

INSTYTUT PODSTAW INFORMATYKI
POLSKIEJ AKADEMII NAUK

Michał Dramiński

Algorytm indukcji reguł decyzyjnych w
problemach klasyfikacji i wyboru cech w
zadaniach wysokowymiarowych

Rozprawa doktorska
przygotowana pod kierunkiem
prof. dr hab. inż. Jacka Koronackiego



WARSZAWA 2007

Niniejsza wersja rozprawy uwzględnia szereg drobnych zmian związanych z uwagami recenzentów: prof. dr hab. Włodzisława Ducha oraz dr hab. Jerzego W. Grzymały-Busse. Pragnę wyrazić im serdeczne podziękowania za wnikliwe przeczytanie pracy oraz wszystkie zgłoszone uwagi, pozwoliło to w konsekwencji na różne drobne poprawki, a tym samym na podniesienie wartości oraz klarowności tekstu.

Dziękuję również dr hab. Janowi Mielniczkowi oraz promotorowi prof. dr hab. inż. Jackowi Koronackiemu za wszystkie uwagi oraz niezwykle serdeczną i owocną współpracę.

1. Streszczenie

Praca skupia się na przedstawieniu dwóch zagadnień. Pierwszym z nich jest opis autorskiego algorytmu ADX (Apriori Decision rules eXtraction) służącego do indukcji reguł decyzyjnych. Wygenerowany zestaw reguł w czytelny sposób opisuje dane użyte w trakcie uczenia oraz służy do późniejszej klasyfikacji nowych przykładów. Czytelność i jakość opisu wpływają na zrozumienie lub odkrycie pewnych nieznanych dotąd zależności/wzorców w danych, które opisują obserwowany świat rzeczywisty. Priorytetem takiego opisu jest dostarczenie łatwo interpretowalnej wiedzy, na podstawie której można w przyszłości podejmować rozmaite decyzje. Algorytm ADX jest propozycją, która powinna spełniać wspomniane oczekiwania, gdyż: generowane reguły są krótkie/proste, można wybrać dowolny podzbiór najistotniejszych reguł (opierając się na pewnej zdefiniowanej mierze jakości), finalny zestaw reguł nie jest nacechowany zbyt silnym dopasowaniem do danych.

Algorytm ADX łączy w sobie szybkość działania porównywalną z klasyfikatorami opartymi na drzewach decyzyjnych wraz z czytelnością generowanej wiedzy typową dla klasyfikatorów regułowych. Ważnym elementem algorytmu jest liniowa zależność czasu uczenia od wielkości zbioru danych trenujących. Przy tym wielkość rozumie się tutaj jako liczbę atrybutów jak również liczbę obiektów w tabeli danych. W procesie uczenia dla każdej wartości decyzji tworzony jest osobny zbiór reguł, które charakteryzują przykłady związane z rozważaną klasą. Wynikowe reguły mają na celu pokrycie podzbioru przykładów każdej klasy w możliwie prosty i kompletny sposób, przy czym reguły te mają być możliwie najbardziej unikalne dla danej wartości decyzji. W procesie decyzyjnym biorą udział wszystkie reguły, które pokrywają przykład a ich parametry (odpowiednie pokrycia na zbiorze uczącym) są podstawą do obliczenia funkcji przynależności do rozpatrywanej klasy. Należy zaznaczyć, że wynikowe reguły nie są hierarchiczne, a o przynależności do klasy nie decyduje jedna reguła lecz pewien ich podzbiór.

W ramach pracy zaproponowano również użycie przedstawionego algorytmu indukcji reguł w zadaniach selekcji cech. Do algorytmu MCFS (służącego do selekcji istotnego podzbioru atrybutów), oryginalnie bazującego na drzewie decyzyjnym, zaproponowano użycie klasyfikatora regułowego. Algorytm MCFS (Monte Carlo Feature Selection) buduje tysiące klasyfikatorów (w podstawowej wersji C4.5) na odpowiednich próbach danych powstałych poprzez losową selekcję atrybutów. Każda taka próba jest następnie wielokrotnie losowo dzielona na zbiór uczący i testujący, które z kolei służą do budowy klasyfikatorów i oceny jakości ich predykcji.

Wysoko oceniany atrybut to taki, który pełnił istotną rolę w budowie klasyfikatorów, mających dodatkowo wysoką jakość predykcji na nieznanym zbiorze danych (zbiorze testującym). Osobną kwestią dla każdego klasyfikatora wspiera-

jącego metodę MCFS pozostaje zaproponowanie miary, która będzie wyrażać istotność atrybutu w budowie samego klasyfikatora w połączeniu z jakością klasyfikacji. Dla drzewa decyzyjnego atrybut, który bierze udział w budowie drzewa z odpowiednio wysokim np. *gain ratio*, oraz separuje dużą część przykładów można uznać za istotny. Podobne kryterium tyczy się systemu regułowego. Atrybut występujący w regułach, które mają wysokie pokrycie oraz wysoką miarę oceny jakości, można uznać za istotny.

Ogólna propozycja algorytmu MCFS, bazującego na drzewie decyzyjnym, oraz propozycja miary istotności atrybutu dla klasyfikatora C4.5 jest wynikiem współpracy, natomiast rozwiązanie w oparciu o klasyfikator ADX zostało zrealizowane w ramach rozprawy doktorskiej. Autorskim wkładem do samego algorytmu MCFS jest również propozycja wyboru istotnego zbioru atrybutów (punktu odcięcia dla rankingu atrybutów), oraz kryterium zbieżności algorytmu. Dzięki tym rozwiązaniom na bieżąco można obserwować postępującą stabilizację hierarchii wraz z kolejnymi iteracjami jak również finalnie otrzymać statystycznie istotną informację ile atrybutów z czoła listy rankingowej jest rzeczywiście istotnych.

Całkowicie nowymi elementami zaproponowanymi w pracy są proces selekcji reguł, proces wnioskowania na ich podstawie, propozycja implementacji klasyfikatora ADX w algorytmie MCFS oraz kilka rozszerzeń algorytmu MCFS. W ramach rozprawy powstała implementacja obu algorytmów ADX i MCFS w obiektowym języku JAVA w wersji 1.5, przy czym implementacja MCFS pozwala na użycie trzech różnych zaimplementowanych klasyfikatorów (ADX, C4.5, SLIQ). Implementację wyróżnia możliwość stosunkowo łatwego dodania kolejnych klasyfikatorów z uwagi na wygodny programistyczny interfejs (przygotowano odpowiednie klasy). Oba algorytmy zrealizowano w ramach rozwijanej przez autora biblioteki dmLAB (data mining laboratory). W celu ułatwienia konwersji oraz edycji wejściowych zbiorów danych zrealizowano również Edytor danych posiadający wygodny graficzny interfejs użytkownika (GUI). Edytor pozwala na import przykładów zorganizowanych w dowolnej formie tabeli danych, wydajną edycję przykładów i atrybutów, a następnie zapis w formie pliku typu 'csv' ('comma separated values' format wspierany przez np. MS Excel), 'adx' (natywny format dmLab) lub 'arff' (natywny format pakietu WEKA).

Cel pracy zdefiniowano w sekcji 2.2. Rozdział 3 opisuje zagadnienia odkrywania wiedzy z danych, natomiast w rozdziale 4 umieszczono bardzo ogólny wstęp do zagadnień z zakresu genetyki, potrzebnych do zrozumienia istoty wykorzystywanych danych. W rozdziale 5 znajduje się dokładny opis algorytmu ADX, natomiast w 6 opisano procedurę MCFS oraz przedstawiono propozycję zastąpienia drzewa decyzyjnego w MCFS klasyfikatorem ADX. Rozdział 7 zajmuje opis przeprowadzonych eksperymentów podzielony na dwie części: jedną poświęconą klasyfikacji (7.1), a drugą selekcji atrybutów z wysokowymiarowych danych (7.2). Rozdział 8 zawiera krótki opis implementacji, instrukcję uruchomienia i opis plików z parametrami do dołączonego oprogramowania. Ostatni rozdział 9 podsumowuje pracę.

Spis treści

1. Streszczenie	5
2. Wprowadzenie	9
2.1. Podstawowe pojęcia	9
2.2. Motywacja i cel pracy	14
3. Przegląd popularnych technik klasyfikacji	17
3.1. Klasyfikacja Bayesowska	17
3.2. Dyskryminacja liniowa i nieliniowa	18
3.3. Metody odległościowe	19
3.4. Drzewa decyzyjne	20
3.5. Klasyfikatory regułowe	23
3.6. Metauczenie	28
3.7. Inne metody	30
4. Od komórki do ekspresji genów	31
5. Indukcja reguł i klasyfikacja w algorytmie ADX	37
5.1. Definicje	37
5.2. Tworzenie zestawu reguł (ruleset) dla jednej klasy	39
5.2.1. Krok pierwszy — znalezienie bazy selektorów	41
5.2.2. Krok drugi — tworzenie zestawu kompleksów	42
5.2.3. Krok trzeci — sklejanie kompleksów	46
5.2.4. Krok czwarty — finalna selekcja zestawu reguł	47
5.3. Miary jakości Q dla kompleksów	50
5.4. Proces klasyfikacji	53
5.5. Obsługa wartości nieznanych	55
5.6. Obsługa sprzeczności w danych	56
5.7. Obsługa atrybutów numerycznych	57
5.8. Oszacowanie złożoności obliczeniowej algorytmu	58
5.9. Cechy algorytmu ADX	59
5.10. Oryginalność rozwiązania	60
6. Selekcja istotnych atrybutów z wysokowymiarowych danych	63
6.1. Opis problemu	63
6.2. Opis badanych wysokowymiarowych zbiorów	66
6.3. Algorytm MCFS	66
6.3.1. Miary jakości klasyfikacji	66
6.3.2. Opis procedury	69
6.3.3. Miara jakości atrybutu dla drzewa decyzyjnego	70
6.3.4. Miara jakości atrybutu dla klasyfikatora ADX	72
6.3.5. Odległość rankingów cech	73
6.3.6. Kryterium zbieżności algorytmu	75
6.3.7. Selekcja istotnego zbioru atrybutów	75
6.3.8. Problem wzajemnego maskowania atrybutów	77

6.3.9.	Problem silnie niezbalansowanych klas	79
6.4.	Opis procedur walidacyjnych i potwierdzającej	80
7.	Eksperymenty	83
7.1.	Zadania klasyfikacji	83
7.1.1.	Badanie wpływu parametrów algorytmu ADX	83
7.1.2.	Badanie złożoności obliczeniowej	94
7.1.3.	Klasyfikacja danych syntetycznych	98
7.1.4.	Klasyfikacja danych rzeczywistych	100
7.2.	Zadanie selekcji istotnych cech	106
7.2.1.	Test wiarygodności danych	106
7.2.2.	Badanie wpływu parametrów u, v	109
7.2.3.	Badanie wpływu parametrów s, t, m	110
7.2.4.	Przeciwdziałanie wzajemnemu maskowaniu cech	116
7.2.5.	Finalna selekcja istotnego zbioru atrybutów	121
7.2.6.	Badanie wpływu klasyfikatora na ranking cech w MCFS	124
7.2.7.	Walidacja metody MCFS	132
7.2.8.	Konkluzja wyników i weryfikacja biologiczna	134
8.	Implementacja ADX i MCFS w bibliotece dmLab	139
8.1.	Edytor danych	139
8.2.	Parametry wejściowe ADX	140
8.3.	Parametry wejściowe MCFS	141
9.	Podsumowanie	145
	Bibliografia	147

2. Wprowadzenie

"Even though we don't yet understand how brains perform many mental skills, we can still work toward making machines that do the same or similar things. 'Artificial intelligence' is simply the name we give to that research. But as I already pointed out, this means that the focus of that research will keep changing, since as soon as we think we understand one mystery, we have to move on to the next."

Marvin Minsky, [Minsky, 2001]

2.1. Podstawowe pojęcia

Przytoczony cytat jednego z ojców dziedziny "Sztuczna Inteligencja" (AI – Artificial Intelligence) doskonale opisuje rozwój badań w tej dziedzinie prowadzony już od początku lat 50 ubiegłego wieku. Po bardzo euforycznych zapowiedziach z okresu swoistego "romantyzmu" (panującego na początku lat 60 XX wieku w tej dziedzinie) okazało się, że zbudowanie myślącej maszyny jest do tej pory zadaniem niemożliwym do zrealizowania. Co więcej, mimo ogromnych środków i bardzo dużego postępu w rozwoju elektroniki, maszynom udaje się tylko w bardzo niewielkim stopniu naśladować bardzo ograniczone zachowania wymagające inteligencji.

Wielki triumf maszyny IBM Deep Blue nad arcymistrzem szachowym Garrim Kasparovem w 1997 roku świadczy bardziej o inteligencji twórców algorytmu i szachistów biorących udział w 700 tysiącach partii zapisanych w pamięci komputera. Samochód marki VW Tuareg zbudowany przez ośrodek ze Stanford University (którego założycielem był John McCarthy twórca terminu AI) wygrał w 2005 roku konkurs DARPA przejeżdżając 132 mile po pustyni bez udziału człowieka. Nikt wtedy jednak oczarowany przypisywaną "inteligencją" Deep Blue lub samochodu nie zapytał się później tych maszyn czy cieszą się one z osiągniętej wygranej oczekując że ich "inteligencja" wykracza poza zakres wykonywanych prac. Obie spełniły jedynie bardzo specjalistyczne zadania, do jakich zostały wcześniej przygotowane i zaprogramowane, obie nie potrafiłyby szybko zareagować na sytuację nie przewidzianą wcześniej przez ich konstruktorów. DARPA zapowiedziała zresztą osobny konkurs na pojazd potrafiący samodzielnie poruszać się w ruchu miejskim i niewątpliwie zwycięzca poprzedniego konkursu nie ma szans, ponieważ przygotowano go wyłącznie do działania w innych łatwiejszych warunkach.

Mimo niewątpliwych drobnych sukcesów w opracowywaniu maszyn naśladowujących ludzkie działanie trudno wciąż mówić o ich inteligencji. Samo pojęcie AI w świecie biznesowym nie wywołuje już tak dużej fascynacji i na poziomie komercyjnym pojawia się głównie w filmach i książkach oraz

jest domeną przedmiotów codziennego użytku niesłusznie określanych inteligentnymi jedynie z powodu implementacji w nich stosunkowo prostych reguł działania. Dziś dzięki rozwojowi komputerów możliwe jest przetwarzanie i analiza ogromnej ilości danych, co spowodowało pojawienie się bardziej adekwatnych terminów określających wykorzystanie metod rozwijanych do tej pory pod parasolem terminu AI. W ten sposób pojawiło się szereg dziedzin z przymiotnikiem *computational*, np. *'Computational Biology'* które z sukcesem wykorzystują zaawansowane metody szeroko rozumianej analizy danych w dziedzinie biologii. W myśl cytatu Marvin'a Minsky przez ponad 50 lat rozwoju dziedziny robimy wciąż tylko niewielkie kroki w celu zbudowania inteligentnej maszyny, zaś wizja jej powstania wciąż jest niezwykle odległa lub według większych sceptyków w ogóle niemożliwa.

Niewątpliwie jednak ogromnym sukcesem związanym z próbami zbudowania inteligentnych maszyn jest duży rozwój metod optymalizacji, statystyki, analizy danych, analizy języka, rozpoznawania wzorców jak również samej cybernetyki ([ASIMO], [RoboCup]). W wyniku prac prowadzonych w tych dziedzinach uzyskaliśmy narzędzia usprawniające codzienne życie: wyszukiwarki internetowe, rozpoznawanie obrazów (np. twarzy, pisma, odcisków palców), sterowanie głosem, automatyzację procesów wspomagających decyzję, zwiększenie wydajności produkcji, odkrywanie wzorców z danych itp.

Niniejsza praca jest wynikiem fascynacji autora dziedziną "Sztucznej Inteligencji", fascynacji która początkowo pojawiła się wraz poznaniem filmu "Łowca Androidów" Ridley'a Scotta czy świetnymi opowiadaniem SF Roberta Sheckleya, a później (choć wcale nie mniejsza!) została nieco zweryfikowana przez poznanie równie interesujących realnych dokonań naukowców. Opis ważnych algorytmów, które posłużyły jako inspiracja do przeprowadzenia badań związanych z rozprawą znajduje się w rozdziale 3, natomiast dalsza część wstępu będzie poświęcona krótkiemu uporządkowaniu najistotniejszych terminów pojawiających się pracy.

Temat rozprawy zalicza się do działu nazwanego eksploracją danych (ang. **data mining**) – według definicji Toma Mitchella z [Mitchell, 1999] termin ten określa: *"używanie historycznych danych do odkrywania regularności i poprawiania przyszłych decyzji"*; inne podobne i popularne definicje można odszukać jeszcze w [Frawley, 1992] i [Hand, 2001]). Eksploracja danych oznacza zatem uczenie się (wydobywanie wiedzy) na podstawie danych, by później zastosować tę nabytą wiedzę do podejmowania prawidłowych decyzji.

Wiedza, zgodnie z Encyklopedią PWN, może zostać zdefiniowana jako *"ogół wiarygodnych informacji o rzeczywistości wraz z umiejętnością ich wykorzystania"*, przy czym w ramach uczenia maszynowego możemy raczej mówić o **bazie wiedzy** czyli "zbiorze reguł, definicji i opisów faktów z określonej dziedziny, umożliwiającym logiczne wnioskowanie lub podejmowanie decyzji w komputerowym systemie doradczym".

Postęp w dziedzinie techniki komputerowej, jaki dokonał się w ostatnich kilkudziesięciu latach, pozwala na gromadzenie i przetwarzanie tak ogromnej ilości danych, że zaczyna zaspokajać potrzeby większości użytkowników. Część użytkowników wykorzystuje pozyskane informacje w dostępnych aplikacjach zawierających mechanizmy analizy rynkowej, zarządzania biznesowe-

go, i systemów wspomagania decyzji (np. cała grupa systemów analitycznych i CRM). Niestety data mining został przez to popularnie potraktowany jako synonim odkrywania wiedzy z baz danych, chociaż jest to tylko zasadniczy krok w tym procesie.

Generalnie, w procesie odkrywania wiedzy z danych (**KDD** - *Knowledge Discovery in Databases* [Fayyad *et al.*, 1996]) można wyróżnić kilka następujących po sobie faz:

Czyszczenie danych. Gromadzone dane bardzo często zawierają szum informacyjny, w postaci niepoprawnych, brakujących czy nieistotnych informacji. Takie dane niepotrzebnie zwiększają objętość przetwarzanej informacji i niekorzystnie wpływają na otrzymywane wyniki. Dlatego istotne jest odpowiednie przefiltrowanie danych, tak by dalszej obróbce podlegały tylko przydatne informacje.

Integracja danych. Bardzo często istnieją różnorodne źródła danych, które trzeba scalić w jedną całość. Związane to jest z różnymi modelami baz danych (relacyjny, obiektowy, hierarchiczny, sieciowy, itp.) lub ich reprezentacją.

Selekcja danych. Tylko istotne dane pozyskuje się z bazy danych do zadań analitycznych. Ponieważ bardzo często analiza ma opierać się tylko na kilku wyspecyfikowanych czynnikach, toteż nie ma sensu poddawać przetwarzaniu wszystkich informacji zawartych w źródłowej bazie danych.

Transformacja danych. Dane w tej fazie muszą ulec transformacji lub konsolidacji w formy aprobowane dla dalszej eksploracji (np. sprowadzenie do postaci tabeli). Jest to podyktowane pewnymi ograniczeniami związanymi z systemami uczącymi, jak format wejściowy lub wydajność.

Eksploracja danych (*data mining*) Najistotniejszym procesem całej operacji pozyskiwania wiedzy jest właśnie ta faza. Specjalnie zaprojektowane algorytmy przetwarzają dane, tak by w konsekwencji otrzymać użyteczną dla człowieka wiedzę.

Ocena wiedzy. Faza ta jest używana do identyfikacji interesujących wzorców reprezentujących wiedzę. Ponieważ produktem wcześniejszego etapu może być ogromny zbiór różnych ogólnych opisów, to należy wybrać spośród nich tylko te, które najbardziej odpowiadają naszym założeniom.

Reprezentacja wiedzy. Na tym etapie wykorzystuje się rozmaite techniki wizualizacji i reprezentacji informacji w celu przedstawienia odkrytej wiedzy użytkownikowi. Najlepsze opisy zbioru poddanego procesowi eksploracji wiedzy trzeba w sposób zrozumiały i łatwy do dalszej analizy (najlepiej w postaci graficznej) przedstawić użytkownikom.

