożonych zależności w danych, których większość popularnych metod mogłaby nigdy nie odnaleźć, sprawia, że opracowane algorytmy mogą stanowić użyteczne narzędzie w dziedzinie eksploracji danych i odkrywania wiedzy.

W ramach pracy przeprowadzono szeroko zakrojone badania eksperymentalne, których celem było potwierdzenie skuteczności i walidacja opracowanych metod na dużej liczbie zróżnicowanych zbiorów danych, dotyczących trzech badanych typów problemów. Uzyskane wyniki porównano także z tymi osiąganymi przez dwa referencyjne algorytmy: RuleKit oraz drzewa decyzyjne. RuleKit był już wcześniej porównywany z szerokim spektrum algorytmów indukcji reguł, działających w oparciu o różne paradygmaty [75, 191] [1]. W rezultacie zestawienie wyników z metodą RuleKit pozwala na ocenę skuteczności proponowanych rozwiązań na tle innych istniejących w literaturze metod.

Dla danych klasyfikacyjnych najlepszą metodą pod kątem jakości predykcji okazał się DeepRules, przewyższając nieznacznie pozostałe algorytmy. Pod kątem sumarycznej liczby warunków w zbiorze, wszystkie metody zaproponowane w pracy doprowadziły do mniej złożonych zbiorów niż metody referencyjne. Algorytm DeepRules wykazywał tutaj tendencję do generowania mniejszej liczby reguł, ich przesłanki były jednak znacząco dłuższe. Zawierały one przy tym sumarycznie najmniejszą liczbę warunków spośród wszystkich badanych metod. W przypadku MofNRules, liczba uzyskanych reguł była mniejsza niż dla algorytmów referencyjnych, były one również krótsze niż w przypadku RuleKit. Zbiory uzyskane z użyciem ComplexConditions miały zbliżoną do RuleKit liczbę reguł, jednak zwykle znacznie mniejszą liczbę warunków w przesłankach.

Najlepsze wyniki predykcyjne dla danych regresyjnych uzyskał ponownie algorytm DeepRules, jako jedyny przewyższając metody referencyjne. Testy statystyczne nie wykazały jednak, by była to różnica znacząca. Pozostałe metody były pod tym kątem

[1] Link do wyników porównań algorytmu RuleKit z innymi metodami indukcji reguł: `https://github.com/ruleminer/ruleminer/tree/main/reports/comparing_RuleKit_with_other_methods`

zbliżone do drzew decyzyjnych. Zarówno MofNRules, jak i DeepRules generowały, w przypadku problemu regresji, zbiory o niższej liczbie reguł oraz sumarycznej liczbie warunków niż algorytmy referencyjne. Dla metody ComplexConditions złożoność uzyskiwanych opisów była podobna jak dla RuleKit. Badania ujawniły także istotne różnice w wynikach pomiędzy dwoma wariantami metody MofNRules, stosującymi warunki „przynajmniej M-z-N" oraz „dokładnie M-z-N". Ten pierwszy uzyskiwał znacząco mniej złożone zbiory reguł zarówno pod kątem ich liczby, jak i sumarycznej liczby warunków.

Również w analizie przeżycia, jedynie DeepRules uzyskał lepsze wyniki predykcyjne niż metody referencyjne. Pozostałe były pod tym kątem zbliżone do algorytmu RuleKit. ComplexConditions wykazał tutaj silną tendencję do generowania większej liczby reguł od pozostałych dwóch metod zaproponowanych przez autora. Biorąc pod uwagę sumaryczną liczbę warunków w zbiorach, były one jednak zbliżone do tych uzyskiwanych przez RuleKit. Zarówno DeepRules, jak i MofNRules generowały na ogół mniejszą liczbę reguł niż pozostałe, jednak ich przesłanki zawierały zwykle więcej warunków. W przeciwieństwie do problemu regresji, w tym wypadku wariant „dokładnie M-z-N" okazał się prowadzić do uzyskania mniej złożonych modeli. Było to widoczne zarówno pod względem liczby reguł, jak i sumarycznej liczby warunków, gdzie jako jedyny przewyższał on referencyjny algorytm RuleKit.

Gdy kluczowe jest dla nas osiągnięcie wysokiej jakości predykcji, najlepszym wyborem będzie prawdopodobnie algorytm DeepRules. Przewyższa on pod tym względem pozostałe metody, w tym referencyjne. Warto jednak zaznaczyć, że indukowane przez niego reguły mogą zawierać dużą liczbę warunków w przesłankach, co negatywnie wpływa na ich interpretowalność. Z kolei ComplexConditions pozwala na uzyskanie najkrótszych reguł o najmniejszej liczbie warunków dla problemu klasyfikacji. Niestety, jego wyniki dla regresji i analizy przeżycia są już wyraźnie gorsze. Metoda MofNRules oferuje z kolei rozsądny kompromis pomiędzy złożonością uzyskiwanych zbiorów reguł a ich zdolnościami predykcyjnymi. Wybór odpowiedniego wariantu warunków M-z-N („dokładnie M-z-N" lub „przynajmniej M-z-N") może być jednak problematyczny, warunkując jakość uzyskanego modelu. Na podstawie przeprowadzonych badań, dla klasyfikacji oba warianty uzyskiwały podobne rezultaty, podczas gdy w przypadku regresji wariant „przynajmniej M-z-N" cechował się na ogół wyższą jakością predykcji. Z kolei dla analizy przeżycia, to wariant „dokładnie M-z-N" okazał się najlepszy pod tym kątem.

Kluczowym wnioskiem z przeprowadzonych eksperymentów jest stwierdzenie, że żadna z metod nie jest uniwersalnie najlepsza dla wszystkich typów problemów. Wyniki pokazują, że ich skuteczność jest w dużej mierze zależna od charakterystyki danych. W przypadku zbiorów, gdzie nie występują złożone wzorce, proponowane podejścia mogą czasem prowadzić do generowania nadmiernie skomplikowanych reguł. Przeprowadzone badania pokazały jednak, że opisane metody mają potencjał do generowania zbiorów reguł o mniejszej złożoności niż algorytmy referencyjne, a ich wyniki predykcyjne pozostają przy tym na ogół na zbliżonym poziomie. Świadczy to o tym, że są one w stanie opisać dane w sposób bardziej zwięzły, a przy tym równie dokładny. Na tej podstawie można podejrzewać, że są one zdolne do odkrywania pewnych głębszych zależności w danych, które są pomijane przez algorytmy referencyjne. Jest to dobrze widoczne w studiach przypadków problemów „MONK". Wszystkie z nich są możliwe do opisania z użyciem zbioru reguł w postaci DNF. Mimo to, algorytm RuleKit nie odnajduje w ich przypadku poprawnych reprezentacji klasy pozytywnej. Pokazuje to, że choć daną zależność można opisać za pomocą reguł z warunkami prostymi, nie oznacza to, że zostanie ona w praktyce odkryta przez algorytm. Dzieje się tak, ponieważ metody uczenia regułowego działają zwykle w oparciu o heurystyki, nie analizując pełnej przestrzeni rozwiązań, przez co niektóre z nich mogą zostać pominięte. Opracowane algorytmy, poprzez ukierunkowanie na poszukiwanie złożonych wzorców, zwiększają szansę na ich wykrycie, prowadząc często do bardziej zwięzłych i potencjalnie bardziej interpretowalnych opisów danych.

Na podstawie przeprowadzonych badań i ich wyników można nakreślić kilka istotnych kierunków dalszych badań. Pierwszym z nich jest opracowanie metod optymalizacji zapisu reguł, tak aby uzyskać możliwie najprostsze formuły. Jak pokazał przykład zbioru „MONK-3", algorytm może czasem odkryć tożsamą logicznie, lecz bardziej skomplikowaną postać pewnego wzorca, utrudniając tym samym jego interpretację. W przyszłości planowane jest opracowanie metod redukowania uzyskanych opisów do ich minimalnej postaci z użyciem praw rachunku zdań.

Studium przypadku zbioru danych „Ozone" pokazało także, że brak ograniczeń nałożonych na warunki relacji atrybutów może w pewnych sytuacjach prowadzić do generowania reguł niezgodnych z wiedzą domenową. Dlatego też w przyszłości konieczne będzie dodanie odpowiednich parametrów do opracowanych metod, które pozwolą na ograniczenie, które atrybuty mogą być ze sobą porównywane.

W dalszej perspektywie uwzględnienie wiedzy domenowej eksperta w procesie indukcji reguł może również okazać się wartościowe. Podobny mechanizm jest oferowany

przez algorytm RuleKit (GuideR) [162]. Pozwala to połączyć proces automatycznego odkrywania wiedzy z tą już posiadaną przez osobę przeprowadzającą analizę. Wprowadzenie takiego trybu indukcji eksperckiej mogłoby znacznie zwiększyć praktyczną wartość opracowanych metod.

Niniejsza praca została zrealizowana w ramach piątej edycji programu „Doktorat wdrożeniowy". Jej efektem było udane wdrożenie funkcjonalności indukcji reguł decyzyjnych za pomocą algorytmu DeepRules w ramach platformy analitycznej RuleMiner. Sama platforma jest publicznie dostępna pod adresem `https://app.ruleminer.ai/` (data dostępu 16.09.2025) i stanowi własność podmiotu zatrudniającego autora.

Bibliografia

[1] Agrawal, R., Imieliński, T., Swami, A. Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, strona 207–216, New York, NY, USA, 1993. Association for Computing Machinery.

[2] Agrawal, R., Srikant, R. Fast algorithms for mining association rules in large databases. *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, strona 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] Akil, M. F., Ertuğrul, Ö. F. *Shallow Learning vs. Deep Learning in Image Processing*, strony 115–129. Springer Nature Switzerland, Cham, 2024.

[4] Alemdar, H., Leroy, V., Prost-Boucle, A., Pétrot, F. Ternary neural networks for resource-efficient ai applications. *2017 International Joint Conference on Neural Networks (IJCNN)*, strony 2547–2554, 2016.

[5] Alkhatib, A., Boström, H., Vazirgiannis, M. Explaining predictions by characteristic rules. Amini, M.-R., Canu, S., Fischer, A., Guns, T., Kralj Novak, P., Tsoumakas, G., redaktorzy, *Machine Learning and Knowledge Discovery in Databases*, strony 389–403, Cham, 2023. Springer International Publishing.

[6] Barakat, N., Diederich, J. Learning-based rule-extraction from support vector machines. *The 14th International Conference on Computer Theory and applications ICCTA'2004*, strony 1–8, Alexandria, Egypt, 2004.

[7] Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., Herrera, F. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020.

[8] Beck, F., Fürnkranz, J. An empirical investigation into deep and shallow rule learning. *Frontiers in Artificial Intelligence*, 4, 2021.