Główne zadania dotyczące odkrywania wiedzy realizowane w fazie data mining definiujemy następująco:

Asocjacje. Analiza asocjacyjna jest popularnie używana w przypadku danych opisujących transakcje ze sprzedaży. Zadanie to skupia się na odkrywaniu reguł asocjacyjnych czyli związków pośród zestawów kupowanych pozycji lub związków pomiędzy różnymi zestawami. Zestawem pozycji określa się pewien zbiór produktów kupowanych jednocześnie. Reguła asocjacyjna przedstawiona w postaci $X \implies Y$ (c, s) interpretuje się jako: X

determinuje wystąpienie Y , oraz dodatkowo odpowiednio określa się wartość ufności c i wsparcie s . Dla zbioru D przykładów wsparciem (*support*) reguły asocjacyjnej nazywamy stosunek liczby przykładów pokrywanych przez regułę (oznaczoną jako $|D_{X,Y}|$) do liczby wszystkich przykładów. Ufność (*confidence*) reguły to stosunek liczby przykładów pokrywanych przez regułę do liczby przykładów spełniających jedynie warunek poprzednika reguły ($|D_X|$). Wsparcie określa powszechność odkrytej reguły, natomiast ufność – poziom jej wiarygodności.

$$support = \frac{|D_{X,Y}|}{|D|} = P(X \wedge Y) \quad (2.1)$$

$$confidence = \frac{|D_{X,Y}|}{|D_X|} = P(Y|X) \quad (2.2)$$

Klasyfikacja. Klasyfikacja to forma analizy zbioru przykładów o znanej przynależności do klas/kategorii i konstrukcja modelu dla każdej z klas opartego na cechach zawartych w tych danych z pominięciem cechy opisującej ową przynależność. Wynikiem procesu jest zbudowanie klasyfikatora (np. w formie drzewa decyzyjnego, czy zestawu reguł klasyfikacyjnych), potrafiącego samodzielnie rozpoznawać przynależność nowych przykładów do odpowiedniej klasy w oparciu o wyuczone wcześniej wzorce. Klasyfikatora można używać w celu lepszego zrozumienia każdej z kategorii w bazie danych lub do klasyfikacji przyszłych nowych danych. Przykładem może być klasyfikowanie choroby na podstawie symptomów występujących u pacjenta. Ten typ uczenia nazywamy nadzorowanym (lub z nauczycielem), ponieważ w fazie budowania modelu należy użyć próby danych z wyspecyfikowanymi etykietami klas, które mają podlegać późniejszemu rozpoznawaniu.

Predykcja. Metody odkrywania wiedzy mogą przewidywać również prawdopodobne wartości nieznanymi danych (pustych, błędnych itp.). Zadanie wymaga znalezienia zestawu atrybutów pozwalających na określenie wartości interesującego atrybutu. Dzięki temu można przewidywać rozkład wartości, oparty na zestawach danych podobnych do rozważanego obiektu. Na przykład, potencjalne wynagrodzenie pracownika może być przewidziane na podstawie rozkładu płac podobnych pracowników w firmie lub liczbową wartość wydobycia ropy określona na podstawie parametrów szybu naftowego. Bardzo często do realizacji tego zadania wykorzystuje się sieci neuronowe, analizę regresyjną, analizę korelacyjną i drzewa regresyjne. Termin głównie odnosi się do ciągłych wartości lecz w przypadku atrybutów nominalnych można mówić o predykcji poszczególnych wartości kategorii. Ten typ uczenia również jest nadzorowany.

Grupowanie (clustering). Ta forma analizy identyfikuje grupy naturalnie występujące w danych, gdzie grupa jest kolekcją wzajemnie podobnych obiektów. Wynikiem grupowania są grupy, które zapewniają możliwie wysokie podobieństwo obiektów wewnątrz nich, a niskie pomiędzy obiektami z różnych grup. Ten typ uczenia nazywamy nienadzorowanym ponieważ algorytm ma wskazać samodzielnie istniejące skupienia w danych.

W przypadku klasyfikacji i grupowania krytycznym zadaniem jest uprzedzenia skuteczna ekstrakcja i selekcja/wybór atrybutów. Większość algorytmów jest stosunkowo wrażliwa na przestrzeń opisu danych i dlatego zadanie konstrukcji/transformacji i oceny atrybutów jest niezwykle ważne. Ekstrakcja atrybutów polega na budowie (w dowolny sensowny sposób liniowy lub nieliniowy) nowych cech opartych na wyjściowych, tak by otrzymać nowe cechy o większych możliwościach predykcyjnych (silniej skorelowane z decyzją) lub pozwalających na wyodrębnienie bardziej spójnych grup. Wybór istotnych atrybutów służy głównie podniesieniu jakości klasyfikacji lub grupowania lecz w ramach tej pracy został właściwie sprowadzony do zadania data mining.

Należy jeszcze wyjaśnić pojęcia wnioskowania indukcyjnego (pojawiające się zresztą w tytule pracy) i dedukcyjnego. Poniższy opis w dużej mierze oparto o definicję pochodzącą ze świetnej książki [Cichosz, 2000]. Do scharakteryzowania obu typów wnioskowania można użyć następującego zapisu związku logicznej konsekwencji:

$$P \wedge W \models K, \quad (2.3)$$

gdzie P oznacza zbiór przesłanek, W wrodzoną wiedzę ucznia, K zbiór konkluzji, natomiast symbol $\models$ oznacza logiczną konsekwencję.

Def. *Formuła α jest konsekwencją logiczną (semantyczną) następującego zbioru formuł $\{\beta_1, \beta_2, \dots, \beta_m\}$, jeśli dla każdego modelu i wartościowania zmiennych, dla których jest spełniona formuła $\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_m$, jest także spełniona formuła α . Piszemy wówczas $\{\beta_1, \beta_2, \dots, \beta_m\} \models \alpha$.*

Z równania 2.3 wynika że na drodze logicznej dedukcji K wynika z P i W , czyli logiczną konsekwencją zbioru przesłanek P i wiedzy wrodzonej W jest wiedza reprezentowana przez konkluzję K . Jeżeli zatem jako zbiór przesłanek P potraktujemy informację trenującą, a za zbiór konkluzji K uznamy generowaną przez algorytm wiedzę, to równanie opisuje uczenie się oparte na wnioskowaniu dedukcyjnym. Jeśli natomiast K potraktuje się jako informację trenującą, a P jako wiedzę generowaną w procesie uczenia, to otrzymamy zapis charakteryzujący wnioskowanie indukcyjne. Powiemy wówczas że konsekwencją logiczną wiedzy wygenerowanej przez ucznia i jego wiedzy wrodzonej jest informacja trenująca. Wnioskowanie indukcyjne opiera się na własności zachowania fałszu (z fałszu K wynika fałsz P), czyli jeśli będziemy dysponować fałszywą informacją trenującą w konsekwencji otrzymamy fałszywe wnioski. Jeśli natomiast otrzymamy prawdziwą informację trenującą, to nie ma pewności że wygenerowana hipoteza będzie prawdziwa. We wnioskowaniu dedukcyjnym na podstawie własności zachowania prawdy (z prawdziwości P wynika prawdziwość K) musimy otrzymać zawsze prawdziwą hipotezę. Dodatkowo wnioskowanie indukcyjne jest obciążone, ponieważ wiedza wrodzona i informacja trenująca nie wyznaczają jednoznacznie hipotezy indukcyjnej (może być ich dowolna ilość), a o jej kształcie decydują przeważnie właściwości samego algorytmu.

Podsumowując, wnioskowanie indukcyjne opiera się na znalezieniu pewnych ogólnych prawidłowości/hipotez (budowie modelu) z przesłanek które są ich szczególnymi przypadkami (przykłady trenujące). Jeśli odkryte hipote-

zy w połączeniu z wiedzą wrodzoną ucznia (dopuszczalnie pustą) pozwalają w możliwie dużym przybliżeniu poprawnie tłumaczyć zaobserwowane fakty (zbiór przykładów trenujących) to mogą (choć wcale nie muszą) tłumaczyć również niespotkane dotąd przypadki (informacja testująca).

2.2. Motywacja i cel pracy

W dzisiejszych czasach istnieje już wiele działających algorytmów wywodzących się z dziedziny uczenia maszynowego, służących do klasyfikacji. Wciąż jednak pracuje się nad ulepszaniem znanych rozwiązań jak również opracowuje się pewne nowe algorytmy, które działają lepiej dla pewnej ściśle określonej klasy problemów. Z uwagi na praktyczne zastosowania klasyfikatorów ich dwa podstawowe kierunki rozwoju można bardzo ogólnie scharakteryzować następująco:

1. Podnoszenie jakości/wydajności klasyfikacji.
2. Zwiększanie czytelności generowanej wiedzy.

W konsekwencji przetwarzania ogromnych ilości informacji i tym samym przenoszenia części procesu decyzyjnego na maszynę, przeważnie priorytetowym zadaniem działającego systemu jest jego możliwie najwyższa jakość. Ostatnio coraz częściej właśnie od takiej decyzji zależy nasze codzienne życie np. przydział kredytu, blokada karty kredytowej w trakcie autoryzacji, dobór parametrów produkcji, przyjęcie kandydata na szkolenie pilotów, otrzymanie stosownej oferty promocyjnej etc. Osoby odpowiedzialne za projektowanie automatycznych modeli muszą dbać o możliwie maksymalną ich nieomyślność, tym większą im większe niebezpieczeństwo grozi w wyniku podjęcia złej decyzji. W przypadku wykorzystania modeli predykcyjnych w zastosowaniach czysto komercyjnych drobna wręcz poprawa modelu może przynieść ogromne oszczędności finansowe. Nawet w 100% poprawna klasyfikacja w trakcie walidacji modelu nie jest jednak w stanie zagwarantować identycznego wyniku w środowisku roboczym, toteż w większości zastosowań albo dopuszcza się incydentalnie błędne działanie (w sprawach mniej istotnych np. przydział klientów do odpowiednich segmentów) albo ostateczną decyzję pozostawia się człowiekowi.

Z uwagi na potrzebę analizy i kontroli modelu drugi kierunek oparty na generowaniu czytelnej wiedzy niekiedy przyjmuje znaczenie pierwszorzędne. Czasami pewna niechęć do wykorzystania np. sieci neuronowej bierze się właśnie z braku możliwości jej swobodnej interpretacji oraz ingerencji w stworzoną bazę wiedzy. Środowisko komercyjne coraz częściej chce kontrolować przebieg klasyfikacji i otrzymywać wynik w postaci wyraźnych wzorców w postaci reguł biznesowych. Można stwierdzić, że w bardzo wielu zastosowaniach jakość predykcji klasyfikatora jest stawiana na równi z dobrym uzasadnieniem podjętej decyzji. Brak możliwości dostarczenia takiego uzasadnienia (np. czytelnej reguły przy pomocy której zakwestionowano transakcję płatniczą) uniemożliwia wnikliwą manualną weryfikację przypadku i w wielu zastosowaniach deklasuje algorytm.

Dodatkowo potrzeba praktycznego stosowania klasyfikatorów objawiła wiele innych zadań/problemów do rozwiązywania, które również często występują jednocześnie np.:

- Działanie w oparciu o silnie zaszumione dane. W bardzo wielu przypadkach dane są nieidealnym odzwierciedleniem nieidealnej rzeczywistości. Skutkuje to częstymi błędami, brakami czy sprzecznościami w danych, a przyczyną może być: błąd w oprogramowaniu lub projekcie, błąd w manualnym uzupełnianiu danych, źle zdefiniowane tabele lub typy pól, błędny pomiar, atrybut wypełniony losowymi wartościami etc. Algorytm uczenia musi być odporny na większość z tych problemów i powinien w miarę możliwości opierać się tylko na użytecznej informacji.
- Wymóg bardzo dużej szybkości działania. Przeważnie przytaczany podstawowy przykład sklepu WalMart wraz z jego ogromnymi terabajtowymi zasobami danych, pozwala wyobrazić sobie dość powszechne zjawisko, iż nawet małe próbki danych mogą zawierać setki tysięcy rekordów. Istnieje zatem potrzeba uczenia dla stosunkowo dużych zbiorów danych, bowiem może to zwiększyć jakość predykcji z uwagi na większą reprezentację zróżnicowanych przykładów. Przy tym warunek dużej szybkości działania tyczy się zarówno fazy uczenia jak i testowania (co ciekawe niektóre algorytmy są pod tym względem silnie niesymetryczne).
- Wymóg bardzo wysokiej jakości działania. W niektórych zastosowaniach, i wobec ogromnej masy danych, dokładność 99.9% poprawnie rozpoznanych wszystkich obiektów wciąż jest niewystarczająca i powoduje niemożliwą do weryfikacji liczbę alarmów np. przy rozpoznawaniu twarzy potencjalnych terrorystów na lotnisku, albo weryfikacji autoryzacji karty kredytowej.
- Działanie na bardzo małych próbach danych z uwagi na trudności techniczne lub koszty. Przykładowo dane geologiczne związane z odwiertami w poszukiwaniu ropy naftowej, próby zbadanych pacjentów.
- Działanie w oparciu o bardzo nietypowe lub wręcz źle zdefiniowane zadania. Przykładem mogą być dane medyczne dotyczące poziomu ekspresji genów u pacjentów chorych na raka, gdzie liczba atrybutów (np. obserwowanych genów) przekracza wielokrotnie ilość pomiarów (pacjentów).

Implementacje wielu popularnych technik uczenia z nadzorem można znaleźć w Statistica [Statistica], SAS [SAS], R [R-project], WEKA [Weka 3] i innych podobnych systemach. W większości z nich pewnemu zaniedbaniu uległy właśnie systemy oparte na indukcji reguł decyzyjnych. Jedynie reguły asocjacyjne zrobiły swego rodzaju "karierę", ponieważ okazały się pomocne szczególnie w sprzedaży i algorytm budujący je jest dość chętnie reimplementowany. Kilkuletnie doświadczenie autora z systemami wspomagającymi analizę danych doprowadziło do decyzji wypracowania implementacji własnego algorytmu opartego na indukcji reguł decyzyjnych, który będzie działał bardzo wydajnie wobec ogromnej ilości danych. Własna implementacja zrealizowana od podstaw w języku programowania JAVA pozwala na uniezależnienie się od różnych ograniczeń wynikających z implementacji w jednym z popularnych środowisk analitycznych (np. cena pakietu, zbyt rozbudowa-

ne GUI zmniejszające elastyczność, powolność działania, niewystarczająca skalowalność środowiska).

Dodatkowym atutem systemów regułowych oraz drzew decyzyjnych jest bardzo łatwa implementacja wiedzy w postaci warunków logicznych zapisywalnych w postaci składni języka SQL. Znakomita większość obecnie stosowanych baz i hurtowni danych oparta jest na typowych relacyjnych bazach w interfejsie tego języka. Prosta implementacja skraca czas wdrożenia oraz niezwykle ułatwia wprowadzanie manualnych korekt do modeli w trakcie ich pracy u klienta.

Krótko podsumowując, pierwszym celem pracy było zaprojektowanie i implementacja algorytmu klasyfikacyjnego który:

- będzie generował czytelną wiedzę w postaci ograniczonej liczby stosunkowo prostych reguł logicznych,
- będzie wystarczająco skalowalny by przetwarzać dziesiątki tysięcy wierszy i setki atrybutów w krótkim czasie (pozwoli to np. na skrócenie procesu optymalizacji parametrów klasyfikatora),
- będzie pozwalał na uzyskanie jakości predykcji porównywalnej z popularnymi technikami realizującymi to zadanie,
- będzie odporny na braki oraz sprzeczności występujące w danych.

Drugim zadaniem postawionym w pracy jest selekcja istotnych cech, a ściślej mówiąc budowa obiektywnego rankingu i wskazania na tym rankingu punktu odcięcia istotnego zestawu. Do realizacji tego zadania zaproponowano schemat algorytmu wykorzystującego różne techniki klasyfikacji. Wykorzystanie klasyfikatorów jest w tym zadaniu raczej środkiem do oceny zdolności predykcyjnych poszczególnych cech i na tej podstawie określa się istotność i pozycję w rankingu każdego z atrybutów.

Mając do dyspozycji dane na temat kilkudziesięciu pacjentów z których każdy jest opisany przez kilka tysięcy cech oraz jedną zmienną decyzyjną opisującą np. typ występującej choroby, chcemy zbudować ranking cech. Z góry wiadomo, że większość atrybutów (opisujących geny) nie jest związana z problemem i jedynie jakaś niewielka grupa wymagająca ustalenia jest istotna. Biologowie po otrzymaniu pełnego rankingu istotnej części są w stanie zweryfikować kolejne cechy z listy w celu odnalezienia związków z występującym schorzeniem. Lista kilkudziesięciu, czy kilkuset genów jest w tym wypadku możliwa do weryfikacji, ponieważ genetycy wydzielają pewne grupy genów odpowiedzialnych za podobne funkcje w organizmie. Dzięki temu można dokonać ogólnej kontroli wyniku algorytmu posługując się określonymi grupami.

Doświadczenie autora wskazuje że problem budowy rankingu cech z wysokowymiarowych danych w obliczu niewielkiej ilości obiektów nie ogranicza się jedynie do danych medycznych opisujących pewien stan genów pacjentów. Ten sam proces można z powodzeniem zastosować również do np. ustalenia istotnych parametrów związanych z budową studni i początkowym okresem wydobywania z niej ropy naftowej jak również innych problemów o dużej liczbie atrybutów.

3. Przegląd popularnych technik klasyfikacji

W tym rozdziale krótko zostaną przedstawione podstawowe techniki klasyfikacji popularnie stosowane w praktycznych rozwiązaniach. Część z technik miała niewątpliwie duży wpływ na konstrukcje algorytmów ADX i MCFS, a niektóre przedstawiono w celu prezentacji pełnego spectrum. Pewne korzystne elementy przedstawionych algorytmów były inspiracją dla autora, jednocześnie jednak dodatkowo starano się uniknąć niektórych ich wad, tak by sprostać postawionym sobie założeniom.

3.1. Klasyfikacja Bayesowska

Podstawą, na której opierają się techniki bayesowskie jest ogólnie znane twierdzenie Bayesa:

$$P(c|D) = \frac{P(c)P(D|c)}{P(D)} \quad (3.1)$$

Oznacza ono prawdopodobieństwo zajścia klasy c należącej do zbioru wszystkich klas C , pod warunkiem zaobserwowanych danych D . $P(c)$ opisuje prawdopodobieństwo *a priori* wystąpienia klasy c , $P(D)$ to prawdopodobieństwo zaobserwowania danych D , natomiast $P(D|c)$ to prawdopodobieństwo wystąpienia klasy c pod warunkiem wystąpienia danych D . W bardzo ogólnym skrócie $P(D|c)$ oraz $P(c)$ są przedmiotem uczenia co oznaczają, że na podstawie danych trenujących określa się odpowiednie prawdopodobieństwa (faktycznie bazując na częstościach): wystąpienia klasy $P(c)$, wystąpienia odpowiednich wartości atrybutów pod warunkiem zajścia klasy $P(D|c)$. Ponadto w praktycznych zastosowaniach istotne jest porównywanie prawdopodobieństw $P(c|D)$, i dlatego czynnik $P(D)$ niezależny od klasy jest pomijany.

Podstawowym klasyfikatorem zbudowanym w oparciu o powyższe twierdzenie jest **Naiwny Klasyfikator Bayesowski** (NBC), który zakłada niezależność zmiennych losowych składających się na obserwowany wektor losowy D z przestrzeni obserwacji. Określenie wartości decyzji d (klasy) nieznanego przykładu x' opisanego poprzez zestaw atrybutów a_i realizowane jest przy pomocy następującej formuły:

$$d(x') = \operatorname{argmax}_{c \in C} P(c) \prod_{i=1}^n P(a_i(x')|c) \quad (3.2)$$

Gdzie x ($x \in D$) oznacza zbiór przykładów trenujących, $P(c)$ oznacza prawdopodobieństwo wystąpienia klasy c , a $P(a_i(x')|c) = P(a_i(x)|c)$ warunkowe prawdopodobieństwo zajścia obserwowanej wartości $a_i(x')$ w klasie c .

Oba prawdopodobieństwa oszacowane na podstawie odpowiednich częstości wystąpień danych trenujących x . Założenie niezależności zmiennych mimo swojej prostoty dość dobrze sprawdza się w praktyce, a w szczególności w niektórych zastosowaniach np. takich jak klasyfikacja dokumentów gdzie decyzje podejmuje się na podstawie oszacowanych prawdopodobieństw warunkowych wystąpień poszczególnych słów (termów).

Naiwny Klasyfikator Bayesowski jest przypadkiem szczególnym **sieci Bayesowskiej**, która przyjmuje strukturę acyklicznego grafu skierowanego. W przypadku sieci Bayesowskiej jej węzły odpowiadają poszczególnym wartościom atrybutów. Połączeniom pomiędzy węzłami przypisuje się prawdopodobieństwo wystąpienia wartości w węzłach pod warunkiem obserwacji wartości w innych węzłach połączonych z nimi odpowiednimi krawędziami. Dla klasy decyzyjnej przy ustalonej strukturze sieci wartości prawdopodobieństw przypisane poszczególnym krawędziom są liczone oddzielnie. NBC w reprezentacji sieci można przedstawić jako graf, w którym węzły odpowiadające wszystkim wartościom obserwowanych atrybutów są połączone krawędziami jedynie z korzeniem oznaczającym klasę. Budowa grafu pełnego oznacza oszacowanie wszystkich możliwych prawdopodobieństw warunkowych osobno dla każdej z klas, co w praktyce jest właściwie niewykonalne. Dla większych zbiorów danych stosuje się tutaj z powodzeniem różne techniki upraszczające strukturę sieci a tym samym liczbę wymaganych szacowań prawdopodobieństw np. algorytm Chow-Liu [Chow *et al.*, 1968] czy ETC [Kłopotek, 2002]. Algorytmy te przy wykorzystaniu pewnych heurystyk pozwalają na budowę niepełnego grafu, który odzwierciedla najbardziej istotne zależności pomiędzy poszczególnymi wartościami atrybutów.