[9] Beck, F., Fürnkranz, J., Huynh, V. Q. P. Generalizing conjunctive and disjunctive rule learning to learning m-of-n concepts. Brejová, B., Ciencialová, L., Holena, M., Jajcay, R., Jajcayová, T., Lexa, M., Mráz, F., Pardubská, D., Plátek, M., redaktorzy, *Proceedings of the 23rd Conference Information Technologies - Applications and Theory (ITAT 2023), Tatranské Matliare, Slovakia, September 22-26, 2023*, wolumen 3498 serii *CEUR Workshop Proceedings*, strony 8–13. CEUR-WS.org, 2023.

[10] Beck, F., Fürnkranz, J., Huynh, V. Q. P. Layerwise learning of mixed conjunctive and disjunctive rule sets. *International Joint Conference on Rules and Reasoning*, strony 95–109. Springer, 2023.

[11] Beck, F., Fürnkranz, J., Huynh, V. Q. P. Layerwise learning of mixed conjunctive and disjunctive rule sets. Fensel, A., Ozaki, A., Roman, D., Soylu, A., redaktorzy, *Rules and Reasoning*, strony 95–109, Cham, 2023. Springer Nature Switzerland.

[12] Beck, F., Fürnkranz, J., Huynh, V. Q. P. Learning deep rule concepts as alternating boolean pattern trees. *Discovery Science: 27th International Conference, DS 2024, Pisa, Italy, October 14–16, 2024, Proceedings, Part II*, strona 50–64, Berlin, Heidelberg, 2025. Springer-Verlag.

[13] Bengio, Y. Learning deep architectures for ai. *Foundations*, 2:1–55, 01 2009.

[14] Bengio, Y., Léonard, N., Courville, A. C. Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv*, abs/1308.3432, 2013.

[15] Benjamini, Y., Hochberg, Y. Controlling the false discovery rate - a practical and powerful approach to multiple testing. *J. Royal Statist. Soc., Series B*, 57:289 – 300, 11 1995.

[16] Bertsimas, D., Dunn, J., Gibson, E., Orfanoudaki, A. Optimal survival trees. *Machine Learning*, 111(8):2951–3023, Aug 2022.

[17] Bezdek, J. C. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Advanced Applications in Pattern Recognition. Springer, New York, NY, 1981.

[18] Bissacco, A., Yang, M.-H., Soatto, S. Fast human pose estimation using appearance and motion via multi-dimensional boosting regression. *IEEE Conference on Computer Vision and Pattern Recognition, CVPR'07*, Minneapolis, MN, 2007.

[19] Blachnik, M., Duch, W. *Prototype Rules from SVM*, strony 163–182. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[20] Bloedorn, E., Michalski, R. S. The aq17-dci system for data-driven constructive induction and its application to the analysis of world economics. Raś, Z. W., Michalewicz, M., redaktorzy, *Foundations of Intelligent Systems*, strony 108–117, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[21] Bloedorn, E., Michalski, R. S., Wnek, J. Multistrategy constructive induction: Aq17-mci. *Proceedings of the Second International Workshop on Multistrategy Learning (MSL93)*, strony 188–203, Harpers Ferry, WV, may 1993. Morgan Kaufmann.

[22] Bloedorn, E., Michalski, R. S., Wnek, J. Multistrategy constructive induction: Aq17-mci. *Proceedings of the 2nd International Workshop on Multistrategy Learning*, Harpers Ferry, WV, 1993.

[23] Bohanec, M., Rajkovic, V. Knowledge-based explanation in multi-attribute decision making. *Produktivnost*, 01 1989.

[24] Bradshaw, G. F., Langley, P. W., Simon, H. A. Studying scientific discovery by computer simulation. *Science*, 222(4627):971–975, 1983.

[25] Breiman, L. Bagging predictors. *Machine Learning*, 24(2):123–140, Sier. 1996.

[26] Breiman, L. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[27] Breiman, L., Friedman, J., Stone, C., Olshen, R. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.

[28] Brodersen, K. H., Ong, C. S., Stephan, K. E., Buhmann, J. M. The balanced accuracy and its posterior distribution. *2010 20th International Conference on Pattern Recognition*, strony 3121–3124, 2010.

[29] Bruha, I. Quality of decision rules: Definitions and classification schemes for multiple rules. Nakhaeizadeh, G., Taylor, C. C., redaktorzy, *Machine Learning and Statistics, The Interface*, strony 107–131. Wiley, New York, USA, 1997.

[30] Cabitza, F., Campagner, A., Natali, C., Parimbelli, E., Ronzio, L., Cameli, M. Painting the black box white: Experimental findings from applying xai to an ecg reading setting. *Machine Learning and Knowledge Extraction*, 5(1):269–286, 2023.

[31] Callan, J. P., Utgoff, P. E. Constructive induction on domain information. Dean, T. L., McKeown, K. R., redaktorzy, *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, 1991, Volume 2*, strony 614–619. AAAI Press / The MIT Press, 1991.

[32] Chan, K. C. C., Au, W.-H. Mining fuzzy association rules. *Proceedings of the Sixth International Conference on Information and Knowledge Management*, CIKM '97, strona 209–215, New York, NY, USA, 1997. Association for Computing Machinery.

[33] Chen, T., Guestrin, C. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, strona 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[34] Cheng, Y., Yu, F. X., Feris, R. S., Kumar, S., Choudhary, A., Chang, S.-F. An exploration of parameter redundancy in deep networks with circulant projections. *2015 IEEE International Conference on Computer Vision (ICCV)*, strony 2857–2865, 2015.