3.2. Dyskryminacja liniowa i nieliniowa

Jedną z pierwszych propozycji rozwiązania problemu klasyfikacji był algorytm liniowej dyskryminacji (**LDA**) zaproponowany przez Fishera [Fisher, 1936]. Oryginalnie metoda została zaproponowana dla dwóch klas w ujęciu rzutowania przykładów na prostą (często oryginalne rozwiązanie Fishera jest nazywane FDA, w przeciwieństwie do LDA, gdzie mówimy o uogólnionej wersji algorytmu).

Wynikiem działania LDA jest funkcja liniowa (hiperpłaszczyzna), która w możliwie najlepszy sposób jest w stanie rozdzielić przykłady należące do dwóch klas w euklidesowej przestrzeni reprezentacji o p wymiarach. Funkcję tę znajduje się przy pomocy rzutowania przykładów na prostą, na której uzyskujemy maksymalne skupienie wewnątrz obu klas oraz maksymalną separację pomiędzy oboma klasami. Takie założenia spełnimy poprzez minimalizację następującej funkcji:

$$J(w) = \frac{w^T S_B w}{w^T S_W w} \quad (3.3)$$

gdzie S_W jest macierzą rozrzutu dla wektorów z tej samej klasy, a S_B macierzą rozrzutu dla wektorów różnych klas. Należy zauważyć że macierze

rozrzutu są proporcjonalne do macierzy kowariancji dlatego można zdefiniować J poprzez macierze kowariancji.

$$S_B = \sum_c N_c (\mu_c - \bar{x})(\mu_c - \bar{x})^T \quad (3.4)$$

$$S_W = \sum_c \sum_{i \in c} (x_i - \mu_c)(x_i - \mu_c)^T; \quad (3.5)$$

we wzorach 3.4 i 3.5 μ_c i $\bar{x}$ oznaczają odpowiednio:

$$\mu_c = \frac{1}{N_c} \sum_{i \in c} x_i \quad (3.6)$$

$$\bar{x} = \frac{1}{N} \sum_i x_i = \frac{1}{N} \sum_c N_c \mu_c, \quad (3.7)$$

gdzie N_c oznacza liczbę przykładów w klasie c . Dla problemu o dwóch klasach komponent S_B można zdefiniować jako:

$$S'_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T, \quad (3.8)$$

natomiast rozwiązanie 3.3 przyjmuje postać:

$$w = S_W^{-1}(\mu_1 - \mu_2) \quad (3.9)$$

W późniejszym okresie zaproponowano wiele różnych technik bazujących na oryginalnym algorytmie Fishera: dla problemu wielu klas ([Bryan, 1951]), kwadratowa dyskryminacja (**QDA**), **regresja liniowa**, **regresja logistyczna**.

Najbardziej zaawansowanym rozwinięciem idei LDA są maszyny wektorów podpierających (**SVM** - *Support Vector Machines* [Vapnik, 1995]), które opisując skrótowo starają się znaleźć taką hiperpłaszczyznę rozdzielającą klasy zachowując możliwie maksymalny margines odległości od wektorów na pograniczu klas. W SVM w przeciwieństwie do LDA dyskryminacja liniowa jest realizowana w odmiennej przestrzeni niż oryginalna, przez co granica podziału w pierwotnej przestrzeni reprezentacji może mieć silnie nieliniowy charakter.

3.3. Metody odległościowe

Jedną z najprostszych i zarazem wystarczająco skutecznych technik klasyfikacji jest k -NN (*k nearest neighbor*) zaproponowana w pracy [Fix et al., 1951], sformalizowana w [Cover et al., 1967]. Klasyfikacja w przypadku k -NN polega na wybraniu k przykładów z próby uczącej najbliższych w stosunku do przykładu rozważanego i nadaniu mu etykiety klasy wynikającej z głosowania wśród k przykładów. W głosowaniu najczęściej wykorzystuje się odpowiednie częstości wystąpień k przykładów z różnych klas, choć w bardziej zaawansowanym wydaniu można dodatkowo ważyć odpowiednie przynależności do klas odległościami od rozważanego przykładu. Proces uczenia polega jedynie na zapamiętaniu wszystkich przykładów trenujących by w trakcie

klasyfikacji można było policzyć odpowiednie odległości. Przyjmowana miara odległości jest przeważnie euklidesowa, natomiast dobór k jest zadaniem czysto eksperymentalnym. Im większa wartość tego parametru tym więcej sąsiadów bierze udział w głosowaniu przyznawanej etykiety klasy, co wpływa na zmniejszenie szumu odpowiedzi, lecz zarazem na jej rozmycie. Technika ta jest chętnie stosowana w praktyce, a liczba pojawiających się prac wskazuje na niesłabnące zainteresowanie jej rozwojem [Jóźwik *et al.*, 1998], [Shakhnarovich *et al.*, 2005].

3.4. Drzewa decyzyjne

Pierwotnie drzewa decyzyjne pojawiły się w obszarze badań dotyczących socjologii i psychologii we wczesnych latach 60-tych. Na stałe zagościły jednak w statystyce trochę później bo po opublikowaniu prac Quinlana [Quinlan, 1986] i Breimana [Breiman *et al.*, 1984].

Drzewo decyzyjne jest acyklicznym, spójnym grafem skierowanym o jednym wierzchołku zwanym korzeniem. Każdy kolejny węzeł w drzewie jest odpowiedzialny za pewien test wartości atrybutu. Wynik testu przesądza o wyborze krawędzi kierującej do kolejnego węzła. Na spodzie drzewa decyzyjnego znajdują się specyficzne węzły zwane liśćmi, które determinują etykietę klasy. By zatem zaklasyfikować nieznany przykład przeprowadza się kolejne testy wartości wędrując wzdłuż krawędzi po jednej ścieżce grafu począwszy od jego korzenia kończąc na odpowiednim liściu.

Jedną z najwcześniejszych propozycji indukcji drzewa decyzyjnego jest zaproponowany przez Quinlana algorytm **ID3** ([Quinlan, 1986], [Mitchell, 1997]), który ma w tej chwili tylko znaczenie historyczne. Budowa drzewa jest realizowana począwszy od korzenia, a jako atrybut testowy wybiera się taki, który maksymalizuje przyrost informacji (ang. *information gain*) w węźle. Dla zbioru przykładów D przyrost informacji dla atrybutu a będzie wyznaczony jako:

$$IG_a(D) = I(D) - E_a(D) \quad (3.10)$$

gdzie $I(D)$ oznacza informację zawartą w zbiorze danych D :

$$I(D) = \sum_{c \in C} -\frac{|D^c|}{|D|} \log \frac{|D^c|}{|D|} \quad (3.11)$$

natomiast $E_a(D)$ dla atrybutu a to średnia ważona odpowiednich entropii określonych dla przykładów gdzie a przyjmuje wartość v ($v \in R_a$).

$$E_a(D) = \sum_{v \in R_a} \frac{|D_{a=v}|}{|D|} \sum_{c \in C} -\frac{|D_{a=v}^c|}{|D_{a=v}|} \log \frac{|D_{a=v}^c|}{|D_{a=v}|} \quad (3.12)$$

Istotną wadą algorytmu ID3 jest działanie tylko dla atrybutów nominalnych; dodatkowo finalne drzewa są zbyt dopasowane do danych oraz miara przyrostu informacji faworyzuje cechy o dużej liczbie wartości.

Wszystkie powyżej przytoczone problemy zostały wyeliminowane przez autora w kolejnym algorytmie nazwanym **C4.5**, który jest jednym z chętniej stosowanych w praktyce. Quinlan w nowym algorytmie ([Quinlan, 1993], [Quinlan, 1996]) miarę przyrostu informacji wzbogacił o czynnik normalizujący, a nowa miara została nazwana współczynnikiem przyrostu informacji (ang. *gain ratio*). Czynnik normalizujący znajdujący się w mianowniku wzoru 3.13 oznacza wartość informacyjną atrybutu a .

$$GR_a(D) = \frac{IG_a(D)}{\sum_{v \in R_a} - \frac{|D_{a=v}|}{|D|} \log \frac{|D_{a=v}|}{|D|}} \quad (3.13)$$

Rozwiązanie dla atrybutów ciągłych polega na testowaniu wszystkich możliwych wartości podziału v_p . Oznacza to policzenie odpowiedniej wartości $GR_a(D)$ dla binarnego atrybutu a o wartościach $v = \{v_1, v_2\}$ powstałych przez podział w v_p (takich że $v_1 \leq v_p$ oraz $v_2 > v_p$). W trakcie budowy drzewa zakłada się, że jeden atrybut numeryczny może występować wielokrotnie na różnych poziomach drzewa. Z punktu widzenia logiki pozwala to na uzyskanie przedziałów obustronnie domkniętych (na mocy koniunkcji).

Kolejnym rozszerzeniem w stosunku do ID3 jest obsługa wartości nieznanymi. Rozwiązanie na jakie zdecydował się autor jest bardzo intuicyjne i opiera się na ignorowaniu przykładów zawierających wartość nieznaną dla atrybutu, który jest rozważany. Oznacza to, iż w przypadku występowania wartości nieznanej dla przykładu $x \in D$ na atrybucie a_1 , współczynnik przyrostu informacji liczy się jako $GR_{a_1}(D_1)$ gdzie $D_1 = D \setminus x$.

W C4.5 wprowadzono bardzo istotną modyfikację polegającą na przycinaniu finalnie zbudowanego drzewa. W skrócie polega to na zastępowaniu fragmentu drzewa liściem wtedy gdy jest spełnione pewne kryterium. Przycinanie kosztem rozmycia decyzji i zwiększenia błędu klasyfikacji na zbiorze trenującym pozwala na zmniejszenie tego błędu dla nieznanymi danych, a tym samym jest niezwykle istotne z punktu widzenia zastosowań praktycznych. Przycięte drzewo jest mniejsze co dodatkowo korzystnie wpływa na czytelność opisu oraz efektywność działania. Kryterium przycinania w C4.5 bazuje na ocenie statystycznej istotności prawdopodobieństwa zmniejszenia błędu klasyfikacji liścia, który miałby zastąpić istniejący fragment drzewa. Przyjmując, że liczba błędów klasyfikacji ma rozkład dwumianowy, węzeł zastępuje się odpowiednim liściem jeśli błąd liścia e_l jest mniejszy niż pewna ustalona wartość progu. W rozwiązaniach praktycznych kryterium przycięcia wyznacza wzór 3.14, gdzie e_n oznacza błąd klasyfikacji dla zastępowanego węzła, a $|D_n|$ liczbę przykładów znajdujących się w węźle n .

$$e_l \leq e_n + \sqrt{\frac{e_n(1 - e_n)}{|D_n|}} \quad (3.14)$$

Warto jeszcze wspomnieć, że autor algorytmu udostępnił publicznie kod implementacji klasyfikatora C4.5 a kolejną wersję C5.0/See5 (odpowiednio wersja dla Linux/Windows) rozwija już pod szyldem firmy komercyjnej. W nowym rozwiązaniu poprawiono kilka aspektów: zwiększono szybkość,

zmniejszono zużycie pamięci, zmniejszono wielkość budowanych drzew, dodano ważenie atrybutów i błędów klasyfikacji, dodano boosting, dodano mechanizm redukcji szumu. Jak widać z przytoczonej listy, wszystkie zmiany mają na celu zwiększenie atrakcyjności algorytmu w zastosowaniach praktycznych, w których nawet wersja C4.5 radzi sobie całkiem dobrze.

Drugim niezwykle popularnym algorytmem indukcji drzewa decyzyjnego jest **CART** (*Classification And Regression Tree*) zaproponowany przez Breimana ([Breiman *et al.*, 1984]). w przeciwieństwie do omówionego już C4.5 wywodzącego się z teorii informacji CART został silniej oparty na analizie statystycznej. Miarą wyznaczającą najlepszy podział jest indeks Giniego (*Gini Index*):

$$GI_a(D) = \sum_{c \in C} \frac{|D^c|}{|D|} \left(1 - \frac{|D^c|}{|D|}\right) \quad (3.15)$$

Przy czym w przeciwieństwie do C4.5 CART buduje tylko drzewa binarne co oznacza, że dla atrybutów nominalnych rozważa się wszystkie możliwości podziału na dwie rozłączne grupy wartości. W przypadku atrybutów numerycznych w momencie budowania węzła podobnie jak w C4.5 należy sprawdzić wszystkie możliwe podziały na dwa rozłączne przedziały. Binaryzacja dotyczy się również obserwowanych klas co oznacza, że w przypadku więcej niż dwóch na potrzeby podziału w węźle rozważa się chwilowe scalenie przykładów z różnych kategorii. Podziały binarne w węźle pozwalają na ogólne osiągnięcie większej głębokości drzewa, co pozwala na odkrycie dłuższych bardziej specyficznych gałęzi/reguł. Dodatkowo binarne testy pozwalają na ogólne przyspieszenie działania drzewa.

W przypadku występowania brakujących wartości w danych w drzewie wykorzystuje się podziały zastępcze (*surrogate splitters*), które są swego rodzaju zabezpieczeniem pozwalającym uzyskać możliwie podobny wynik podziału przykładów co wybrany warunek. Węzły w drzewie zawierają listę takich alternatywnych podziałów opartych na różnych atrybutach. Podziały z listy wykorzystuje się w przypadku braku wartości, na podstawie której można by było wnioskować przy użyciu podstawowej reguły węzła.

Do przycinania drzewa stosuje się kryterium minimalnego kosztu złożoności (*minimal cost complexity pruning*), które zakłada że obciążenie błędu fragmentu drzewa rośnie liniowo wraz z liczbą liści. Koszt przypisany do drzewa t jest sumą: kosztu błędu klasyfikacji $E(t)$ oraz liczbą liści $num.leaf(t)$ mnożoną przez parametr opisujący złożoność α . Najlepiej przycięte drzewo będzie minimalizować wyrażenie 3.16

$$E_\alpha(t) = E(t) + \alpha \cdot num.leaf(t) \quad (3.16)$$

W ramach algorytmów realizujących budowę drzew decyzyjnych należy wymienić CHAID (*CHi-squared Automatic Interaction Detection*), które używa jako miary czystości węzła statystyki χ^2 . Warto wspomnieć o innych interesujących algorytmach budowania drzew jak: FACT (*Fast Algorithm for Classification Trees*) [Loh *et al.*, 1988], QUEST (*Quick, Unbiased, Effi-*

cient, *Statistical Tree*) [Loh *et al.*, 1997], SLIQ [Mehta *et al.*, 1996], SPRINT [Shafer *et al.*, 1996] które to implementują pewne modyfikacje oraz pozwalają na przyspieszenie procesu uczenia w stosunku do omówionych rozwiązań.

Drzewa decyzyjne poprzez dużą uniwersalność są w praktyce jednymi z chętniej stosowanych klasyfikatorów. Ich niewątpliwymi zaletami jest: duża szybkość procesu uczenia i klasyfikacji, generowanie czytelnej wiedzy, wysoka skuteczność działania, intuicyjne podejście do dekomponowania problemu typu 'dziel i rządź', popularność różnych implementacji.

Niestety dla stosunkowo dużych i skomplikowanych zbiorów danych drzewa z racji wielkości przestają być czytelne dla człowieka. W takim wypadku w dość prosty sposób daje się je przedstawiać jako zbiór rozłącznych reguł decyzyjnych. W przypadku np. C4.5 integralną częścią algorytmu jest tłumaczenie drzewa na zbiór reguł i przeprowadzenie operacji przycinania już na bazie reguł.

Drugim problemem występującym w przypadku bardziej skomplikowanych zbiorów danych jest niestabilność. Zdarza się, że niewielka zmiana zbioru trenującego prowadzi do uzyskania zupełnie innej postaci grafu, która jednak nadal będzie dobrze wykonywać zadanie klasyfikacji choć będzie odmienną postacią. Własności opisowe drzewa są na skutek tego zjawiska nieco ograniczone choć skala tego problemu jest głównie zależna od charakteru wykorzystywanych danych.

3.5. Klasyfikatory regułowe

Jak wspomniano drzewa decyzyjne właściwie budują wiedzę w postaci grafu kolejnych warunków który można przedstawić w formie listy reguł decyzyjnych. Osobną grupę systemów uczących stanowią jednak takie, które bezpośrednio z danych budują bazę wiedzy w postaci zestawu reguł. Algorytmy tej rodziny przeważnie mają silne oparcie w dziale logiki predykatów i wymagają wartości dyskretnych atrybutów. Dyskretyzacja nie jest jednak dużym problemem, ponieważ wszystkie rozwiązania wykorzystywane w praktyce bądź wykorzystują algorytm dyskretyzacji wstępnej, bądź bazują na odpowiednich cięciach (przedziałach wartości) podobnie jak to ma miejsce w przypadku drzew decyzyjnych.

W przypadku systemów regułowych trzeba omówić kilka prostych pojęć, którymi operuje się w definicjach większości tej rodziny algorytmów. Dokładniejsze, bardziej formalne definicje pojęć zamieszczono dalej przy okazji pełnego opisu algorytmu ADX, natomiast w tej chwili wystarczy bardzo ogólnie zasygnalizować, co rozumie się poprzez poszczególne pojęcia. Selektor to warunek logiczny oparty na atrybucie. Kompleks to koniunkcja selektorów. Pokrycie to iloraz liczby przykładów spełniających warunki selektora lub kompleksu i liczby wszystkich przykładów.

W przeciwieństwie do reguły asocjacyjnej przedstawionej we wprowadzeniu, reguła decyzyjna determinuje klasę obiektu, a jej formalną postać przedstawia zapis:

$$\text{IF } \textit{warunki} \text{ THEN } \textit{klasa} \quad (3.17)$$

Co oznacza, że jeśli spełnione są warunki części warunkowej podanej formuły przez przykłady, to część decyzyjna ściśle określa klasę tych przykładów.

Przegląd najpopularniejszych algorytmów indukcji reguł zostanie jednak rozpoczęta od algorytmu **Apriori** [Agrawal *et al.*, 1994] indukcji reguł asocjacyjnych. Problem, jaki jest rozwiązywany przez algorytm *Apriori*, nie ma na celu klasyfikacji lecz należy tutaj o nim wspomnieć ponieważ jest on ważną inspiracją dla algorytmu ADX.

Podstawowymi miarami jakości reguły asocjacyjnej są *wsparcie* i *zaufanie* zdefiniowane wcześniej (równania 2.1 i 2.2). Parametrami wejściowymi algorytmu jest minimalne wsparcie i zaufanie reguł które chcemy znaleźć. Reguły te mają opierać się na pojedynczych koszykach klientów, w których jednocześnie zakupiono produkty $\langle X, Y, Z \rangle$ a zakup np. pozycji X, Y wpłynął na kupno pozycji Z ($X, Y \implies Z$). W pierwszym kroku algorytmu wybiera się wszystkie pojedyncze pozycje spełniające warunek wsparcia a następnie poprzez połączenie ich w pary buduje reguły typu $X \implies Y$ określane jako zestawy *kandydaci*. Każda iteracja algorytmu polega na wygenerowaniu kolejnych dłuższych kandydatów, obliczeniu stosownych wartości miar wsparcia i zaufania, i następnie odrzuceniu tych które nie spełniają zadanych warunków. W momencie generowania dłuższych kandydatów łączy się jedynie takie pary które mają $n - 1$ pozycji poprzednika wspólnych, gdzie n oznacza liczbę pozycji poprzednika reguły, a następnik obu jest identyczny. Proces kończy się w momencie gdy nie da się zbudować kolejnych kandydatów lub wygenerowane oceniane zestawy nie spełniają warunków minimalnych wartości wsparcia i zaufania.

Algorytm **PRISM** [Cendrowska, 1987] buduje reguły wykorzystując strategię podobną jak przy budowaniu jednej gałęzi drzewa decyzyjnego. W celu znalezienia reguły opisującej przykłady rozpatrywanej klasy (przykłady pozytywne) w każdym kolejnym kroku dodaje się jeden warunek który maksymalizuje na danym etapie miarę ilorazu pokrywanych przykładów rozpatrywanej klasy do wszystkich przykładów. Dodanie kolejnego warunku do reguły jest analogiczną operacją jak dodanie kolejnego węzła dla gałęzi drzewa. Algorytm ten jest dość prosty, działa tylko w oparciu o nominalne atrybuty, cechuje się nadmiernym dopasowaniem i z tych powodów nie sprawdza się w praktyce, ale może być traktowany jako dobry punkt wyjścia dla bardziej zaawansowanych rozwiązań. Jedną z modyfikacji, która właśnie przeciwdziała nadmiernemu dopasowaniu reguł, jest algorytm **INDUCT** [Gaines, 1989]. Wykorzystuje on badanie statystyczne do oceny ewentualnego odrzucenia ostatnio dodanego warunku. Inny algorytm **RIDOR** [Weka 3] dodatkowo pozwala na generowanie wyjątków od reguł wykorzystując przy tym strategię pokrywania jak w PRISM. Dodano w nim również obsługę atrybutów numerycznych.

Jednym z bardziej znanych algorytmów indukcji reguł jest algorytm **RIPPER** przedstawiony w [Cohen, 1995]. Pojedyncze warunki w tym algorytmie mogą opierać się na relacji równości (dla atrybutów nominalnych) lub

większości, mniejszości (dla atrybutów numerycznych). W pierwszym kroku wszystkie możliwe warunki (selektory) są oceniane według kryterium podziału (opartym na entropii podobnie jak w C4.5) i następnie sortowane. Budowanie reguły polega na kolejnym dodawaniu selektorów z listy do momentu braku pokrywania przykładów negatywnych (wszystkich innych niż z rozpatrywanej klasy) przez regułę. Przycinanie reguł dzieje się w dokładnie odwrotnym porządku jak wzrost, tzn. odrzuca się warunki dodane poprzednio w celu maksymalizacji formuły:

$$Q_{ripper} = \frac{p - n}{p + n} \quad (3.18)$$

gdzie p oznacza liczbę przykładów pozytywnych a n liczbę przykładów negatywnych pokrywanych przez regułę. Do przycinania, podobnie jak w C4.5, stosuje się wydzieloną grupę przykładów (1/3) ze zbioru trenującego nie używanych do budowy reguł. Po zbudowaniu jednej reguły usuwa się ze zbioru przykładów trenujących te, które reguła pokrywa. W ten sposób wymuszamy dla kolejnych reguł pokrywanie innego obszaru, a proces często nazywamy sekwencyjnym pokrywaniem. Oznacza to, że algorytm budując każdą kolejną regułę opisuje nią kolejny niepokryty dotąd obszar, przeważnie zwiększając specyficzność kolejnej reguły.

Drugim mechanizmem przeciwdziałającym nadmiernemu dopasowaniu jest ograniczanie długości opisu (*description length*), która wyraża liczbę bitów potrzebną do zakodowania bieżącego zestawu reguł (reguł zbudowanych dla rozpatrywanej klasy). W kolejności odwrotnej do porządku dodawania kolejnych reguł usuwa się reguły, jeśli wpływa to na zmniejszenie długości opisu całego zestawu. Strategia przycinania zestawu jest następująca:

- znajdź przykłady pozytywne p i negatywne n pokrywane przez regułę R ,
- inicjuj R_1 jako regułę pustą, a $R_2 := R$, równą regule R ,
- buduj i przycinaj R_1 , R_2 w oparciu o p i n ,
- finalnie podstaw $R := \{R_1, R_2\}$ dla której długość opisu jest mniejsza.

Innym bardzo ważnym algorytmem bazującym na strategii sekwencyjnego pokrywania jest **AQ** [Kaufman *et al.*,1999], [Michalski *et al.*,2001]. Autor algorytmu zaproponował wiele różnych modyfikacji, a sama implementacja występuje w kilku różnych wersjach, podstawowy schemat pozostaje jednak bez zmian i poniższy opis zostanie do niego ograniczony.

Zbiór przykładów oznaczmy jako D , natomiast zbiór przykładów dotychczas niepokrytych jako T . Strategia algorytmu opiera się na skupieniu na jednym z niepokrytych dotąd przykładów z T i budowaniu w oparciu o niego możliwie ogólnego kompleksu niepokrywającego żadnego z przykładów z klasy przeciwnej z D . Ze zbioru przykładów T algorytm wybiera przykład zwany ziarnem pozytywnym. Następnie inicjuje się podręczny zbiór kompleksów (mających pokrywać tylko przykłady z klasy ziarna pozytywnego) zwany gwiazdą S (na tym etapie jest to jeden uniwersalny kompleks - bezwarunkowy pokrywający wszystkie przykłady). Podobnie tak jak wybrano ziarno pozytywne wybiera się również ziarno negatywne reprezentujące przykład z przeciwnej klasy. Wybór ziaren może przyjmować kilka różnych strategii,

np. mogą to być kolejne przykłady z listy, losowo wybrane lub spełniające odpowiednie warunki maksymalnego zróżnicowania w stosunku do uprzednio wybranych ziaren. W trakcie budowania kolejnych reguł ziarnami pozytywnym mogą być przykłady należące do różnych klas. Finalny zbiór reguł jest nieuporządkowany i składa się z reguł determinujących odpowiednie klasy.

Po wybraniu obu ziaren buduje się częściową gwiazdę S' , która skupia wszystkie możliwe maksymalnie ogólne kompleksy pokrywające ziarno pozytywne i nie pokrywające ziarno negatywne. Częściowa gwiazda w kolejnym kroku zostaje poddana operacji przecięcia z uprzednio zainicjowaną gwiazdą S . Operacja przecięcia obu zbiorów S i S' polega na koniunkcji wszystkich par kompleksów odpowiednio z S i S' . Liczba przechowywanych kompleksów w S wyznacza parametr b i jeśli po operacji przecięcia w zbiorze jest więcej niż b kompleksów to pozostawia się tylko te o najwyższym poziomie ogólności. Poziom ogólności/szczegółowości bada się na podstawie zbioru przykładów pokrywanych przez porównywane kompleksy. Jeśli zbiór przykładów pokrywanych przez jeden kompleks jest podzbiorem przykładów pokrywanych przez drugi z nich, to drugi w stosunku do pierwszego jest bardziej ogólny. Jeśli te zbiory przykładów są tożsame, to można ogólność wyrażać poprzez np. liczbę uniwersalnych selektorów zawartych w każdym z kompleksów lub liczbę pokrywanych przykładów pozytywnych. Strategia oceny poziomu ogólności kompleksu może być sterowana parametrem, dzięki czemu można finalnie otrzymać wynik o nieco odmiennym charakterze.

Wybrany zbiór b kompleksów stanowi S i jeśli wciąż pokrywa on jakiegokolwiek przykłady negatywne z D , to wybiera się kolejne ziarno negatywne z T . Proces budowania częściowej gwiazdy i przecinania jej z S powtarza się iteracyjnie do momentu, gdy kompleksy w S nie będą pokrywały żadnego przykładu negatywnego z D . Jeśli udało się już zbudować taki zbiór b kompleksów w S to najlepszy (np. najbardziej ogólny) z nich trafia do finalnego zestawu reguł, a przykłady pokrywane przez niego są usuwane z dotychczasowego zbioru T i realizuje się kolejną sekwencję pokrywania wybierając nowe ziarno.

Podsumowując, algorytm AQ przy wykorzystaniu pewnej heurystyki generuje kolejno maksymalnie ogólne kompleksy pokrywające możliwie dużo przykładów z jednej klasy. Z każdą kolejną sekwencją generowane reguły mają za zadanie pokryć nieco odmienny obszar (przynajmniej jeden niepokryty dotąd przykład).

Istotną cechą AQ jest budowanie silnych (pokrywających jedynie przykłady pozytywne) reguł, a końcowy ich zestaw jest kompletny i spójny. Reklasyfikacja zbioru trenującego pozwala uzyskać praktycznie zerowy błąd. Niestety taka strategia silnie wpływa na nadmierne dopasowanie do zbioru uczącego. W procesie klasyfikacji wymaga to używania pewnej miary mierzącej odległość przykładu od reguły, ponieważ silnie dopasowane reguły pokrywają stosunkowo mało nieznanych przykładów.

Algorytm **CN2** zaproponowany w [Clarc *et al.*, 1997] miał na celu modyfikację podstawowej wersji AQ przez wykorzystanie pewnych technik znanych z drzew decyzyjnych. Najistotniejszymi różnicami są przeciwdziałanie nadmiernemu dopasowaniu i budowanie zbioru uporządkowanych reguł. Ope-

racja budowy pojedynczej reguły polega na iteracyjnym przecinaniu zbioru kompleksów S (tutaj również zwanych gwiazdą) ze zbiorem kompleksów atomowych $\mathbb{S}$. Kolejne operacje przecięcia powodują powstawanie coraz bardziej szczegółowych kompleksów, a proces budowy jednej reguły kończy się gdy zbiór S zostaje pusty lub udało się wygenerować kompleks o maksymalnej jakości. Do oceny jakości kompleksów stosuje się tutaj negację entropii oraz dodatkowo test istotności statystycznej. Wtedy funkcja zanegowanej entropii przyjmuje wartość maksymalną zero w przypadku idealnej separacji klas przez kompleks. Stosowanie statystyki istotności (np. χ^2) można interpretować jako pomiar odległości między oczekiwanym rozkładem częstości klas a rozkładem przykładów między klasy występującym w zbiorze przykładów pokrywanych przez kompleks. Jeśli ta odległość będzie odpowiednio duża to rozważany kompleks zostanie uznany za statystycznie istotny. Wprowadzenie miary oceny kompleksu i testu istotności w CN2 powoduje, że w przeciwieństwie do AQ, reguły nie muszą pokrywać przykładów należących jedynie do tej samej klasy.

Po każdym przecięciu S z $\mathbb{S}$ usuwa się wszystkie kompleksy sprzeczne i wybiera b najwyżej ocenionych, które z kolei stanowią nową gwiazdę S . W całym procesie specjalizacji jednej gwiazdy przechowuje się najlepszy kompleks wygenerowany dotychczas, r^* . Po operacji przecięcia testuje się, czy którykolwiek z kompleksów w nowej gwiazdzie S jest wyżej oceniany niż r^* , jeśli tak to lepszy stanowi nowe r^* . Jeśli S po usunięciu sprzecznych kompleksów jest puste lub r^* stanowi kompleks o maksymalnej wartości oceny, to r^* trafia do zbioru finalnych reguł, a przykłady pokrywane przez r^* są usuwane ze zbioru D . Następnym krokiem takiej operacji jest tworzenie hierarchicznego zbioru reguł oraz możliwość wygenerowania reguły która pokrywa przykłady negatywne wcześniej odrzucone.

Osobną rodzinę algorytmów budujących wiedzę w postaci reguł są algorytmy oparte na teorii **zbiorów przybliżonych** (*rough sets*). Podstawy tej teorii sformułował wybitny polski uczony prof. Zdzisław Pawlak w pracach [Pawlak, 1982], [Pawlak, 1991] i jest to w tej chwili bardzo popularny kierunek badań. Teoria, na bazie której zaproponowano wiele algorytmów indukcji reguł, głównie skupia się na obsłudze sprzeczności w danych, które w praktyce (szczególnie w medycynie) pojawiają się bardzo często. W teorii zbiorów przybliżonych U oznacza uniwersum czyli zbiór wszystkich przykładów (w pracy oznaczane jako D), A oznacza zbiór atrybutów ($a \in A$), a V zbiór wartości ($v \in V$). Dwa obiekty $x, y \in D$ będą ze sobą związane relacją nierozróżnialności (IND), gdy wszystkie cechy obu obiektów przyjmują takie same wartości (3.19)

$$IND(A) = \{(x, y) \in D | \forall_{a \in A}, a(x) = a(y)\} \quad (3.19)$$

Klasy abstrakcji $[x]_{IND}$ występujące w relacji nierozróżnialności IND są nazywane klasami nierozróżnialności lub zbiorami elementarnymi. Oznaczają one rozłączne podzbiory przykładów w D takich, które są nierozróżnialne w oparciu o zbiór atrybutów A . Jeśli dla przykładu $x \in D$ nie można wyróżnić innego będącego z nim w relacji nierozróżnialności (posiadającego identyczne

wartości atrybutów) to x tworzy jednoelementowy zbiór pewnej klasy abstrakcji.

Niech $X \subseteq D$ będzie zbiorem przykładów, który chcemy opisać używając podzbioru atrybutów B ($B \subseteq A$). Każdy zbiór X można opisać przy pomocy jego przybliżenia dolnego 3.20 i górnego 3.21.

$$\underline{B}X = \{x \in D \mid [x]_{IND} \subseteq X\} \quad (3.20)$$

$$\overline{B}X = \{x \in D \mid [x]_{IND} \cap X \neq \emptyset\} \quad (3.21)$$

Jeżeli dla zbioru X , $\underline{B}X = \overline{B}X$ to zbiór nazywamy dokładnym, w przeciwnym przypadku jest on nazywany przybliżonym. Różnicę zbiorów $\overline{B}X - \underline{B}X$ nazywa się brzegiem zbioru X i dla zbioru przybliżonego zawiera ona zbiór obiektów, których część klasy abstrakcji relacji nierozróżnialności należy do zbioru X a część nie.

Reduktem nazywamy minimalny podzbiór atrybutów potrzebny do opisu i rozróżnienia zbioru obiektów.

Na bazie teorii zbiorów przybliżonych powstało kilka systemów analitycznych przykładowo LERS [Grzymała-Busse, 1992], [Grzymała-Busse, 1999], Rosetta [Ohrn, 1997] czy RSES [RSES].

Najpopularniejszym algorytmem indukcji reguł w oparciu o zbiory przybliżone jest algorytm **LEM2** zaproponowany we wspomnianym systemie LERS. Podstawowa wersja algorytmu wymaga przeprowadzenia dyskretyzacji wstępnej i nie umożliwia obsługi braków wartości lecz odpowiednie modyfikacje LEM2 radzą sobie z tymi problemami. Algorytmy **MODLEM** [Stefanowski, 2001] i **MLEM2** [Grzymała-Busse, 2003] proponują znajdowanie podziałów w trakcie budowy reguł, a modyfikacja przedstawiona w [Grzymała-Busse *et al.*, 1997] pozwala w LEM2 na obsługę pustych wartości.

Algorytm LEM2 tworzy minimalny zestaw reguł dla każdej z klas w oparciu o ogólny schemat sekwencyjnego pokrywania. Zbiór reguł budowanych w oparciu o przybliżenie dolne to zbiór reguł *pewny* natomiast w oparciu o przybliżenie górne to zbiór *możliwy*.

3.6. Metauczenie

W sekcji omówione zostaną techniki które dzięki odpowiedniemu próbkowaniu przeprowadzonym na zbiorze trenującym pozwalają na uzyskiwanie relatywnie lepszych rezultatów np. szacowania parametrów rozkładu czy klasyfikacji zbioru testującego.

Historycznie pierwszą propozycją odpowiedniego próbkowania zbioru danych była technika zwana **Bootstrap**, [Efron, 1979]. Polega ona na wielokrotnym losowaniu z powtórzeniami przykładów ze zbioru danych w celu zbudowania szeregu zbiorów danych o wielkości równej oryginalnemu. W oparciu o próbki *bootstrapowe* można zbudować rozkłady badanej wartości (np. błędu, przedziałów ufności, średniej, mediany, współczynnika korelacji).

Bagging (*Bootstrap aggregating*) został zaproponowany w [Breiman, 1996] i

opiera się na wykorzystaniu próbek *bootstrapowych* do budowy szeregu klasyfikatorów. Technika ta poprawia wyniki klasyfikacji i modeli regresyjnych pod względem stabilności i dokładności, dodatkowo również zmniejsza wariancję i przeciwdziała nadmiernemu dopasowaniu. Pierwotnie *bagging* został użyty w połączeniu z drzewami decyzyjnymi i regresyjnymi choć poprawa wyników jest właściwością procesu niezależną od typu zastosowanego klasyfikatora.

Boosting to technika opierająca się na sekwencyjnej budowie wielu słabych klasyfikatorów w oparciu o próbki danych zbioru trenującego, przygotowane tak by kolejne modele uczono głównie na przykładach do tej pory źle rozpoznanych. Z każdym przykładem zbioru trenującego związana jest pewna waga w podlegająca modyfikacji po każdym kroku budowania i testowania modelu. Jeśli przykład i został źle zaklasyfikowany to odpowiadająca mu wartość w jest odpowiednio zwiększana, co podnosi prawdopodobieństwo wylosowania i jako próbki uczącej w następnym kroku.

Najpopularniejszy algorytm bazujący wykorzystujący *boosting* to **Ada-Boost**, a został on przedstawiony w [Freund *et al.*, 1996], [Freund *et al.*, 1997]. W pierwszym kroku wszystkie wagi przykładów są inicjowane jednakową wartością $w_i = \frac{1}{|D|}$, gdzie w_i oznacza wagę przykładu o_i , a $|D|$ liczbę przykładów w zbiorze trenującym. Do zbudowania modelu wykorzystuje się próbę przykładów wylosowanych z rozkładem w_i ze zbioru trenującego. Błąd klasyfikatora k oblicza się jako sumę wag źle zaklasyfikowanych przez niego przykładów:

$$e_k = \sum_{o_i} w_i [c_i \neq c_i^k] \quad (3.22)$$

gdzie c_i^k oznacza wartość klasy przypisaną przez klasyfikator k , przykładowi o_i . Jeśli obliczony błąd należy do obustronnie otwartego przedziału $(0; 0.5)$ to przystępuje się do modyfikacji wag w_i . W przypadku przykładów poprawnie zaklasyfikowanych wagi mnoży się przez współczynnik:

$$\gamma_k = \frac{1 - e_k}{e_k} \quad (3.23)$$

a następnie normalizuje tak by $\sum_{o_i} w_i = 1$. Finalny wynik klasyfikacji przykładu o_i opiera się na sumie ważonej wszystkich klasyfikatorów a waga głosu wynosi $\log(\frac{1}{\gamma_k})$.

Las losowy jest to technika *baggingu* wykorzystująca specyficzną strategię budowy pojedynczego modelu jakim jest drzewo decyzyjne. Breiman po licznych badaniach porównawczych *baggingu* i *boostingu* opracował algorytm *baggingu* w oparciu o dość proste drzewa decyzyjne [Breiman, 2001]. By pojedyncze drzewo było klasyfikatorem możliwie niezależnym od innych drzew, podział w węźle jest realizowany na podstawie najlepszego atrybutu z wylosowanej każdorazowo puli m' atrybutów. W przeciwieństwie do procesu budowy pojedynczego drzewa decyzyjnego w przypadku wyboru testu do węzła celowo w sposób losowy zawęża się możliwości wyboru możliwie najlepszego testu maksymalizującego przyrost informacji w węźle. Dodatkowo drzewa

budowane w ramach lasu nie są przycinane co powoduje ich nadmierne dopasowanie do zbioru. Na drodze eksperymentalnej wyznaczono, że dla $m' = \sqrt{m}$ przeważnie uzyskuje się najlepsze wyniki klasyfikacji całego lasu. Zgodnie z ideą baggingu modele budowane są w oparciu o próbki danych losowanych ze zwracaniem, a końcowa klasyfikacja opiera się na równoprawnym głosowaniu wszystkich drzew.

3.7. Inne metody

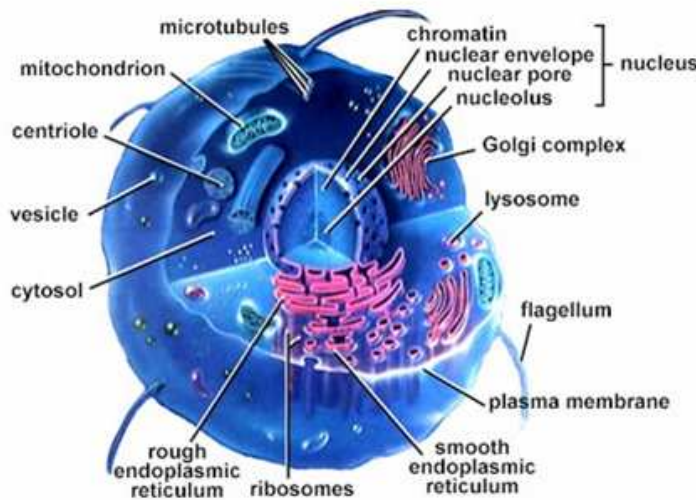
W opisie różnych metod klasyfikacji pominięto systemy oparte na logice rozmytej i sieci neuronowe choć stosuje się je również do klasyfikacji. Miały one jednak znikomy wpływ na ukształtowanie się zaproponowanych w pracy algorytmów i nie ma potrzeby opisu tych technik jako tła dla proponowanych rozwiązań.

Bardzo dobrymi książkami szeroko traktującymi temat systemów uczących się są [Cichosz, 2000], [Koronacki *et al.*, 2005] oraz [Mitchell, 1997]. Można w nich znaleźć bardziej szczegółowe opisy większości z omówionych w tym rozdziale technik i algorytmów.

4. Od komórki do ekspresji genów

Temat, który jest integralną i bardzo ważną częścią pracy, związany z wyborem informatywnych cech, został w dużej mierze zainspirowany jednym z problemów przed którym stoi ostatnio bioinformatyka. Chodzi o wybór spośród tysięcy genów tych niewielu, które wpływają na wystąpienie pewnych zjawisk (np. chorób) u pacjentów lub o selekcję istotnych aminokwasów w konkretnym badanym białku. Mimo, iż w pracy skupiono się właściwie na rozwiązaniu problemu od strony statystycznej i informatycznej (uczenia maszynowego) to czasami pojawiają się pojęcia z zakresu biologii komórkowej, które wymagają usystematyzowania oraz poprzedzenia krótkim wstępem. Niniejszy wstęp traktuje temat genetyki na poziomie bardzo ogólnym, niemniej jednak w zupełności wystarczającym do zrozumienia z punktu widzenia informatyki celu badań zamieszczonych w pracy.