[35] Chi, C.-C., Jiang, J.-H. R. Logic synthesis of binarized neural networks for efficient circuit implementation. *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, strony 1–7, 2018.

[36] Clark, P., Niblett, T. The cn2 induction algorithm. *Machine learning*, 3:261–283, 1989.

[37] Cohen, W. Fast effective rule induction. *Twelfth International Conference on Machine Learning: 1995*, 95, 10 2000.

[38] Cohen, W. W. Fast effective rule induction. *In Proceedings of the Twelfth International Conference on Machine Learning*, strony 115–123. Morgan Kaufmann, 1995.

[39] Cohen, W. W. Fast effective rule induction. *Machine learning proceedings 1995*, strony 115–123. Elsevier, 1995.

[40] Cohen, W. W., Singer, Y. A simple, fast, and effective rule learner. *AAAI/IAAI*, 99(335-342):3, 1999.

[41] Cordon, O., Herrera, F., Villar, P. Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base. *IEEE Transactions on Fuzzy Systems*, 9(4):667–674, 2001.

[42] Cortes, C., Vapnik, V. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[43] Courbariaux, M., Bengio, Y., David, J.-P. Binaryconnect: training deep neural networks with binary weights during propagations. *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, strona 3123–3131, Cambridge, MA, USA, 2015. MIT Press.

[44] Craven, M. W., Shavlik, J. W. Using sampling and queries to extract rules from trained neural networks. Cohen, W. W., Hirsh, H., redaktorzy, *Machine Learning Proceedings 1994*, strony 37–45. Morgan Kaufmann, San Francisco (CA), 1994.

[45] Dargan, S., Kumar, M., Ayyagari, M. R., Kumar, G. A survey of deep learning and its applications: A new paradigm to machine learning. *Archives of Computational Methods in Engineering*, 27:1071 – 1092, 2019.

[46] Dash, S., Gunluk, O., Wei, D. Boolean decision rules via column generation. *Advances in neural information processing systems*, 31, 2018.

[47] Delalleau, O., Bengio, Y. Shallow vs. deep sum-product networks. Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K., redaktorzy, *Advances in Neural Information Processing Systems*, wolumen 24. Curran Associates, Inc., 2011.

[48] Dembczyński, K., Kotłowski, W., Słowiński, R. Ender: a statistical framework for boosting decision rules. *Data Mining and Knowledge Discovery*, 21:52–90, 2010.

[49] Demšar, J. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, Gru. 2006.

[50] Deshmukh, A., Morghade, J., Khera, A., Bajaj, P. Binary neural networks – a cmos design approach. Khosla, R., Howlett, R. J., Jain, L. C., redaktorzy, *Knowledge-Based Intelligent Information and Engineering Systems*, strony 1291–1296, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[51] Drastal, G., Czako, G., Raatz, S. Induction in an abstraction space: a form of constructive induction. *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'89, strona 708–712, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[52] Dries, A., De Raedt, L., Nijssen, S. Mining predictive k-cnf expressions. *IEEE Transactions on Knowledge and Data Engineering*, 22(5):743–748, 2010.

[53] Dubois, D., Prade, H. What are fuzzy rules and how to use them. *Fuzzy Sets and Systems*, 84(2):169–185, 1996.

[54] Duch, W., Adamczak, R., Grabczewski, K. A new methodology of extraction, optimization and application of crisp and fuzzy logical rules. *IEEE Transactions on Neural Networks*, 12(2):277–306, 2001.

[55] Duch, W., Grudzinski, K. Prototype based rules-a new way to understand the data. *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, wolumen 3, strony 1858–1863 vol.3, 2001.

[56] Duch, W., Grudzinski, K. Prototype based rules-a new way to understand the data. *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, wolumen 3, strony 1858–1863 vol.3, 2001.

[57] Duch, W., Jankowski, N., Grąbczewski, K., Adamczak, R. Optimization and interpretation of rule-based classifiers. *Intelligent Information Systems*, strony 1–13, Heidelberg, 2000. Physica-Verlag HD.

[58] Ebrahimnejad, A., Verdegay, J. L. *Fuzzy Sets-Based Methods and Techniques for Modern Analytics*. Studies in Fuzziness and Soft Computing. Springer, Cham, 2018.

[59] Elio, R., Watanabe, L. An incremental deductive strategy for controlling constructive induction in learning from examples. *Machine Learning*, 7:7–44, 07 1991.

[60] Entekhabi, Z., Shamsinejadbabki, P. Farm: Fuzzy action rule mining. *International Journal of Advanced Computer Science and Applications*, 9(1), 2018.

[61] Falkenhainer, B. C., Michalski, R. S. 6 - integrating quantitative and qualitative discovery in the abacus system. Kodratoff, Y., Michalski, R. S., redaktorzy, *Machine Learning*, strony 153–190. Morgan Kaufmann, San Francisco (CA), 1990.

[62] Fix, E., Hodges, J. L. Discriminatory analysis - nonparametric discrimination: Consistency properties. *International Statistical Review*, 57:238, 1989.

[63] Freund, Y., Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[64] Fürnkranz, J. Separate-and-conquer rule learning. *Artif. Intell. Rev.*, 13(1):3–54, Luty 1999.

[65] Fürnkranz, J., Flach, P. A. Roc 'n'rule learning—towards a better understanding of covering algorithms. *Machine learning*, 58:39–77, 2005.

[66] Fürnkranz, J., Gamberger, D., Lavrač, N. *Foundations of Rule Learning*. Cognitive Technologies. Springer Berlin Heidelberg, 2012.