Komórka

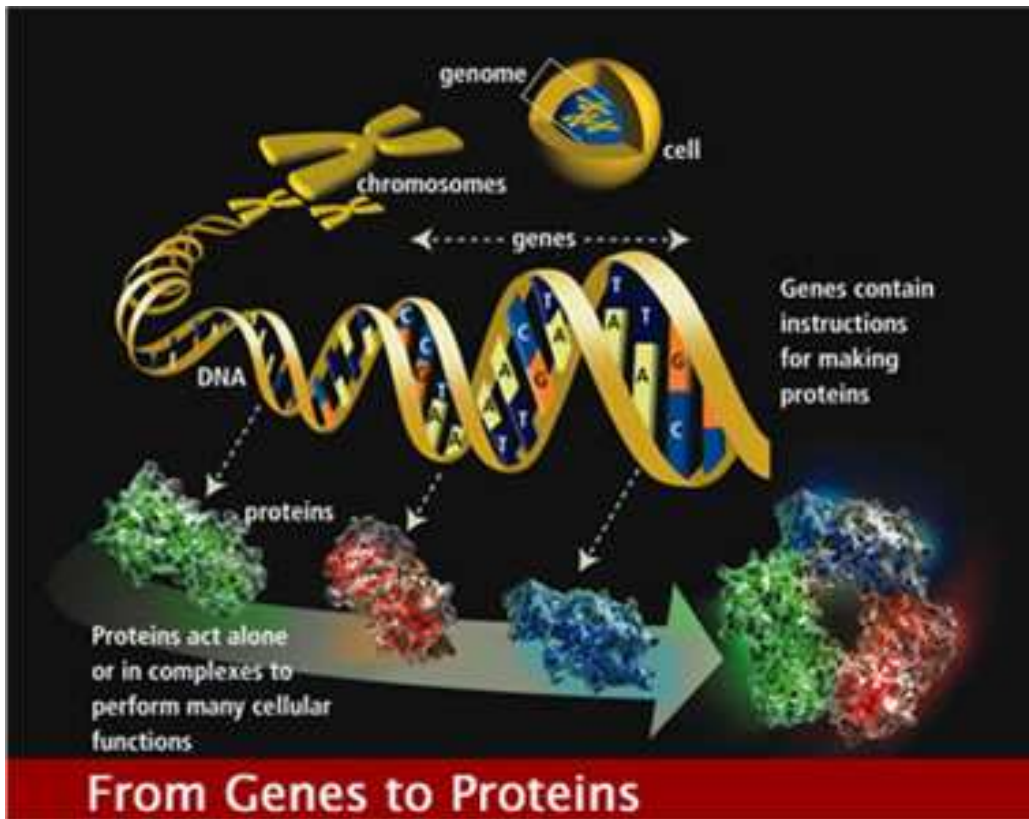


Rysunek 4.1. Schemat typowej komórki eukariota

Wszystkie organizmy żywe są zbudowane z maleńkich **komórek** (bardzo ogólny schemat komórki przedstawiono na rysunku 4.1, który pochodzi z [EBI]). Świat organizmów żywych dzielimy na świat *eukariota* i *prokariota*, a zasadniczą różnicą między nimi jest występowanie **jądra** komórkowego. Każda komórka świata *eukariota* zawiera wyodrębnione jądro, które jest odseparowane błoną oraz zawiera organelle: plastydy i mitochondria, które nie występują u organizmów świata *prokariota*. Do świata *eukariontów* zalicza się większość organizmów żywych poza bakteriami i sinicami.

W ciele człowieka znajduje się około $6 \cdot 10^{13}$ komórek, wśród których wyróżniamy ich około 320 różnych typów. Zbiór komórek jednego typu to **tkanka** i tak wyróżniamy np. tkankę nabłonkową, tkankę nerwową, tkankę łączną, krew itd.

Kwas nukleinowy *DNA/RNA*



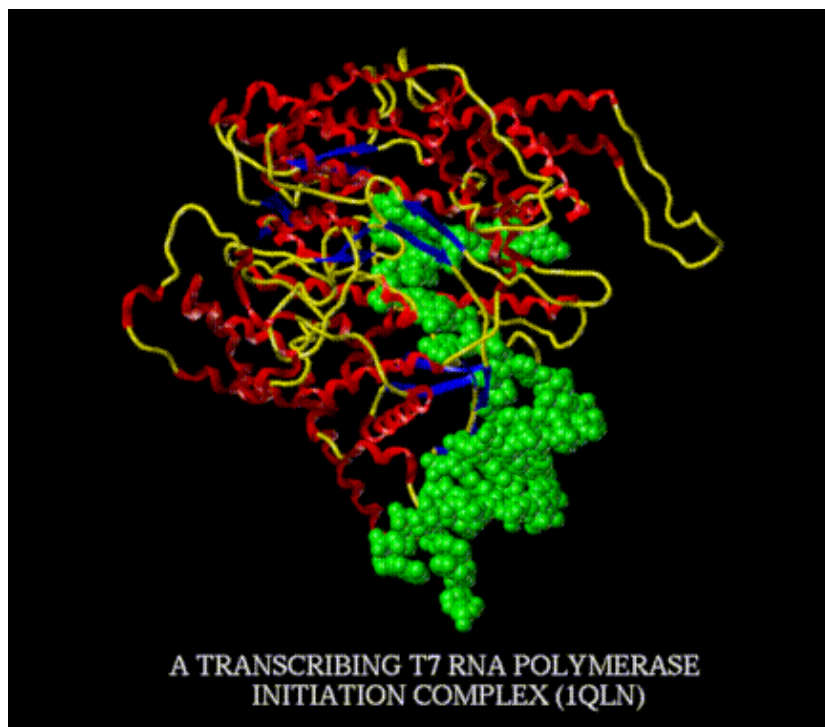
Rysunek 4.2. Schemat łańcucha DNA oraz genu

Kwas nukleinowy: deoksyrybonukleinowy/rybonukleinowy (skrótowo nazywany ***DNA/RNA*** przedstawiony na rysunku 4.2 [Human Genome Project]) jest podstawowym nośnikiem informacji w komórce. W przeciwieństwie do organizmów *eukariota*, gdzie *DNA* jest głównie skupione w jądrze komórkowym, u *prokariotów* kwasy nukleinowe występują w postaci luźno skupionego nukleoidu, nie oddzielonego od reszty komórki błoną jądrową.

DNA jest dwuwstęgowym łańcuchem cząstek zwanych **nukleotydami** i wyróżnia się ich cztery następujące typy: adenina (*A*), guanina (*G*), cytozyna (*C*), tymina (*T*). *RNA* jest jednowstęgowym odpowiednikiem *DNA* (gdzie można wyodrębnić dwa równoległe łańcuchy), w którym zamiast tyminy *T* występuje uracyl (*U*). Nukleotydy ułożone na przeciw siebie w *DNA* (lub dla *RNA* podczas transkrypcji t.j. przepisania *DNA* na *RNA*) mogą tworzyć jedynie ściśle określone pary tzn. *G-C* i *A-T* lub *A-U*. Ludzkie *DNA* zawiera około $3 \cdot 10^9$ par nukleotydów natomiast np. dla różnych bakterii jest to przedział pomiędzy 400 tysięcy do 10 milionów. *DNA* jest bardzo silnie skręcone i zorganizowane w postaci **chromosomów**, które znajdują się w jądrze komórkowym. Człowiek posiada 23 pary chromosomów, a np. bakte-

ria tylko 1 parę. U organizmów *eukariota* niewielka część *DNA* zawarta jest również w **mitochondriach** natomiast obie części, to znaczy: *DNA* z jądra oraz fragmenty z mitochondrium nazywamy **genomem**. Długość ludzkiego *DNA* po rozwinięciu wynosi około 2 metry i w 99.9% jest takie samo u wszystkich ludzi. Najbardziej zróżnicowanym fragmentem jest ten znajdujący się w mitochondriach.

Białko



Rysunek 4.3. Przykładowa struktura białkowa - polimeraza

Istnieje 20 różnych związków chemicznych zwanych **aminokwasami**. **Białko** (przykładowe przedstawione na rysunku 4.3 [BioInfo Bank]) jest zbudowane z dowolnej kombinacji liniowej aminokwasów i z tego powodu również nazywane jest łańcuchem polipeptydowym. Białka są podstawowym budulcem komórki (np. kolagen – obecny w tkance łącznej i kościach) oraz pełnią w jej działaniu zasadnicze funkcje (np. białkami są również enzymy służące do przeprowadzania/katalizacji reakcji biochemicznych – przykładowo do uruchomienia transkrypcji DNA służy polimeraza). Znamy około 15 tysięcy różnych struktur białkowych - insulina składa się z 51 aminokwasów, ale już titina z około 28 tysięcy.

Gen

Słowo **gen** pochodzi z języka greckiego "*genos*" i oznacza pochodzenie. Geny to regiony w *DNA/RNA* odpowiedzialne za zakodowanie poszczególnych białek (rysunek 4.2). W ludzkiej *DNA* jest około 6700 eksperymentalnie potwierdzonych genów, natomiast szacuje się, że jest ich około 25 tysięcy (choć tutaj nie ma pełnej zgodności). Wiemy, że mniej niż 5% z całego ludz-

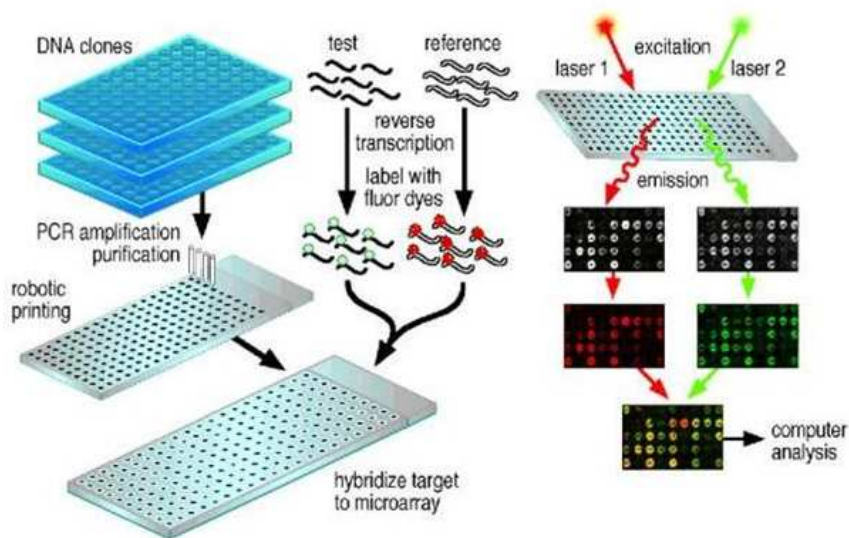
kiego genomu koduje białka, natomiast reszta w tej chwili jest przedmiotem badań, ponieważ jej funkcja nie jest jeszcze dokładnie znana. W jednej z teorii na temat pozostałych 95% *DNA* mówi się o bezużyteczności tej części i tylko jako pozostałości w ramach ewolucji człowieka – choć ostatnie badania wskazują, że nie jest to do końca prawdą.

Bakteria *E.Coli* ma szacunkowo tylko 5 tysięcy genów co stanowi z kolei aż 90% jej genomu. Fakt ten pokazuje, że wraz ze wzrostem skomplikowania organizmów rośnie ich liczba genów, które to stanowią coraz mniejszą część genomu. Trzy kolejne nukleotydy w genie nazywamy **kodonom** lub **tripletem**, a ich funkcją jest zakodowanie jednego aminokwasu. Skoro istnieją 4 różne nukleotydy to możliwych różnych kodonów jest $4^3 = 64$. Co ciekawe to przy istnieniu 20 aminokwasów wyraźnie widać, że kodowanie jest nadmiarowe – triplety *CAT* i *CAC* kodują ten sam aminokwas histydynę.

Proces **syntezy białka** polega na transkrypcji *DNA* na *mRNA* (messenger RNA), a następnie translacji poszczególnych kodonów na kolejne aminokwasy w celu budowy pojedynczego białka. W trakcie transkrypcji buduje się komplementarny łańcuch *mRNA* do łańcucha *DNA*, a katalizatorem procesu jest enzym (białko) zwany *polimerazą*.

Technologia mikromacierzy - ekspresja genów

Technologia mikromacierzy ma za zadanie pomiar aktywności poszczególnych genów w celu późniejszego zidentyfikowania w trakcie analizy danych, grupy genów, która ma istotny wpływ na obserwowany czynnik (np. rodzaj choroby).



Rysunek 4.4. Proces pozyskiwania danych o ekspresji genów

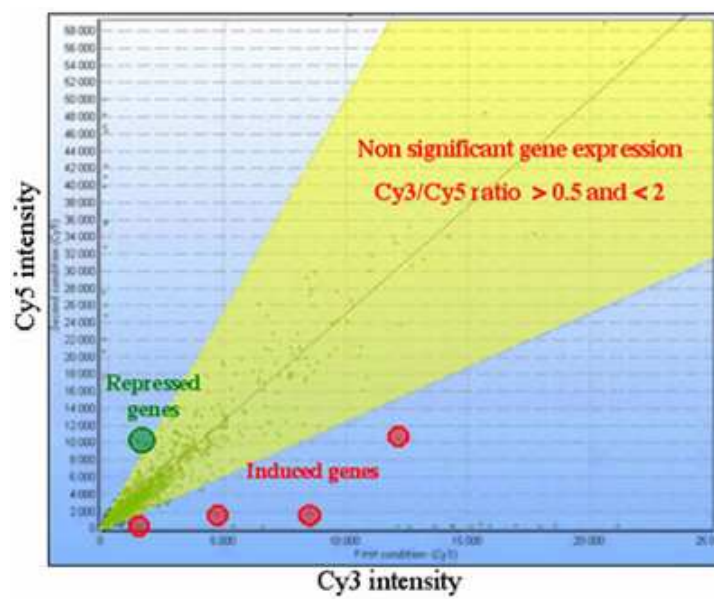
Mikromacierz (ang. microarray, *DNA chip*) jest to niewielkiego rozmiaru (1 2.5 cm x 1 2.5 cm) szklana lub nylonowa płytką. Płytką taką posiada kilka tysięcy (do 50 000) naniesionych przez robota w regularnych pozycjach mikroskopowego (30 pikolitrow) rozmiaru pól/kropli (ang. spots) zawierających specjalnie spreparowane różne cząsteczki *DNA*. Fragmenty te mają za

zadanie wykryć komplementarne do siebie cząsteczki kwasu nukleinowego - odpowiednie geny. Szacunkowo w jednym polu znajduje się 10^7 – 10^8 identycznych cząsteczek *DNA*.

Do przeprowadzenia pojedynczego procesu ekstrakcji ekspresji genów pobiera się dwie próbki tkanki chorą (np. rakową) oraz zdrową (rysunek 4.4 [Alsberg]). Następnie osobno dla każdej próbki w procesie transkrypcji otrzymuje się odpowiednio *mRNA* ze źródłowego *DNA*. Oczywiście na każdym etapie procesu liczba poszczególnych cząsteczek jest ogromna – porównywalna lub większą z liczbą cząsteczek na polu mikromacierzy. W kolejnym etapie cząsteczki *mRNA* pozyskane z obu tkanek, są znakowane odpowiednim barwnikiem, jedno na zielono drugie zaś na czerwono. Ściślej mówiąc pewien światłoczuły/fluorescencyjny związek chemiczny doczepiany jest do cząsteczek *mRNA* tworząc w dalszym procesie komplementarne *cDNA* (*colored DNA*). Następnie po zmieszaniu cząsteczek z obu próbek przeprowadza się proces hybrydyzacji na mikromacierzy. Polega to na połączeniu się odpowiednich komplementarnych fragmentów wyekstrahowanego *cDNA* do tych, przygotowanych wcześniej na odpowiednich polach mikromacierzy. W każdym z pól może zatem wystąpić bardzo wiele instancji identycznego genu. Następnie każde pole jest poddawane oświetleniu laserem raz czerwonym natomiast za drugim razem zielonym i wyniki są zapisywane. Powstają w ten sposób dwa obrazy z różną dla różnych genów intensywnością świecenia dwoma kolorami. Przewaga intensywności jednego z kolorów oznacza większą liczbę cząsteczek pobranych z odpowiednio oznaczonej tkanki (chorej lub zdrowej). Następnie oba obrazy są nakładane na siebie tworząc obraz intensywności odpowiednich barwników dla każdego z genów. Czarne pole oznacza brak zaobserwowanych cząstek *cDNA*, natomiast żółte mówi o podobnej koncentracji cząstek z obu pobranych próbek tkanek.

Cały proces jest oczywiście bardziej skomplikowany, lecz zaprezentowany opis ograniczający niuanse biotechnologiczne ma na celu jedynie w poglądowy sposób przedstawić sposób pozyskiwania jednego rekordu danych o ekspresji genów.

Po nałożeniu na siebie obu obrazów właściwie pozostaje jedynie pewne wyskalowanie otrzymanych danych. Natężenie (czerwieni/zieleni) sygnału (rysunek 4.5 [BDGS]) jest normalizowane w stosunku do różnych parametrów (liczby próbek poddanej każdemu czynnikowi, siły emisji czynników fluorescencyjnych, etc.). Zakładamy że ilość fluorescencyjnego *DNA* (*cDNA*) jest proporcjonalna do ilości *mRNA* obecnego w każdej komórce na początku procesu. Stosunek jasności zieleni do czerwieni mówi o nadmiernej aktywności jednego z czynników. Prosta analiza otrzymanych pojedynczych danych skupia się na nadmiernie świecących w stosunku do reszty genach.



Rysunek 4.5. Analiza natężenia poziomu świecenia poszczególnych genów

5. Indukcja reguł i klasyfikacja w algorytmie ADX

5.1. Definicje

Niech D oznacza zbiór danych zorganizowanych w formie tabeli (system informacyjny). Każdy wiersz w owej tabeli to przypadek e (zwany również obiektem lub przykładem *event*, *object*). Oznacza to, iż zbiór danych D jest zbiorem przypadków/obiektów. Zapis $|D|$ oznaczać będzie moc zbioru D , czyli liczbę obiektów w tabeli. Każda kolumna w D oznacza atrybut a (zwany również cechą lub zmienną), a wartości atrybutów opisują przykłady w D . Atrybuty dzielimy na kilka typów w zależności od dziedziny atrybutu (zbioru wszystkich możliwych wartości).

Ze względu na typ atrybuty można podzielić na:

- nominalne - gdzie lista możliwych wartości ogranicza się do zbioru trudnego bądź niemożliwego do uporządkowania (np. kolor oczu pacjenta),
- numeryczne - gdzie dziedziną są wartości liczbowe (np. wzrost podany w centymetrach), tu można wyróżnić dodatkowo podział ze względu na skalę przedziałową lub ilorazową,
- porządkowe - atrybut taki łączy w sobie cechy obu omówionych wcześniej typów tzn opisuje obiekt poprzez etykiety lecz zbiór wszystkich możliwych wartości daje się uporządkować (np. wzrost pacjenta wyrażony w formie opisu wysoki, średni, niski),
- cykliczne - atrybut porządkowy o cyklicznym odwołaniu (np. dni tygodnia).

W praktyce w znakomitej większości przypadków tabel z danymi podczas definiowania typów atrybutów można się właściwie ograniczyć wyłącznie do dwóch pierwszych typów.

Zakłada się, że D zawiera jeden specjalny atrybut o znaczeniu fundamentalnym z punktu widzenia klasyfikacji zwany decyzyjnym. Wartości atrybutu decyzyjnego d definiują klasę przykładu i proces klasyfikacji ma na celu ich przewidywanie. Uczenie klasyfikatora polega zatem na znalezieniu i późniejszym wykorzystaniu ogólnych zależności (liniowych oraz nieliniowych) pomiędzy atrybutem d a pozostałymi cechami opisującymi przykłady w D .

Selektor (*selector*) s jest wyrażeniem, które opisuje zbiór przykładów poprzez ograniczenie/określenie wybranych wartości atrybutu ze wszystkich możliwych jego wartości np.: *kolor = 'niebieski'* lub *waga > 80* wybierając przykłady spełniające zadany warunek logiczny. Prosty selektor zawiera nazwę atrybutu, operator arytmetyczny oraz wartość jaką może przyjąć atry-

a_1	a_2	$\dots$	a_n	d
v_{11}	v_{12}	$\dots$	v_{1n}	d_1
v_{21}	v_{22}	$\dots$	v_{2n}	d_2
v_{31}	v_{32}	$\dots$	v_{3n}	d_3
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
v_{m1}	v_{m2}	$\dots$	v_{mn}	d_m

Tablica 5.1. Tabela danych D

but. Bardziej skomplikowany selektor może zawierać listę wartości (nominalnych lub numerycznych) np. $kolor = ['niebieski', 'zielony']$ lub może określać przedział liczbowy np. $waga = (70; 80]$. Zapis $']'$ oznacza przedział domknięty prawostronnie. Oba przedstawione nieproste selektory można przedstawić jako alternatywy (sumy logiczne) lub koniunkcje (iloczynny) odpowiednich selektorów prostych.

Przedstawiony selektor $kolor = ['niebieski', 'zielony']$ można zastąpić sumą selektorów $kolor = niebieski \vee kolor = zielony$, natomiast selektor $waga = (70; 80]$ można przedstawić jako koniunkcję $waga > 70 \wedge waga \leq 80$. Selektor bez żadnego warunku narzuconego na atrybut nazwiemy ogólnym lub uniwersalnym, a formalnie zapiszemy $atrybut = *$ (np. $kolor = *$ oznacza jakikolwiek kolor).

Niech **kompleks** (*complex*) c oznacza zbiór selektorów i niech długość n kompleksu charakteryzuje liczbę selektorów (prostych lub złożonych z wyjątkiem ogólnych) zawartych w c . Przykładowo kompleks $c = \langle s_1, s_2, s_3 \rangle$ ma długość $n=3$. Kompleks rozumiany jest jako koniunkcja selektorów wymienionych w jego definicji.

Pokrycie (*coverage*) cov selektora s jest to iloraz liczby przykładów, które wspierają warunek zawarty w s (przykłady takie są pokrywane przez s i oznaczmy je poprzez D^s) i liczby $|D|$. Pokrycie dowolnego selektora ogólnego jest równe 1. Analogicznie, pokrycie cov kompleksu c jest to liczba przykładów, które wspierają równocześnie wszystkie selektory w c podzielona przez $|D|$.

$$cov(s) = \frac{|D^s|}{|D|}, \quad (5.1)$$

$$cov(c) = \frac{|D^c|}{|D|} \quad (5.2)$$

Zbiór pozytywny (D_p) dla danej klasy jest to zbiór przykładów, dla których atrybut decyzyjny zawiera wartość korespondującą z tą klasą. Podczas tworzenia reguł dla danej wartości v atrybutu decyzyjnego d , wszystkie przykłady oznaczone przez v są nazywane pozytywnymi dla d . Natomiast wszystkie pozostałe przykłady są nazywane negatywnymi (D_n) dla d .

Pokrycie pozytywne $pcov$ rozpatrywanego kompleksu dla zadanej klasy jest pokryciem wyliczonym na zbiorze ograniczonym tylko do przykładów

pozytywnych (tzn. przykładów z klasy dla której w danym momencie buduje się model). **Pokrycie negatywne** $ncov$ kompleksu jest wyliczane natomiast dla wszystkich przykładów takich, które należą do innej klasy niż rozpatrywana. Zbiór przykładów ze zbioru pozytywnego pokrywany przez kompleks c oznaczmy jako D_p^c i analogicznie dla zbioru negatywnego jako D_n^c . Odpowiednio dla selektora zbiór przykładów pozytywnych pokrywanych przez niego to D_p^s , a negatywnych D_n^s .

$$pcov(s) = \frac{|D_p^s|}{|D_p|}, pcov(c) = \frac{|D_p^c|}{|D_p|} \quad (5.3)$$

$$ncov(s) = \frac{|D_n^s|}{|D_n|}, ncov(c) = \frac{|D_n^c|}{|D_n|} \quad (5.4)$$

$$cov(s) = \frac{|D_p^s| + |D_n^s|}{|D|}, cov(c) = \frac{|D_p^c| + |D_n^c|}{|D|}, \quad (5.5)$$

$$prob_p(c) = \frac{|D_p^c|}{|D_p^c| + |D_n^c|} \quad (5.6)$$

Kompleks dla którego policzono $pcov$ i $ncov$ można nazwać regułą. Reguła taka składa się wtedy z wyrażenia logicznego wyrażonego poprzez kompleks oraz przyporządkowanych statystyk ($pcov$, $ncov$, cov) określających odpowiednie pokrycia a tym samym istotność reguły. Z definicji 'mocne reguły' determinują klasę, a ich negatywne pokrycie wynosi 0. Kompleks dla którego $pcov=1$ a $ncov=0$ będzie zatem silną regułą implikującą rozpatrywaną klasę, dla której prawdopodobieństwo a posteriori $prob_p(c)$ zaistnienia tej klasy będzie w zasadzie równe 1. Należy tu jednak pamiętać, że prawdopodobieństwo to dotyczy się zbioru, na którym takie wnioski konstruujemy. W zasadzie dysponując próbą danych trudno jest określić w jakim stopniu odzwierciedla on całą dziedzinę. Można jedynie stwierdzić iż jeśli dysponujemy zbiorem odpowiednio reprezentatywnym to reguły i ich statystyki (a tym samym oszacowane prawdopodobieństwa) wystarczająco odzwierciedlają rzeczywistość. Przeważnie dla danych rzeczywistych silne reguły są dużą rzadkością, zatem wnioskowanie opiera się raczej na kompleksach, dla których $ncov \geq 0$ oraz $pcov > ncov$. Reguły takie wciąż opisują zadaną klasę lecz determinują ją z prawdopodobieństwem mniejszym od 1.

Zbiór kompleksów (suma logiczna) opisujących jedną klasę nazywamy zestawem reguł (*ruleset*). Wszystkie zestawy reguł zbudowane dla wszystkich możliwych wartości decyzji nazywamy rodziną zestawów reguł (*ruleset family*).

5.2. Tworzenie zestawu reguł (ruleset) dla jednej klasy

Cały proces tworzenia reguł decyzyjnych w ADX jest oparty na założeniu, że koniunkcja selektorów nie może mieć większego pokrycia (cov) niż minimalne pokrycie spośród tych selektorów. Podstawowym celem algorytmu jest znalezienie najlepszego zestawu reguł - reguł które maksymalizują $pcov$ i równocześnie minimalizują $ncov$. Jeśli taki zestaw kompleksów istnieje

(zostanie odkryty w procesie uczenia), to w dalszym procesie będzie on służył do klasyfikacji. Klasyfikacja będzie się opierać w pewnym sensie na mierzeniu podobieństwa nowego nieoznaczonego przykładu do kolejnych zestawów reguł spośród całej rodziny. Największe podobieństwo spośród wyliczonych dla każdego zestawu determinuje klasę, jaką zostanie oznaczony nowy przykład. Oczywiście w całym procesie uczenia i klasyfikacji za pomocą algorytmu ADX zakłada się, że dane rzeczywiste (z praktyki wiadomo, że bardzo słusznie) mogą zawierać wartości nieznane, zaszumione lub sprzeczności (identyczne przykłady o różnej przynależności do klas).

Algorytm ADX nie stara się budować bardzo długich, skomplikowanych i silnie dopasowanych reguł z pokryciem negatywnym $ncov = 0$ (mocnych reguł). Jeśli takie reguły z odpowiednio wysokim $pcov$ zostaną odkryte to oczywiście zasilą bazę finalnych reguł. W ADX tworzy się zestaw wielu lecz raczej prostych reguł z możliwie wysokim $pcov$, a zarazem możliwie niskim $ncov$. Takie podejście pozwala na późniejszą klasyfikację nowego obiektu w oparciu o pokrywający go podzbiór reguł, a nie tylko na podstawie jednej jak to ma miejsce w ujęciu, gdzie jeśli obiekt spełnia warunek to przypisuje się mu odpowiednią etykietę klasy.

Można powiedzieć, że $pcov$ oraz $ncov$ odzwierciedlają odpowiednie prawdopodobieństwa *a priori* pokrywania przykładu przez kompleks pod warunkiem wystąpienia klasy pozytywnej bądź negatywnej. Często problemem systemów opartych na regułach decyzyjnych (również eksperckich) jest brak możliwości sklasyfikowania nowego przykładu ponieważ żadna reguła nie pokrywa przykładu. Mocne reguły tworzone na zbiorze trenującym mają tendencję do nadmiernego dopasowania (przeuczenia) co prowadzi do późniejszych problemów w klasyfikacji. Najczęściej omija się ten problem w dwojaki sposób. Po pierwsze jeśli żadna reguła nie pokrywa nowego przykładu to można liczyć pewne podobieństwo do lub odległość od istniejących reguł. Następnie nadaje się etykietę klasy, którą wskazuje reguła najbliższa/ najbardziej podobna do przykładu. Można zauważyć, że takie podejście operuje na rozmytym pokryciu. Drugim sposobem jest odpowiednie przycinanie reguł w momencie procesu uczenia i takie przycinanie jest tożsame z przycinaniem drzew decyzyjnych.

Tworzenie reguł decyzyjnych ma zatem dwa podstawowe cele:

- Generalizację wiedzy.
- Wygodną interpretację zgromadzonej wiedzy przez człowieka.

Pierwszy z wymienionych celów wymaga kilku słów komentarza. Generalizacja wiedzy ma za zadanie odkrycie pewnych wzorców z próbki uczącej, które (jeśli próba jest dostatecznie reprezentatywna) powinny się regularnie pojawiać w całej populacji przykładów. W najprostszym ujęciu system regułowy powinien wygenerować liczbę istotnych reguł mniejszą niż liczba przykładów w zbiorze uczącym, przy okazji pokrywając cały (lub prawie cały) zbiór przykładów. Najlepszym sprawdzeniem czy wiedza jest dostatecznie ogólna jest oczywiście klasyfikacja zbioru testowego. Im więcej przykładów system jest w stanie bezbłędnie sklasyfikować, tym można mieć silniejsze przekonanie, że stworzone reguły odzwierciedlają wzorce w danych.

Ze względu na wymienione problemy ADX skupia się na tworzeniu zbioru reguł, który niejako ma generalizować wiedzę. Zbiór reguł nie jest oparty tylko i wyłącznie na mocnych regułach a raczej na regułach, które pokrywają jak najwięcej przykładów pozytywnych, a jednocześnie ich zdolności predykcyjne są wysokie. W ADX nie ma potrzeby przycinania reguł ponieważ reguły są raczej proste, krótkie i możliwie ogólne. W dalszych rozważaniach zakłada się, że wejściowe dane numeryczne/ilościowe zostały poddane operacji dyskretyzacji, a implementacja algorytmu ADX w JAVA ma odpowiednie techniki dyskretyzacji wbudowane i działające automatycznie (patrz sekcja 5.7).

Kolejne opisane kroki algorytmu są osobno stosowane dla każdej wartości decyzji. Ich efektem jest stworzenie zestawu reguł opisujących jedną klasę ze zbioru wejściowego. Bez względu na liczbę klas rozpatrywana wartość decyzji jest przeciwstawiana pozostałym przykładom traktowanym w danej chwili jako negatywne. Przykłady pozytywne natomiast to wszystkie związane z daną wartością decyzji.

5.2.1. Krok pierwszy — znalezienie bazy selektorów

Rozważmy następujący przykładowy zbiór uczący przedstawiony w tabeli 5.2.

a_1	a_2	a_3	d
3	3	1	a
3	1	0	c
3	3	0	a
4	2	1	a
3	1	1	c
5	2	0	b
3	3	1	b
5	1	0	a
1	2	1	c
5	1	1	a

Tablica 5.2. Zbiór uczący D

Dla danej wartości zmiennej decyzyjnej (klasy) dla każdej wartości każdego atrybutu w D algorytm oblicza odpowiednie pokrycia pozytywne oraz pokrycia negatywne ($pcov$ oraz $ncov$). Oznacza to zbudowanie zbioru odpowiednich tablic od a_1 do a_m (gdzie m oznacza liczbę atrybutów wyłączając atrybut d). Przykład policzonych $pcov$ oraz $ncov$ dla tabeli danych 5.2 przedstawiono w tabeli 5.3.

Można teraz zdefiniować parametr *minSelectorPosCoverage*. Parametr określa minimalne akceptowalne pokrycie pozytywne jakie może przyjąć se-

value	<i>pcov</i>	<i>ncov</i>
1	0.0	0.2
3	0.4	0.6
4	0.2	0.0
5	0.4	0.2

Tablica 5.3. Tablica pokryć dla przykładowego atrybutu a_1 z D dla wartości decyzji "a"

lektor. Wszystkie selektory z pokryciem mniejszym niż zadane w parametrze są usuwane. Na przykład selektory $a_1 = 3$ ($pcov = 0.4$) i $a_1 = 5$ ($pcov = 0.4$) spełniają warunek $minSelectorPosCoverage = 0.2$. Wszystkie selektory spełniające warunek zdefiniowany przez $minSelectorPosCoverage$ są bazą dla następnego kroku. Warto zwrócić uwagę iż w bazie selektorów są wyłącznie selektory proste. Wielkość bazy selektorów (liczba selektorów w bazie) ma krytyczne znaczenie w dalszej kolejności dla złożoności obliczeniowej.

Drugim możliwym kryterium wyboru selektorów do bazy jest ich maksymalna liczba w bazie. Parametr *searchBeam* (szerokość wiązki przeszukiwania) definiuje zatem taką liczbę selektorów i spośród wszystkich ocenionych wybierane są najlepsze według pewnego kryterium opartego na *pcov* i *ncov*. Kryterium to zostanie zdefiniowane w dalszej części opisu algorytmu ADX, w trakcie omawiania sposobu oceny kompleksów.

5.2.2. Krok drugi — tworzenie zestawu kompleksów

Można zauważyć, że baza selektorów jest zbiorem kompleksów o długości równej 1. Dłuższe kompleksy są tworzone poprzez proces kolejnych iteracji. Każda iteracja składa się z następujących kroków, które będą objaśnione w dalszym ciągu paragrafu:

1. Utworzenie zbioru kandydatów (każdy z nich jest dłuższy o 1 prosty selektor niż kompleksy z poprzedniej iteracji).
2. Ocena jakości kandydatów.
3. Usunięcie zbędnych kompleksów.
4. Wybór zbioru rodziców.
5. Sprawdzenie kryterium stopu algorytmu.

Utworzenie zbioru kandydatów

Nowo utworzone kompleksy dla których nie obliczono jeszcze jakości nazwiemy tutaj kandydatami. Tworzenie kompleksów o długości 2 jest dość proste i opiera się na bazie selektorów. Zbiór kandydatów tworzy się na podstawie wszystkich możliwych par selektorów z bazy ograniczonych dwoma prostymi założeniami. Po pierwsze nie tworzy się kompleksów z selektorów opartych na tym samym atrybucie. Po drugie nie tworzy się więcej niż jeden kompleks opartych na dokładnie tych samych selektorach. Kompleks jako koniunkcja

lista rodziców	operacja	lista kandydatów
$c_1 = \langle s_1, s_2, s_3 \rangle$		
$c_2 = \langle s_2, s_3, s_5 \rangle$	$c_1 \otimes c_2$	$c_6 = \langle s_1, \mathbf{s}_2, \mathbf{s}_3, s_5 \rangle$
$c_3 = \langle s_2, s_3, s_8 \rangle$	$c_2 \otimes c_3$	$c_7 = \langle s_1, \mathbf{s}_2, \mathbf{s}_3, s_8 \rangle$
$c_4 = \langle s_3, s_4, s_7 \rangle$	$c_1 \otimes c_3$	$c_8 = \langle \mathbf{s}_2, \mathbf{s}_3, s_5, s_8 \rangle$
$c_5 = \langle s_3, s_5, s_7 \rangle$	$c_2 \otimes c_5$	$c_9 = \langle s_2, \mathbf{s}_3, \mathbf{s}_5, s_7 \rangle$

Tablica 5.4. Przykład listy kandydatów wygenerowanych na podstawie kompleksów rodziców

wyrażeń logicznych nie rozróżnia ich kolejności (kompleks $c_1 = \langle s_1, s_2 \rangle$ jest zatem równoważny $c_2 = \langle s_2, s_1 \rangle$).

Należy tutaj jeszcze omówić proces tworzenia kandydatów dłuższych niż opartych na 2 selektorach. Do zbudowania nowego kompleksu kandydata, potrzeba dwóch kompleksów rodziców obu o długości mniejszej o 1. Rodzice muszą mieć część wspólną o długości $n - 1$ (gdzie n oznacza ich długość - liczbę selektorów).

Na przykład kompleks $c_1 = \langle s_1, s_2, s_3 \rangle$ i $c_2 = \langle s_2, s_3, s_5 \rangle$ mogą zostać użyte do zbudowania nowego $c_3 = \langle s_1, s_2, s_3, s_5 \rangle$. Tworzenie kompleksów o długości 2 jest oparte na bazie prostych selektorów (w gruncie rzeczy kompleksów o długości 1), tworzenie kompleksów o długości 3 jest oparte na kompleksach o długości 2 itd. Przykład bazy kompleksów rodziców i możliwych do wygenerowania na ich podstawie kandydatów znajduje się w tabeli 5.4. Symbolem $\otimes$ oznaczono operację tworzenia kompleksu kandydata w oparciu o kompleksy rodziców.

Ocena jakości kandydatów

Po stworzeniu każdego zbioru kandydatów należy osobno ocenić jakość nowo utworzonych kompleksów. Podczas procesu oceny, dla każdego kandydata oblicza się jego pozytywne i negatywne pokrycie ($pcov$ i $ncov$) na zbiorze uczącym. Realizuje się to poprzez policzenie liczby przykładów pokrywanych przez oceniany kompleks odpowiednio z rozpatrywanej klasy decyzyjnej oraz poza nią. Mając liczbę pokrywanych przykładów wystarczy je podzielić przez odpowiadające im licznosci klas.

Dla tablicy z przykładu 5.1 przy decyzji a , kompleks $c_1 = \langle a_1 = 3, a_2 = 3 \rangle$ ma odpowiednio pokrycie pozytywne $pcov = 2/5 = 0.4$, a pokrycie negatywne $ncov = 1/5 = 0.2$. Na podstawie $pcov$ i $ncov$ oblicza się pewną miarę oceny kompleksu Q . W niniejszej pracy zaproponowano kilka takich miar i są one szerzej omówione w sekcji 5.3. W dalszym procesie zbiór najwyższej ocenionych kompleksów staje się bazą rodziców w kolejnej iteracji budowy reguł i na ich podstawie tworzy się kompleksy dłuższe o 1.

Usuwanie zbędnych kompleksów

Pośród wielu wygenerowanych kompleksów część stanowi grupa takich, które nie pokrywają ani jednego pozytywnego przykładu (pokrycie pozytywne $pcov = 0$). Dodając do kompleksu dodatkowy jeden selektor (wydłużając kompleks) można jedynie zmniejszyć lub pozostawić na uprzednim poziomie związane z kompleksem pokrycia. Zatem kompleksy o $pcov = 0$, bez względu na ich $ncov$, są bezużyteczne z punktu widzenia rozpatrywanej klasy i wymagają usunięcia. Usunięcie ich z pamięci jest potrzebne gdyż przeważnie grupa kandydatów jest bardzo liczna, a dużą jej część stanowią właśnie takie zbędne kompleksy. Część najlepszych kompleksów o dostatecznie wysokim Q jest wybierana jako rodzice do kolejnej iteracji, natomiast reszta o $pcov$ większym od zera pozostaje do fazy finalnego wyboru omówionej w sekcji 5.2.4.

W tej fazie algorytmu stosuje się jeszcze jedno - bardziej zaawansowane kryterium usuwania zbędnych kompleksów. Kryterium to jest sterowane poprzez parametr *cleanCandidates*. Parametr ten może przyjmować następujące wartości:

- 0 – Dodatkowe kryterium jest zablokowane.
- 1 – Usuwane są kompleksy, które nie wykazują poprawy w stosunku do selektorów z których są zbudowane.
- 2 – Usuwane są kompleksy, które nie wykazują poprawy w stosunku do kompleksów rodziców.

Kompleks wykazuje poprawę jeśli jego Q jest większe od Q przynajmniej jednego z odpowiednich kompleksów rodziców, bądź selektorów składowych. Przy czym miara oceny jakości kompleksu/selektora jest omówiona w sekcji 5.3.

Wybór rodziców

Po zbudowaniu zbioru kandydatów, a następnie po dokonaniu oceny każdego z nich przeprowadza się selekcję kompleksów/rodziców na podstawie których zbudowani zostaną kolejni kandydaci. Podstawowym kryterium wyboru jest jakość kompleksu Q oparta na wyliczonych $pcov$ i $ncov$. Dodatkowo stosuje się tutaj pewne kryteria, które mają zapobiec budowaniu niepotrzebnych kandydatów. Faza wyboru rodziców jest zatem sterowana dodatkowo poprzez parametry *complexGenerality*, *searchBeam*, i *cleanCandidates*.

Parametr *searchBeam* definiuje maksymalną liczbę rodziców, którzy zostaną wybrani. Pośrednio parametr ten definiuje minimalną akceptowalną wartość Q dla danego zbioru kompleksów. Kompleksy są sortowane pod względem jakości po czym określa się Q kompleksu będącego na pozycji *searchBeam* na liście. Jakość takiego kompleksu oznaczmy jako $minQ$ – minimalną dopuszczalną jakość dla rodzica określającą poziom odcięcia. Dla zbiorów o niewielkiej liczbie obiektów zdarza się jednak, że istnieje duża liczba kompleksów dla których Q wynosi dokładnie $minQ$. Szczególnie taka sytuacja jest niedopuszczalna gdy liczba wszystkich kompleksów o $Q \geq minQ$ znacznie przekracza wartość określoną poprzez *searchBeam*, ponieważ grozi to eksplozją kombinatoryczną w kolejnej fazie budowania kandydatów. Liczba generowanych kandydatów zależy w dużej mierze od parametru *searchBeam*

(liczby rodziców), natomiast od liczby kandydatów zależy czas budowania pojedynczego zestawu reguł (wydłuża się czas oceny kandydatów, łączenia reguł, finalnego wyboru etc.). Parametr *searchBeam* z racji tego, iż steruje dokładnością oraz czasem budowania pojedynczego zestawu reguł musi wybierać dokładnie ustaloną liczbę rodziców bez względu na charakter danych. Jeśli zatem liczba rodziców wyznaczana za pomocą *minQ* przekracza zdefiniowaną poprzez parametr *searchBeam*, następuje losowy wybór kompleksów o $Q = \min Q$, tak by liczba rodziców odpowiadała dokładnie wartości *searchBeam*.

lista kompleksów	miara jakości Q	
c₁	Q₁₀	
c₂	Q₉	
c₃	Q₉	
c₄	Q₈	
c₅	Q₇	
c₆	Q₇	
c₇	Q₆	
c₈	Q₆	
c₉	Q₅	
c ₁₀	Q ₅	<i>minQ=Q₅</i>
c₁₁	Q₅	
c ₁₂	Q ₅	
c ₁₃	Q ₄	
c ₁₄	Q ₃	

Tablica 5.5. Sposób doboru wartości *minQ* oraz kompleksów rodziców.