[67] Fürnkranz, J., Kliegr, T., Paulheim, H. On cognitive preferences and the plausibility of rule-based models. *Mach. Learn.*, 109(4):853–898, Kwi. 2020.

[68] Fürnkranz, J., Kliegr, T., Paulheim, H. On cognitive preferences and the plausibility of rule-based models. *Machine Learning*, 109(4):853–898, dec 2019.

[69] Garey, M. R., Johnson, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[70] Garg, K., Kumar, D. Comparing the performance of frequent pattern mining algorithms. *International Journal of Computer Applications*, 69:21–28, 05 2013.

[71] Gerds, T., Schumacher, M. Consistent estimation of the expected brier score in general survival models with right-censored event times. *Biometrical journal. Biometrische Zeitschrift*, 48:1029–40, 12 2006.

[72] Goodfellow, I., Bengio, Y., Courville, A. *Deep Learning*. MIT Press, 2016.

[73] Gudys, A. *RuleKit documentation*. ADAA-POLSL, 2023.

[74] Gudyś, A., Sikora, M., Wróbel, Ł. RuleKit: A comprehensive suite for rule-based learning. *Knowledge-Based Systems*, 194:105480, 01 2020.

[75] Gudys, A., Sikora, M., Wróbel, L. Rulekit: A comprehensive suite for rule-based learning. *Knowl. Based Syst.*, 194:105480, 2020.

[76] Gudyś, A., Sikora, M., Łukasz Wróbel. Separate and conquer heuristic allows robust mining of contrast sets in classification, regression, and survival data. *Expert Systems with Applications*, 248:123376, 2024.

[77] Gunning, D., Aha, D. Darpa's explainable artificial intelligence (xai) program. *AI magazine*, 40(2):44–58, 2019.

[78] Han, J., Kamber, M., Pei, J. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2011.

[79] Han, J., Pei, J., Yin, Y., Mao, R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.

[80] Han, S., Pool, J., Tran, J., Dally, W. J. Learning both weights and connections for efficient neural network. *Neural Information Processing Systems*, 2015.

[81] Håstad, J. Almost optimal lower bounds for small depth circuits. *Symposium on the Theory of Computing*, 1986.

[82] Hastie, T., Tibshirani, R., Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009.

[83] Hu, Y.-J. *Constructive Induction: Covering Attribute Spectrum*, strony 257–272. Springer US, Boston, MA, 1998.

[84] Hutchinson, R. A., Liu, L.-P., Dietterich, T. G. Incorporating boosted regression trees into ecological latent variable models. *AAAI'11*, strony 1343–1348, San Francisco, CA, 2011.

[85] Huth, M., Ryan, M. *Logic in Computer Science : Modelling and Reasoning about Systems*. Cambridge University Press, 2004.

[86] Huynh, V. Q. P., Fürnkranz, J., Beck, F. Efficient learning of large sets of locally optimal classification rules. *Machine Learning*, 112(2):571–610, Feb 2023.

[87] Hühn, J., Hüllermeier, E. Furia: An algorithm for unordered fuzzy rule induction. *Data Min. Knowl. Discov.*, 19:293–319, 12 2009.

[88] I., B. Quality of decision rules: Definitions and classification schemes for multiple rules. *Machine Learning and Statistics: The Interface. John Wiley*, strona 107–131, 1997.

[89] i, S., Herrera, F. An extension on "statistical comparisons of classifiers over multiple data sets"for all pairwise comparisons. *Journal of Machine Learning Research - JMLR*, 9, 12 2008.

[90] Ilkou, E., Koutraki, M. Symbolic vs sub-symbolic ai methods: Friends or enemies? *International Conference on Information and Knowledge Management*, 2020.

[91] Izui, Y., Pentland, A. Analysis of neural networks with redundancy. *Neural Computation*, 2(2):226–238, 1990.

[92] Jiang, Q., Abidi, S. S. R. From clusters to rules: A hybrid framework for generalized symbolic rule induction. Yeung, D. S., Liu, Z.-Q., Wang, X.-Z., Yan, H., redaktorzy, *Advances in Machine Learning and Cybernetics*, strony 219–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[93] Jordan, M. I., Mitchell, T. M. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[94] Katzir, L., Elidan, G., El-Yaniv, R. Net-dnf: Effective deep modeling of tabular data. *International Conference on Learning Representations*, 2021.

[95] Kim, D., Lee, J. Instance-based method to extract rules from neural networks. Dorffner, G., Bischof, H., Hornik, K., redaktorzy, *Artificial Neural Networks — ICANN 2001*, strony 1193–1198, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[96] Kokar, M. Determining arguments of invariant functional descriptions. *Machine Learning*, 1:403–422, 12 1986.

[97] Krizhevsky, A., Sutskever, I., Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, Maj 2017.

[98] Kusters, R., Kim, Y., Collery, M., de Sainte Marie, C., Gupta, S. Differentiable rule induction with learned relational features. *International Workshop on Neural-Symbolic Learning and Reasoning*, 2022.

[99] Lal, G. R., Mithal, V. Nn2rules: Extracting rule list from neural networks, 2022.

[100] Lange, N. *Case Studies in Biometry*. Number t. 1 serii A Wiley-interscience publication. Wiley, 1994.

[101] Larson, J., Michalski, R. S. Inductive inference of vl decision rules. *ACM SIGART Bulletin*, (63):38–44, 1977.

[102] Le Roux, N., Bengio, Y. Deep belief networks are compact universal approximators. *Neural Computation*, 22(8):2192–2207, 2010.

[103] LeBlanc, M., Crowley, J. J. Survival trees by goodness of split. *Journal of the American Statistical Association*, 88:457–467, 1993.

[104] LeCun, Y., Bengio, Y., Hinton, G. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[105] Leo, J., Pun, K., Rendell, L. *Genetic Plans and the Probabilistic Learning System: Synthesis and Results*. Number nry 1211-1217 serii A Evaluation of Two Data Dictionary Directories. Department of Computer Science, University of Illinois at Urbana-Champaign, 1985.

[106] Li, Y., Hu, Z., Cai, Y., Zhang, W. Support vector based prototype selection method for nearest neighbor rules. Wang, L., Chen, K., Ong, Y. S., redaktorzy, *Advances in Natural Computation*, strony 528–535, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[107] Mahajan, P., Uddin, S., Hajati, F., Moni, M. Ensemble learning for disease prediction: A review. *Healthcare*, 11(12):1808, 2023.

[108] Mangalampalli, A., Pudi, V. Fuzzy association rule mining algorithm for fast and efficient performance on very large datasets. *2009 IEEE International Conference on Fuzzy Systems*, strony 1163–1168, 2009.

[109] Mantel, N. Evaluation of survival data and two new rank order statistics arising in its consideration. *Cancer Chemotherapy Reports*, 50(3):163–170, Mar 1966.

[110] Margot, V., Baudry, J.-P., Guilloux, F., Wintenberger, O. Rule induction partitioning estimator. Perner, P., redaktor, *Machine Learning and Data Mining in Pattern Recognition*, strony 288–301, Cham, 2018. Springer International Publishing.

[111] Martens, D., Huysmans, J., Setiono, R., Vanthienen, J., Baesens, B. *Rule Extraction from Support Vector Machines: An Overview of Issues and Application in Credit Scoring*, strony 33–63. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[112] Maszczyk, C., Kozielski, M., Sikora, M. Rule-based approximation of black-box classifiers for tabular data to generate global and local explanations. *2022 17th Conference on Computer Science and Intelligence Systems (FedCSIS)*, strony 89–92, 2022.

[113] Maszczyk, C., Sikora, M., Wróbel, Classification, regression, and survival rule induction with complex and m-of-n elementary conditions. *Machine Learning and Knowledge Extraction*, 6:554–579, 03 2024.

[114] Matheus, C., Rendell, L. Constructive induction on decision trees. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 12 1998.

[115] Mendel, J. *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions, 2nd Edition*. Springer International Publishing, 2017.

[116] Michalak, M. Induction of lda oblique decision rules. *Proc. of the Intl. Conf. on Advances In Information Processing And Communication Technology - IPCT 2014.*, strony 21–25, 01 2014.

[117] Michalski, R., Mozetic, I., Hong, J., Lavrac, N. The aq15 inductive learning system: An overview and experiments (intelligent systems group report). Intelligent systems group report, University of Illinois, Department of Computer Science, Urbana-Champaign, 1986.

[118] Michalski, R. S. Aqval/1–computer implementation of a variable-valued logic system vl1 and examples of its application to pattern recognition. *Proceedings of the First International Joint Conference on Pattern Recognition*, strony 3–17, Washington, DC, october 1973.

[119] Michalski, R. S. Discovering classification rules using variable-valued logic system vl_1. Nilsson, N. J., redaktor, *Proceedings of the 3rd International Joint Conference on Artificial Intelligence. Standford, CA, USA, August 20-23, 1973*, strony 162–172. William Kaufmann, 1973.

[120] Mienye, I. D., Sun, Y. A survey of ensemble learning: Concepts, algorithms, applications, and prospects. *IEEE Access*, 10:99129–99149, 2022.

[121] Mitchell, H. B. *Ensemble Learning*, strony 125–142. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[122] Mitchell, T. M. Version spaces: a candidate elimination approach to rule learning. *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'77, strona 305–310, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

[123] Mooney, R. J. Encouraging experimental results on learning cnf. *Machine Learning*, 19:79–92, 1995.

[124] Muggleton, S. Duce, an oracle-based approach to constructive induction. *Proceedings of the 10th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'87, strona 287–292, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[125] Muggleton, S., Santos, J., Tamaddoni-Nezhad, A. Progolem: A system based on relative minimal generalisation. De Raedt, L., redaktor, *Inductive*

Logic Programming, strony 131–148, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[126] Murphy, P. M., Pazzani, M. J. Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. Birnbaum, L. A., Collins, G. C., redaktorzy, *Machine Learning Proceedings 1991*, strony 183–187. Morgan Kaufmann, San Francisco (CA), 1991.

[127] Murthy, S. K., Kasif, S., Salzberg, S. A system for induction of oblique decision trees. *J. Artif. Int. Res.*, 2(1):1–32, Sier. 1994.

[128] Muselli, M. Switching neural networks: A new connectionist model for classification. Apolloni, B., Marinaro, M., Nicosia, G., Tagliaferri, R., redaktorzy, *Neural Nets*, strony 23–30, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[129] Muselli, M., Quarati, A. Reconstructing positive boolean functions with shadow clustering. *Proceedings of the 2005 European Conference on Circuit Theory and Design, 2005.*, wolumen 3, strony III/377–III/380 vol. 3, 2005.

[130] Muñoz, L., Trujillo, L., Silva, S. Transfer learning in constructive induction with genetic programming. *Genetic Programming and Evolvable Machines*, 21, 12 2020.