W tabeli 5.5 zaprezentowano przykład wyboru rodziców przy parametrze *searchBeam* ustawionym na 10. Kompleksy w tabeli zostały uporządkowane według miary Q (wyższa wartość indeksu Q oznacza wyższą jej wartość). Wartość Q dla kompleksu dziesiątego w hierarchii wynosi Q_5 , lecz taką wartością odznaczają się c_9 , c_{10} , c_{11} oraz c_{12} . By łączna liczba wybranych rodziców wynosiła 10 losuje się dwa spośród czterech kompleksów o wartości Q_5 . Wszyscy rodzice zostali zaznaczeni w tabeli pogrubionym tekstem.

Parametr *complexGenerality* ogranicza tworzenie nowych kompleksów, które nie mogą być już lepsze niż ich rodzice. Jeśli parametr przyjmuje wartość 1, wtedy zabronione jest używanie kompleksów o $ncov = 0$ jako rodziców dla następnej fazy. W przeciwnym wypadku kompleksy o $ncov = 0$ mogą zostać wyselekcjonowane jako rodzice. Każdy kompleks oparty na rodzicu o zerowym pokryciu negatywnym może przyjmować co najwyżej $pcov$

równe rodzicowi. Oznacza to, iż jakość określana poprzez miary oparte na pokryciach w zasadzie nie może już ulec poprawie. Niemniej jednak dłuższe kompleksy dostarczają pełniejszego opisu, który z powodzeniem może być użyty w klasyfikacji.

Parametr ten wpływa zatem na długość otrzymywanych kompleksów, przy czym jeśli wynosi on 0 to zgadzamy się na budowanie kompleksów dłuższych, kosztem ewentualnego pogorszenia jakości. Przy ustalonym *searchBeam* *complexGenerality* = 0 sprzyja eksploatacji (staramy się "wydłużyć" maksymalnie dobre kompleksy). Jeśli natomiast parametr *complexGenerality* jest równy 1 to budowa zestawu reguł jest nacechowana eksploracją (rodzicami stają się kompleksy, które poprzez koniunkcję z kolejnym selektorem mogą przyjąć niższe *ncov*). Ustawienie *complexGenerality* = 0 jest domyślne w algorytmie, ponieważ prowadzi ono do budowy dłuższych i bardziej specyficznych kompleksów, które odgrywają istotną rolę podczas procesu klasyfikacji.

Kryterium stopu

Tworzenie kolejnych kandydatów może się zakończyć, gdy zostanie spełniony przynajmniej jeden następujących z warunków:

- Długość aktualnie ocenionych kompleksów jest równa liczbie atrybutów.
- Bieżąca baza ocenionych kompleksów jest pusta.
- Nie jest możliwe stworzenie żadnych nowych kandydatów w oparciu o zbiór rodziców.

Pierwszy warunek wynika z założenia, że kompleks nie może się składać z kilku selektorów tego samego atrybutu. W związku z tym liczba możliwych selektorów w kompleksie jest równa liczbie atrybutów wejściowych (bez decyzyjnego). W drugim wypadku, jeśli aktualna baza kompleksów (po usunięciu zbędnych o *pcov* = 0) zawiera mniej niż dwa kompleksy to nie ma możliwości generacji kolejnych kandydatów. Trzeci warunek odnosi się do ocenionej grupy kompleksów. W sytuacji jeśli spośród grupy rodziców żadna para nie ma części wspólnej o długości $n - 1$ również nie ma możliwości wytworzenia kolejnych kandydatów. We wszystkich trzech przypadkach kryterium stopu odnosi się tylko i wyłącznie do iteracyjnego generowania reguł i po jego spełnieniu przystępuje się do fazy łączenia kompleksów.

5.2.3. Krok trzeci — sklejanie kompleksów

Jeśli którykolwiek z omówionych wcześniej warunków stopu zostanie spełniony wtedy kończy się faza tworzenia zestawu kompleksów i algorytm przechodzi do fazy łączenia reguł. W tym kroku zmniejsza się ogólna liczba wygenerowanych reguł i zarazem zwiększa się odpowiednie pokrycie połączonych. Każde dwa kompleksy, które są oparte na takich samych atrybutach można ze sobą połączyć poprzez sumę logiczną odpowiednich selektorów. Połączenie dwóch kompleksów powoduje zdefiniowanie nowych selektorów, a one jako listę wartości przyjmują alternatywę wartości poszczególnych selektorów składowych. Sklejenie powoduje również usunięcie kompleksów składowych ze zbioru, ponieważ zakłada się (bazując na kryteriach sklejanie omówionych dalej), iż nowy połączony kompleks jest lepszy od bazowych.

lista kompleksów	operacja	lista połączonych kompleksów
$c_1 = \langle a_1 = 2, a_2 = 1, a_3 = 0 \rangle$		
$c_2 = \langle a_1 = 3, a_2 = 1, a_3 = 0 \rangle$	$c_1 \oplus c_2$	$c_6 = \langle a_1 = [2, 3], a_2 = 1, a_3 = 0 \rangle$
$c_3 = \langle a_2 = 3, a_3 = 7, a_5 = 1 \rangle$	$c_3 \oplus c_4$	$c_7 = \langle a_2 = 3, a_3 = [7, 1], a_5 = [1, 0] \rangle$
$c_4 = \langle a_2 = 3, a_3 = 1, a_5 = 0 \rangle$		
$c_5 = \langle a_3 = 6, a_4 = 3, a_5 = 3 \rangle$		

Tablica 5.6. Przykład łączenia kompleksów

Niech operacja sklejanie dwóch kompleksów będzie oznaczana jako $\oplus$. Następujące kompleksy $c_1 = \langle A = 1, B = 3, C = 4 \rangle$ i $c_2 = \langle A = 1, B = 3, C = 7 \rangle$ po połączeniu utworzą nowy kompleks $c_3 = c_1 \oplus c_2 = \langle A = 1, B = 3, C = [4, 7] \rangle$, gdzie $C = [4, 7]$ będzie nowym selektorem w bazie. Dla wynikowego kompleksu $c_{1,2}$ pokrycie negatywne i pozytywne będą sumą wartości odpowiednich pokryć dla poszczególnych łączonych kompleksów. Inne przykłady łączenia kompleksów znajdują się w tabeli 5.6.

W implementacji algorytmu odpowiedni parametr pozwala na sterowanie procesem łączenia reguł. Dzięki niemu można wyłączyć ten etap, sklejać zawsze, bądź wymusić sprawdzenie czy nowy sklejony kompleks ma wyższą jakość Q w stosunku do kompleksów bazowych. Jeśli po ewentualnym połączeniu nie uzyskamy podniesienia jakości kompleksu w stosunku do przynajmniej jednego z rodziców to sklejenie się nie odbywa.

5.2.4. Krok czwarty — finalna selekcja zestawu reguł

Po etapie budowy reguł należy dokonać ich wyboru do finalnego zestawu. W zależności od zbioru danych oraz ustawionego parametru *searchBeam* w pamięci zawarta jest dość duża liczba reguł. Po analizie przedstawionego procesu tworzenia reguł widać również, że wiele z nich jest w pewnym sensie redundantnych i dla przejrzystości opisu należy ograniczyć ich liczbę. Ograniczanie to jest operacją analogiczną do przycinania drzewa decyzyjnego oraz przynosi podobny efekt - uogólnienie wiedzy, przy jednoczesnym podniesieniu jakości klasyfikacji.

Spośród wszystkich wygenerowanych i nie usuniętych dotąd reguł wybiera się dla określonej wartości decyzji finalny zestaw. Selekcji podlegają tutaj również kompleksy, które nie zostały nigdy rodzicami ale nie oceniono ich jako bezużyteczne. Proces selekcji jest bardzo istotny a jego przebiegiem steruje parametr *selSignificantMethod*.

Sposób wyboru istotnych reguł może być jeden z następujących:

1. Z całego zbioru dostępnych reguł usuwane są wszystkie, których negatywne pokrycie jest większe od zera. Wszystkie pozostałe reguły zostają w finalnym zestawie (*select where ncov!=0*). Podejście to jest bardzo rygorystyczne lecz przydatne w celach opisowych jeśli bez względu na wielkość *pcov* zależy nam na wyborze jedynie silnych reguł.

2. Z całego zbioru dostępnych reguł usuwane są wszystkie, których pozytywne pokrycie jest mniejsze niż negatywne. Wszystkie pozostałe reguły zostają w finalnym zestawie (*select where pcov>ncov*). Podejście to jest dość naturalne, ponieważ wybieramy wszystkie reguły, które są bardziej charakterystyczne dla rozpatrywanej klasy niż poza nią. Z drugiej jednak strony kryterium to jest spełniane przez wiele reguł, które z punktu widzenia opisu i klasyfikacji są nieistotne (np. pokrywają w dość podobnym stopniu zbiór pozytywny i negatywny).
3. Z całego zbioru dostępnych reguł usuwane są wszystkie, których jakość Q jest mniejsza niż zdefiniowana przez parametr $qMin$. Wszystkie pozostałe reguły zostają w finalnym zestawie (*select where ncov!=0*). Wybór sposobu liczenia Q definiuje parametr $qMethodFinal$.
4. Z całego zbioru dostępnych reguł wybiera się zdefiniowaną przez parametr $finalBeam$ liczbę najlepszych reguł. Kryterium jakości Q służy do oceny kompleksów, a typ definiuje parametr $qMethodFinal$. To podejście pozwala na wybór ściśle określonej liczby najlepszych reguł według Q .
5. Z całego zbioru dostępnych reguł wybiera się zdefiniowaną przez parametr $finalBeam$ liczbę najlepszych reguł z uwzględnieniem ich łącznych możliwości predykcyjnych. Podobnie jak poprzednio miarę jakości definiuje $qMethodFinal$, a liczbę reguł w finalnym zbiorze $finalBeam$. Wszystkie zbudowane kompleksy sortuje się według Q a następnie dodaje pojedynczo do zbioru finalnego poczynając od ocenionego najwyżej. Jeśli pewna miara Q_r dla całego finalnego zestawu nie zmniejsza się to kompleks pozostaje, w przeciwnym przypadku jest usuwany i próbnie dodaje się kolejny z listy o mniejszym Q .

$$Q_r = (S^p - S^n)(1 - S^n) \quad (5.7)$$

Wartości S^p i S^n to sumy wartości miary przynależności przykładu do zestawu reguł S (miara używana w klasyfikacji i omówiona w sekcji 5.4) otrzymane odpowiednio dla podpróby przykładów pozytywnych oraz negatywnych zbioru trenującego. Bezwzględna wielkość tej losowej próby definiuje parametr $maxEventsForSelection$.

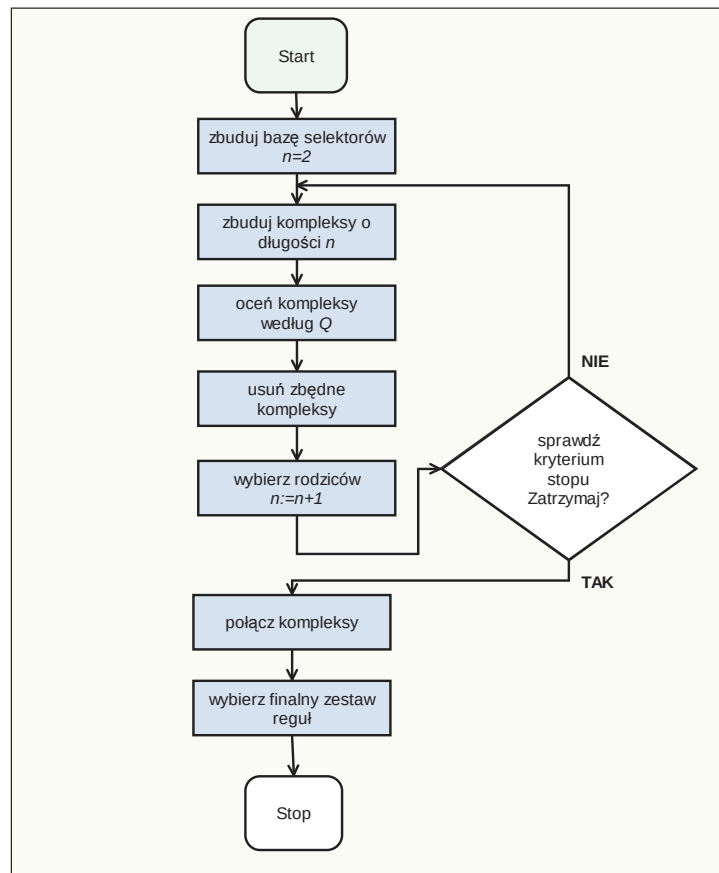
$$S^p = \sum_{e \in D^p} S(e) \quad (5.8)$$

$$S^n = \sum_{e \in D^n} S(e) \quad (5.9)$$

Ta metoda selekcji finalnych reguł skupia się na maksymalizacji sumy wartości $score$ S^p dla przykładów pozytywnych i minimalizacji S^n dla zbioru negatywnych co w konsekwencji pozwala na optymalizację finalnego zestawu reguł pod kątem najlepszej reklasyfikacji podpróby zbioru trenującego.

Warto podkreślić, że liczba i jakość wybranych reguł do finalnego zbioru silnie wpływa na szybkość i jakość klasyfikacji. Stąd ich liczba powinna być możliwie mała i powinny one pokrywać wyłącznie przykłady rozpatrywanej klasy w całej dziedzinie.

Na rysunku 5.1 znajduje się ogólny schemat blokowy algorytmu ADX. Schemat przedstawia kolejne fazy budowy zestawu reguł.



Rysunek 5.1. Schemat blokowy algorytmu ADX

5.3. Miary jakości Q dla kompleksów

Miary jakości dla kompleksu czy selektora opierają się na wyliczonych pokryciach pozytywnym i negatywnym. Pokrycia te są szacowane na podstawie danych, za każdym razem kiedy powstaje nowy selektor bądź kompleks. W pracy proponuje się użycie następujących miar dla jakości Q opartych na $pcov$ i $ncov$:

1.
$$Q_1 = pcov - ncov \quad (5.10)$$

2.
$$Q_2 = (pcov - ncov)(1 - ncov) \quad (5.11)$$

3.
$$Q_3 = (pcov - ncov)(1 - ncov)^2 \quad (5.12)$$

4.
$$Q_4 = pcov \quad (5.13)$$

5.
$$Q_5 = 1 - ncov \quad (5.14)$$

6.
$$Q_6 = pcov(1 - ncov) \quad (5.15)$$

Dla wszystkich zaproponowanych miar maksymalne Q daje najlepszy kompleks w odniesieniu do zastosowanego kryterium.

Miara Q musi spełniać jednocześnie następujące kryteria:

- powinna promować kompleksy o wysokim $pcov$,
- powinna karać kompleksy o wysokim $ncov$,
- powinna być znormalizowana.

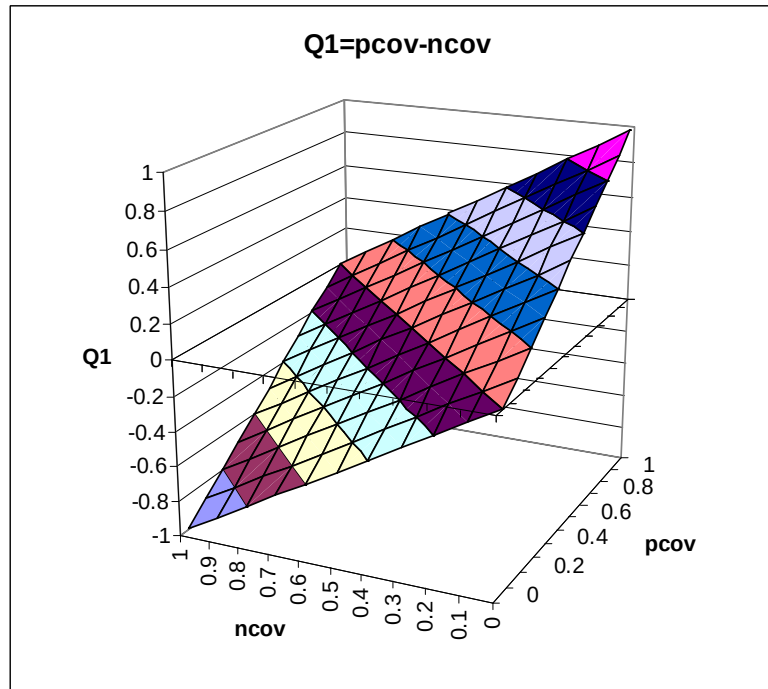
Najprostsza miara Q_1 oparta na różnicy $ncov$ od $pcov$ w zasadzie spełnia wszystkie kryteria. Oba argumenty przyjmują wartości z zakresu $[0,1]$ toteż Q_1 jest z przedziału $[-1,1]$. Niestety promowanie niskiego $ncov$ jest w tym wypadku niewystarczające.

Na przykład:

- c_1 ma $pcov_1 = 1$ i $ncov_1 = 0.5$ wtedy $Q_1(c_1) = 1 - 0.5 = 0.5$
- c_2 ma $pcov_2 = 0.5$ i $ncov_2 = 0$ wtedy $Q_1(c_2) = 0.5 - 0 = 0.5$

Patrząc tylko na same pokrycia widać, iż drugi kompleks opisuje przykłady pozytywne znacznie lepiej. Miara Q_1 mimo spełniania podstawowych kryteriów, nie różnicuje obu przypadków. Należy zatem dodatkowo w Q promować niskie $ncov$ tak by kompleks c_2 miał wyższą ocenę jakości. Ciekawymi alternatywnymi propozycjami są tutaj kolejne miary Q_2 i Q_3 . Obie propozycje spełniają założone kryteria i skupiają się dodatkowo na minimalizacji $ncov$. W Q_3 wartość czynnika kary dla wysokiego $ncov$ jest dodatkowo wzmocniona poprzez pierwiastek wyrażenia $1 - ncov$. Miara oceny kompleksów Q jest używana w kilku miejscach w algorytmie ADX i zależnie od etapu należy wzmacniać karę za wysokie $ncov$.

Różnice pomiędzy wartościami poszczególnych miar Q_1 , Q_2 i Q_3 najlepiej obrazuje przykład:



Rysunek 5.2. Przestrzeń wartości przyjmowanych przez miarę Q_1 w zależności od $pcov$ i $ncov$

- c_1 ma $pcov_1 = 1$ i $ncov_1 = 0.5$ wtedy $Q_2(c_1) = (1 - 0.5) * (1 - 0.5) = 0.25$
- c_2 ma $pcov_2 = 0.5$ i $ncov_2 = 0$ wtedy $Q_2(c_2) = (0.5 - 0) * (1 - 0) = 0.5$
- c_1 ma $pcov_1 = 1$ i $ncov_1 = 0.5$ wtedy $Q_3(c_1) = (1 - 0.5) * (1 - 0.5)^2 = 0.125$
- c_2 ma $pcov_2 = 0.5$ i $ncov_2 = 0$ wtedy $Q_3(c_2) = (0.5 - 0) * (1 - 0)^2 = 0.5$

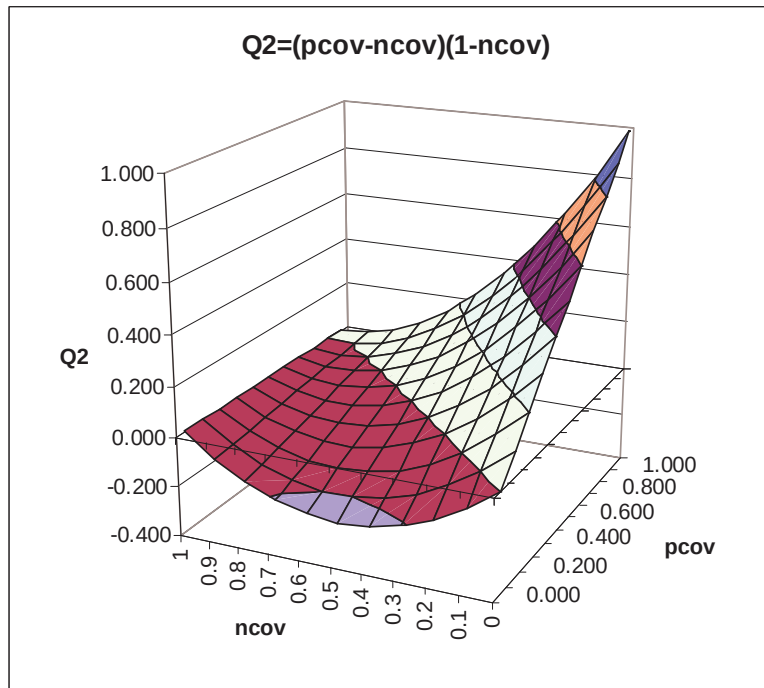
Po serii eksperymentów z zaproponowanymi miarami wynika, iż miara Q_3 daje lepsze rezultaty na etapie finalnej selekcji. W tym etapie zależy nam już na selekcji najsilniej różnicujących reguł, a miara Q_3 sprzyjająca eksploatacji dobrze się do tego nadaje. Miara Q_2 natomiast wydaje się znacznie bardziej odpowiednia do oceny i wyboru rodziców w trakcie iteracyjnego budowania kolejnych kandydatów, ponieważ bardziej sprzyja eksploracji. Na etapie budowy kandydatów bardziej zależy nam na wysokim $pcov$ niż bardzo niskim $ncov$, ponieważ poprzez dodawanie kolejnych odpowiednich selektorów do kompleksu $ncov$ będzie się stopniowo zmniejszać, a gdy osiągnie wartość zero dalsza poprawa jest niemożliwa.