[131] Napierala, K., Stefanowski, J. Bracid: A comprehensive approach to learning rules from imbalanced data. *Journal of Intelligent Information Systems*, 39, 10 2012.

[132] Nomura, H., Hayashi, I., Wakami, N. A learning method of fuzzy inference rules by descent method. *[1992 Proceedings] IEEE International Conference on Fuzzy Systems*, strony 203–210, 1992.

[133] Odense, S., d'Avila Garcez, A. Extracting m of n rules from restricted boltzmann machines. Lintas, A., Rovetta, S., Verschure, P. F., Villa, A. E., redaktorzy, *Artificial Neural Networks and Machine Learning – ICANN 2017*, strony 120–127, Cham, 2017. Springer International Publishing.

[134] Pascanu, R., Montúfar, G., Bengio, Y. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv: Learning*, 2013.

[135] Pierrard, R., Poli, J.-P., Hudelot, C. A fuzzy close algorithm for mining fuzzy association rules. Medina, J., Ojeda-Aciego, M., Verdegay, J. L., Pelta, D. A., Cabrera, I. P., Bouchon-Meunier, B., Yager, R. R., redaktorzy, *Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Foundations*, strony 88–99, Cham, 2018. Springer International Publishing.

[136] Pittman, S. J., Brown, K. A. Multi-scale approach for predicting fish species distributions across coral reef seascapes. *PLoS ONE*, 6:e20583, 2011.

[137] Plotkin, G. D. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.

[138] Prajapati, S., Fernandez, E. Performance evaluation of membership function on fuzzy logic model for solar pv array. *2020 IEEE International Conference on Computing, Power and Communication Technologies (GUCON)*, strony 609–613, 2020.

[139] Qiao, L., Wang, W., Lin, B. Learning accurate and interpretable decision rule sets from neural networks. *AAAI Conference on Artificial Intelligence*, 2021.

[140] Qiao, L., Wang, W., Lin, B. Alternative formulations of decision rule learning from neural networks. *Machine Learning and Knowledge Extraction*, 5(3):937–956, 2023.

[141] Qin, H., Gong, R., Liu, X., Bai, X., Song, J., Sebe, N. Binary neural networks: A survey. *Pattern Recognition*, 105:107281, 2020.

[142] Quinlan, J. R. Induction of decision trees. *Machine Learning*, 1(1):81–106, Mar 1986.

[143] Quinlan, J. R. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, Aug 1990.

[144] Quinlan, R. *C4.5: Programs for Machine Learning*, wolumen 1. Morgan Kaufmann Publishers, Inc., 1993.

[145] Raś, Z. W., Dardzińska, A., Liu, X. System adred for discovering rules based on hyperplanes. *Engineering Applications of Artificial Intelligence*, 17(4):401–406, 2004.

[146] Raś, Z. W., Wieczorkowska, A. Action-rules: How to increase profit of a company. Zighed, D. A., Komorowski, J., Żytkow, J., redaktorzy, *Principles of Data Mining and Knowledge Discovery*, strony 587–592, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[147] Rijnbeek, P. R., Kors, J. A. Finding a short and accurate decision rule in disjunctive normal form by exhaustive search. *Machine Learning*, 80(1):33–62, Jul 2010.

[148] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain [j]. *Psychol. Review*, 65:386 – 408, 11 1958.

[149] Rudin, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, May 2019.

[150] Rullo, P., Cumbo, C., Policicchio, V. L. Learning rules with negation for text categorization. *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, strona 409–416, New York, NY, USA, 2007. Association for Computing Machinery.

[151] Saad, E. W., Wunsch, D. C. Neural network explanation using inversion. *Neural Networks*, 20(1):78–93, 2007.

[152] Salzberg, S. L. C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Machine Learning*, 16(3):235–240, Sep 1994.

[153] Schlimmer, J. C., Granger, R. Incremental learning from noisy data. *Machine Learning*, 1:317–354, 2004.

[154] Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks*, 61, 04 2014.

[155] Schumacher, M., Graf, E., Gerds, T. How to assess prognostic models for survival data: A case study in oncology. *Methods of information in medicine*, 42:564–71, 02 2003.

[156] Setiono, R. Extracting m-of-n rules from trained neural networks. *IEEE Transactions on Neural Networks*, 11(2):512–519, 2000.

[157] Setiono, R. Extracting m-of-n rules from trained neural networks. *IEEE Transactions on Neural Networks*, 11(2):512–519, 2000.

[158] Setiono, R., Liu, H. Understanding neural networks via rule extraction. *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, strona 480–485, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[159] Shavlik, J. W., Mooney, R. J., Towell, G. G. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2):111–143, Mar 1991.

[160] Sidhu, S., Meena, U. K., Nawani, A., Gupta, H., Thakur, N. Fp growth algorithm implementation. *International Journal of Computer Applications*, 93(8), 2014.

[161] Sikora, M., Gudyś, A. Chira—convex hull based iterative algorithm of rules aggregation. *Fundamenta Informaticae*, 123(2):143–170, 2013.

[162] Sikora, M., Gudyś, A., Wróbel, Ł. GuideR: a guided separate-and-conquer rule learning in classification, regression, and survival settings. *Knowledge-Based Systems*, 173:1–14, 2019.

[163] Sikora, M., Matyszok, P., Wróbel, Ł. SCARI: Separate and conquer algorithm for action rules and recommendations induction. *Information Sciences*, 2022.

[164] Sikora, M., Wróbel, L. Data-driven adaptive selection of rule quality measures for improving rule induction and filtration algorithms. *Int. J. Gen. Syst.*, 42(6):594–613, 2013.

[165] Singhal, N., Himanshu. A review on knowledge discovery from databases. Mallick, P. K., Bhoi, A. K., González-Briones, A., Pattnaik, P. K., redaktorzy, *Electronic Systems and Intelligent Computing*, strony 457–464, Singapore, 2022. Springer Nature Singapore.

[166] Sivaram, A., Venkatasubramanian, V. Xai-meg: Combining symbolic ai and machine learning to generate first-principles models and causal explanations. *AIChE Journal*, 68, 03 2022.

[167] Stefanowski, J. Rough set based rule induction techniques for classification problems. *6th European Congress of Intelligent Techniques and Soft Computing*, strony 107–119, 1998.

[168] Stefanowski, J. The bagging and n 2-classifiers based on rules induced by modlem. *International Conference on Rough Sets and Current Trends in Computing*, strony 488–497. Springer, 2004.

[169] Tajbakhsh, A., Rahmati, M., Mirzaei, A. Intrusion detection using fuzzy association rules. *Applied Soft Computing*, 9(2):462–469, 2009.

[170] Towell, G. G., Shavlik, J. W. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, Oct 1993.

[171] Towell, G. G., Shavlik, J. W. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, Oct 1993.

[172] Tsay, L.-S., Raś, Z. W. Action rules discovery system dear_3. Esposito, F., Raś, Z. W., Malerba, D., Semeraro, G., redaktorzy, *Foundations of Intelligent Systems*, strony 483–492, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[173] Tsukimoto, H. Extracting rules from trained neural networks. *IEEE Transactions on Neural Networks*, 11(2):377–389, 2000.

[174] Uno, T., Asai, T., Uchida, Y., Arimura, H. An efficient algorithm for enumerating closed patterns in transaction databases. Suzuki, E., Arikawa, S., redaktorzy, *Discovery Science*, strony 16–31, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[175] Unwin, A., Kleinman, K. The iris data set: In search of the source of virginica. *Significance*, 18, 2021.

[176] Urquhart, A. Hard examples for resolution. *J. ACM*, 34(1):209–219, Sty. 1987.

[177] Venturini, G. Sia: A supervised inductive algorithm with genetic search for learning attributes based concepts. Brazdil, P. B., redaktor, *Machine Learning: ECML-93*, strony 280–296, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[178] Vouk, B., Guid, M., Robnik-Šikonja, M. Feature construction using explanations of individual predictions. *Engineering Applications of Artificial Intelligence*, 120:105823, 2023.

[179] Wang, L.-X., Mendel, J. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6):1414–1427, 1992.

[180] Wang, Q., Ma, Y., Zhao, K., Tian, Y. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 9, 04 2022.

[181] Webb, G. Efficient search for association rules. *Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 09 2000.

[182] Weiss, S. M., Indurkhya, N. Lightweight rule induction. *Proceedings of the seventeenth international conference on machine learning*, strony 1135–1142, 2000.

[183] Wickramarachchi, D., Robertson, B., Reale, M., Price, C., Brown, J. Hhcart: An oblique decision tree. *Computational Statistics & Data Analysis*, 96:12–23, 2016.

[184] Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[185] Wilson, D. R., Martinez, T. R. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286, Mar 2000.

[186] Wnek, J. MONK's Problems. UCI Machine Learning Repository, 1992.

[187] Wnek, J., Michalski, R. S. Discovering representation space transformations for learning concept descriptions combining dnf and m-of-n rules. *Working Notes of the ML-COLT'94 Workshop on Constructive Induction and Change of Representation*, New Brunswick, NJ, July 1994.

[188] Wnek, J., Michalski, R. S. Hypothesis-driven constructive induction in aq17-hci: A method and experiments. *Machine Learning*, 14:139–168, 1994.

[189] Wojtusiak, J., Michalski, R. S., Kaufman, K. A., Pietrzykowski, J. The aq21 natural induction program for pattern discovery: initial version and its novel features. *2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, strony 523–526. IEEE, 2006.

[190] Wolpert, D., Macready, W. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[191] Wróbel, L., Gudys, A., Sikora, M. Learning rule sets from survival data. *BMC Bioinform.*, 18(1):285:1–285:13, 2017.

[192] Wróbel, L., Sikora, M., Michalak, M. Rule quality measures settings in classification, regression and survival rule induction - an empirical approach. *Fundam. Informaticae*, 149(4):419–449, 2016.

[193] Wróbel, Ł., Sikora, M., Michalak, M. Rule quality measures settings in classification, regression and survival rule induction—an empirical approach. *Fundamenta Informaticae*, 149(4):419–449, 2016.

[194] Yang, Y., Morillo, I. G., Hospedales, T. M. Deep neural decision trees. *ArXiv*, abs/1806.06988, 2018.

[195] Zadeh, L. A. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.

[196] Zhang, Q., Wang, T. Deep learning for exploring landslides with remote sensing and geo-environmental data: Frameworks, progress, challenges, and opportunities. *Remote Sensing*, 16(8):1344, 2024.

[197] Zheng, Z. Constructing nominal x-of-n attributes. *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, strona 1064–1070, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[198] Zhou, Y., Zhou, Z., Hooker, G. Approximation trees: statistical reproducibility in model distillation. *Data Mining and Knowledge Discovery*, 38(5):3308–3346, Sep 2024.

[199] Zimmerman, H. *Fuzzy Set Theory and Its Applications*. Allied Publishers, 1996.