Miary Q_4 i Q_5 zostały przedstawione i zaimplementowane jedynie w celach porównawczych. Obie są właściwie najprostszą propozycją jakości oceny kompleksu i odpowiednio opierają się na maksymalizacji $pcov$ oraz minimalizacji $ncov$. Miara Q_6 łączy w sobie własności Q_4 i Q_5 lecz podobnie jak Q_1 nie promuje dodatkowo niskiego $ncov$.

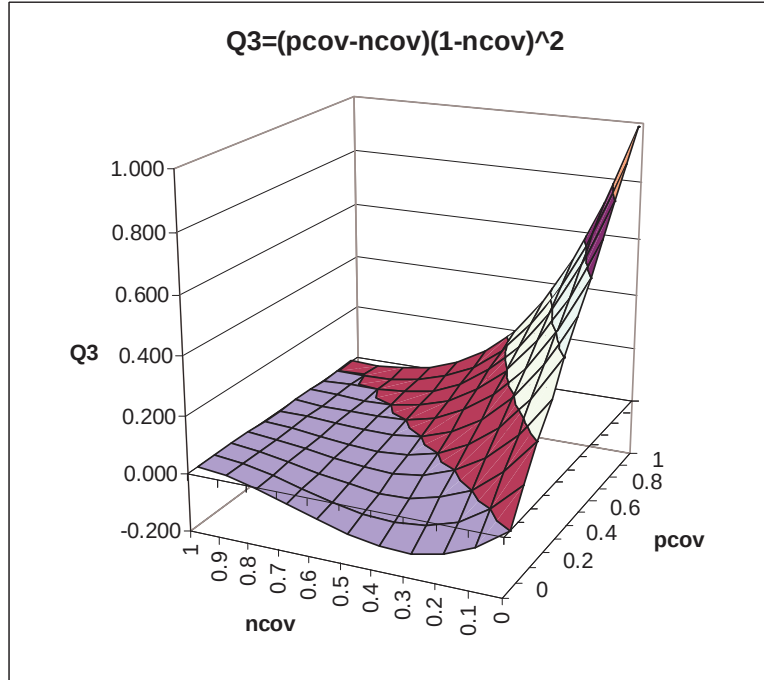
Dla zaprezentowanych już przykładów można policzyć Q_6 :

- c_1 ma $pcov_1 = 1$ i $ncov_1 = 0.5$ wtedy $Q_6(c_1) = 1 * (1 - 0.5) = 0.5$
- c_2 ma $pcov_2 = 0.5$ i $ncov_2 = 0$ wtedy $Q_6(c_2) = 0.5 * (1 - 0) = 0.5$

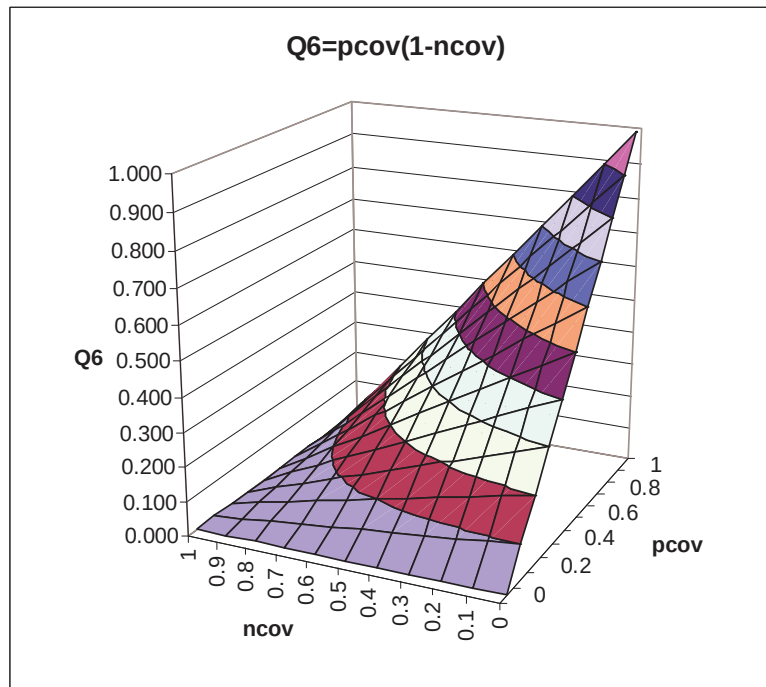
Rysunki [5.2,5.3,5.4,5.5] dodatkowo prezentują trójwymiarowe wykresy omówionych miar w zależności od możliwych wartości $pcov$ oraz $ncov$.



Rysunek 5.3. Przestrzeń wartości przyjmowanych przez miarę Q_2 w zależności od $pcov$ i $ncov$



Rysunek 5.4. Przestrzeń wartości przyjmowanych przez miarę Q_3 w zależności od $pcov$ i $ncov$



Rysunek 5.5. Przestrzeń wartości przyjmowanych przez miarę Q_6 w zależności od $pcov$ i $ncov$

5.4. Proces klasyfikacji

Niewątpliwym atutem systemów regułowych jest duża czytelność i użyteczność gromadzonej przez nie wiedzy. Oczywiście opis w postaci spójnego osobnego zestawu reguł charakteryzujących każdą klasę jest tym bardziej cenny, im lepiej na jego podstawie jesteśmy w stanie później wnioskować. Przydatny opis (tutaj zestaw reguł), który generalizuje wiedzę musi zostać pozytywnie zweryfikowany pod względem jakości klasyfikacji nowych przypadków. Im lepsza jakość predykcji takich nieznanych obiektów, tym można być bardziej pewnym, że w bardziej dokładny sposób udało się opisać różnice występujące pomiędzy klasami. Algorytm ADX stara się zbudować takie reguły które są charakterystyczne dla klasy rozpatrywanej i w jak najmniejszym stopniu opisują przykłady spoza niej.

Klasyfikacja nowego przykładu polega na określeniu podzbioru reguł pokrywających przykład z każdego zestawu, a następnie wyznaczeniu pewnej miary przynależności (*score*) osobno dla każdej z klas. Podczas procesu klasyfikacji kluczową rolę odgrywają wartości odpowiednich wskaźników poszczególnych reguł ($pcov$, $ncov$, $prob$ - oszacowane prawdopodobieństwo a posteriori wystąpienia klasy pozytywnej pod warunkiem pokrywania), wyznaczonych wcześniej na podstawie zbioru trenującego i zapisane. Wartości te pozwalają na wyliczenie *score* dla każdej z klas i nadanie obiektowi etykiety dla której uzyskano maksymalną wartość *score*.

W niniejszej pracy dla ustalonego zbioru reguł r , należących do jednego zestawu R i pokrywających przykład, proponuje się kilka następujących metod szacowania przynależności do klasy:

1.

$$S_0 = \frac{\frac{p-n}{|r|}}{\frac{P-N}{|R|}} \quad (5.16)$$

2.

$$S_1 = \frac{p-n}{P-N} \quad (5.17)$$

3.

$$S_2 = \frac{p}{P} * \frac{N}{n} \quad (5.18)$$

4.

$$S_3 = \frac{p}{P} * [1 - \frac{n}{N}] \quad (5.19)$$

5.

$$S_4 = 1 - \prod_r (1 - prob(r)) \quad (5.20)$$

6.

$$S_5 = \frac{\sum_r prob(r)}{|r|} \quad (5.21)$$

7.

$$S_6 = \begin{cases} S_5 \\ \frac{p}{P} \end{cases} \text{ jeżeli } \bigwedge_c S_5^c = 1$$

8.

$$S_7 = \begin{cases} S_4 \\ \frac{p}{P} \end{cases} \text{ jeżeli } \bigwedge_c S_4^c = 1$$

gdzie we wzorach przyjęto następujące oznaczenia:

- p — oznacza sumę $pcov$ reguł pokrywających rozważany przykład,
- n — oznacza sumę $ncov$ reguł pokrywających rozważany przykład,
- P — oznacza sumę $pcov$ wszystkich reguł dla danej klasy,
- N — oznacza sumę $ncov$ wszystkich reguł dla danej klasy,
- R — oznacza zbiór reguł dla danej klasy (zestaw),
- r — oznacza zbiór reguł pokrywających rozważany przykład (podzbiór zestawu $r \in R$),
- $prob$ — oznacza prawdopodobieństwo wystąpienia klasy pod warunkiem pokrywania przypadku przez regułę,

Przedstawione miary w zasadzie tak kumulują odpowiednie wskaźniki pokryć, by wysoką wartość przynależności przyjmował przykład, który jest pokrywany przez możliwie dużo dobrych reguł. Wysoka jakość jest wciąż rozumiana jako wysokie $pcov$ oraz możliwie niskie $ncov$. Prawdopodobieństwo *apriori* - $prob$, również występujące w miarach jest wyznaczane na podstawie liczby przykładów pozytywnych (trenujących) w stosunku do wszystkich pokrywanych przez regułę. Pozwala ono określić raczej "czystość" danej reguły niż jej "powszechność" określaną przy pomocy $pcov$ i $ncov$. Wartość maksymalną prawdopodobieństwa 1 można dość łatwo uzyskać jeśli reguła pokrywa zaledwie jeden przykład pozytywny. Dwie ostatnie miary S_6 i S_7 łączą oba aspekty – w przypadku gdy wyliczona wartość $score$ dla przykładu jest równa dla kilku klas, przynależność wyznacza się na podstawie odpowiednich $pcov$.

Takie rozwiązanie pomaga w sytuacji pokrywania klasyfikowanego przykładu jedynie przez idealnie "czyste" reguły z wszystkich klas, wtedy $score=1$, a jego wartość należy wyznaczyć w odmienny sposób. Dla większości badanych zbiorów miara S_6 daje najlepsze wyniki klasyfikacji i jest ona domyślnie używana w algorytmie.

Wybór sposobu liczenia $score$ zależy jednak w dużej mierze od problemu i jest to istotny parametr do zbadania podczas procesu tworzenia klasyfikatora. Mimo iż istnieje pewien uniwersalny dobry zestaw parametrów ADX, to dla każdego zadania należy osobno dobrać optymalny ich zestaw (co jest zresztą zgodne z doświadczeniem autora w budowie rozmaitych klasyfikatorów działających na danych komercyjnych). W sekcji 7.1.1 przeprowadzono porównanie jakości klasyfikacji przy użyciu przedstawionych miar używanych do liczenia wartości $score$.

Obok parametru *scoreMethod* (odpowiedzialnego za sposób liczenia przynależności) w procesie klasyfikacji można dodatkowo posłużyć się kilkoma innymi parametrami. Parametr *useSensitivity* pozwala na sterowanie wykorzystania tablicy czułości zdefiniowanych poprzez *sensitivityArray* dla poszczególnych klas, a każda wartość czułości jest z zakresu 0–1. Możliwość taka jest nieoceniona jeśli z jakiś względów jedna klasa silnie dominuje w predykcji i chcemy dla niej osłabiać S poprzez mnożenie przez odpowiedni współczynnik ze zdefiniowanej tablicy czułości. Dodatkowo, istnieje możliwość automatycznego ustawienia odpowiednich współczynników w tablicy czułości na podstawie prawdopodobieństwa występowania poszczególnych klas (parametr *useSensitivity* = 2). Przy ustawieniu *useSensitivity* = 0 czułość dla każdej z klas wynosi 1, czyli nie zmienia ona wartości $score$.

W przypadku gdy dla klasyfikowanego przykładu dla kilku klas uzyskano jednakową wartość $score$ to nadanie etykiety jest realizowane losowo z prawdopodobieństwem dla klasy równym odpowiedniej czułości. W przypadku ustawienia *useSensitivity* = 0 etykieta nadawana jest losowo na podstawie prawdopodobieństwa *a priori* (dokładnie tak samo jak w przypadku *useSensitivity* = 2). W przypadku gdy żadna reguła nie pokrywa przykładu to pozostaje on nieoznaczony i w implementacji ADX traktuje się go jako błąd klasyfikacji. Niemniej jednak taka sytuacja w praktyce występuje bardzo rzadko.

5.5. Obsługa wartości nieznanych

Algorytm oraz jego implementacja są przygotowane do obsługi danych zawierających wartości puste. Wszystkie potrzebne statystyki (pokrycia) wylicza się na podstawie przykładów, które dla rozważanych atrybutów zawierają stosowne wartości.

Dla przykładu z tabeli 5.7 dla klasy *no* mamy pokrycie pozytywne $pcov(a_1 = \textit{sunny}) = \frac{2}{3}$ obliczone na podstawie wyróżnionych przykładów w tabeli. Przykład, który posiada etykietę *no* i zawiera znak '?' (wartość nieznana) nie bierze udziału w obliczaniu wspomnianego pokrycia.

a_1	a_2	d
sunny	85	no
rainy	80	no
?	70	yes
rainy	68	yes
?	65	no
sunny	72	no
rainy	75	yes

Tablica 5.7. Przykład tabeli z brakami w danych

Obsługa nieznanych wartości w procesie klasyfikacji przebiega w zasadzie w identyczny sposób. Oznacza to, iż dla przykładu, który dla niektórych atrybutów przyjmuje wartości puste wnioskujemy jedynie na podstawie reguł rzeczywiście go pokrywających, a pokrycie przez pojedynczą regułę określa się na podstawie znanych wartości. Jak wspomniano wcześniej algorytm ADX tworzy pewien nadmiarowy zestaw reguł (jeden przykład może być pokrywany przez kilka finalnych reguł), zwiększa to szanse na wnioskowanie w oparciu o reguły pokrywające obiekt o atrybutach z niepustymi wartościami.

5.6. Obsługa sprzeczności w danych

Przez pojęcie sprzeczności w danych rozumie się identyczne przykłady w zbiorze trenującym, którym nadano różne etykiety klasy. Sprzeczne dane w algorytmie ADX nie muszą być obsługiwane w żaden specjalny sposób. Skoro algorytm stara się znaleźć możliwie długie i unikalne kompleksy dla rozpatrywanej klasy to występowanie kilku identycznych przykładów o różnych etykietach nie jest problemem. Załóżmy, że odkryto kompleks w danych, który pokrywa tylko obiekty sprzeczne w danych – zarówno w zbiorze pozytywnym jak i negatywnym. Odpowiednie pokrycia ($pcov$ i $ncov$) na podstawie których jest liczona miara Q dla tego kompleksu w sposób jednoznaczny definiują jego przydatność. Jeśli udział obiektów sprzecznych w zbiorze pozytywnym i negatywnym jest taki sam to rozważany kompleks będzie miał $pcov = ncov$ i tym samym będzie uznany za mało interesujący. Jeśli natomiast udział takich sprzecznych przykładów jest znacznie większy w jednej z klas to odpowiednia reguła zostanie odkryta, lecz będzie miała ona niezerowe $ncov$. Niezerowe $ncov$ dla kompleksu nie jest sytuacją niezwykłą w realnych danych, w szczególności dla reguł pokrywających przykłady leżące w hiperprzestrzeni na granicy klas. Przykład testowy tożsamy z tymi o sprzecznych etykietach będzie zatem zaklasyfikowany do klasy gdzie dominowały one w danych trenujących na podstawie reguły o wysokim $pcov$ i niezerowym $ncov$.

5.7. Obsługa atrybutów numerycznych

W implementacji posłużono się kilkoma metodami wstępnej dyskretyzacji danych. Dyskretyzacja jest niezależna od algorytmu konstrukcji klasyfikatora i następuje automatycznie dla ustawienia *discretizeData = true*. Obok prostych nienadzorowanych metod dyskretyzacji opierających się na równych przedziałach i równych częstotliwościach istnieje możliwość włączenia algorytmu ChiMerge, dyskretyzacji opartej na zmianie wartości decyzji lub metody zaproponowanej w pracy [Fayyad *et al.*, 1992]. O ile algorytm ChiMerge jest powszechnie znany ([Cichosz, 2000],[Kerber, 1992]) to pozostałe dwie metody wymagają krótkiego objaśnienia.

Założmy, że będzie dyskretyzowany atrybut a_1 , a zmienną decyzyjną opisuje atrybut d . W pierwszym kroku dane sortuje się względem atrybutu dyskretyzowanego (tutaj a_1). Każda zmiana wartości decyzji (począwszy od np. wartości minimalnej) wyznacza kolejny przedział. Ten prosty, szybki (głównym nakładem jest sortowanie) i dość skuteczny algorytm dyskretyzacji zaprezentowano w tabeli 5.8. Ostatnia kolumna w tabeli prezentuje wartość atrybutu a_1 po dyskretyzacji (*cut point*), średnią arytmetyczną atrybutu na styku klas.

a_1	d	$disc(a_1)$
0.64	yes	0.705
0.65	yes	0.705
0.68	yes	0.705
0.69	yes	0.705
0.7	yes	0.705
0.71	no	0.73
0.72	no	0.73
0.74	yes	0.78
0.75	yes	0.78
0.81	no	0.86
0.86	no	0.86

Tablica 5.8. Przykład dyskretyzacji na podstawie zmiany wartości decyzji

Zbiór takich punktów cięć w przypadku dyskretyzacji opartej na zmianie wartości wyznacza przedziały dyskretyzacji. W przypadku algorytmu z [Fayyad *et al.*, 1992] jest on zbiorem kandydatów poddanych kolejnym ocenom. Następnie wybiera się najlepszy punkt cięcia dla zadanego przedziału (posiłkując się jedną z miar jakości podziału np. *information gain*), a następnie sprawdza czy spełnia on odpowiednie kryteria przyrostu informacji. Jeśli punkt spełnia kryterium to rekurencyjnie w identyczny sposób postępuje się z przedziałem lewym i prawym wyznaczonymi przez wybrany punkt. Podział

w punkcie t dla przedziału wartości s uważa się za skuteczny jeśli przyrost informacji $infoGain = Ent(s) - Ent(t, s)$ jest większy niż waga $weight$ (patrz równanie 5.22). Przy tym $C(s)$ oznacza liczbę klas przyjmowanych przez obiekty w przedziale s , $O(s)$ liczbę przykładów w s , s_1 i s_2 podprzedziały s wydzielone przez podział w punkcie t , $cut(s)$ liczbę wszystkich możliwych cięć w s , a $Ent(s)$ to entropia policzona dla zakresu wyznaczonego przez s .

$$weight = \log_2(C(s)^3 - 2) - (C(s)Ent(s) - C(s_1)Ent(s_1) - C(s_2)Ent(s_2)) \quad (5.22)$$

$$delta = \frac{\log_2(cut(s)) + delta}{O(s)} \quad (5.23)$$

Należy jeszcze dodać, iż dyskretyzacji z nadzorem podlega jedynie zbiór trenujący. Oznacza to, iż na zbiorze trenującym znajdowane są najlepsze przedziały dla atrybutów numerycznych dla danej metody, a następnie w procesie klasyfikacji przykładów testujących przedziały te służą w regułach jako zakresy numeryczne. Dyskretyzacja jest zatem wykorzystana do pracy klasyfikatora na przedziałach numerycznych, co jest niezwykle wygodne nie tylko jeśli chodzi o czytelność reguł lecz również o proces klasyfikacji nowych przykładów. Przykładowa bardzo prosta reguła wykorzystująca przedział to np. `'temperature = (-inf; 65.0]'`.

Wymienione (dedykowane dla ADX) metody dyskretyzacji zostały re-implementowane przez autora i mimo iż w dmLab można użyć kilku innych algorytmów to nie są one już przedmiotem badań niniejszej pracy. Domyślnie używa się algorytmu zaproponowanego przez Fayyada i Iraniego.

5.8. Oszacowanie złożoności obliczeniowej algorytmu

Niech A oznacza zbiór atrybutów natomiast $|A|$ ich liczbę. Zapis $|D|$ oznacza liczbę przykładów w zbiorze uczącym, natomiast $|C|$ liczbę odkrytych kompleksów w trakcie całej fazy budowania zestawu reguł. Nakłady poniesione przez algorytm dla zbudowania pojedynczego zestawu reguł są następujące:

1. Znalezienie bazy selektorów. Potrzeba jednokrotnego przeskanowania całej tabeli danych do stworzenia tablic kontyngencji.
2. Budowa zbioru reguł. W tabeli zaprezentowano oszacowanie dla najbardziej pesymistycznie przyjętego wypadku, gdzie $cover(i)$ oznacza funkcję odpowiedzialną za policzenie pokrycia dla pojedynczego kompleksu (liniowo zależna od liczby przykładów $|D|$).
3. Sklejanie kompleksów. Budowa kolejnych kompleksów wymusza ich naturalne uporządkowanie, toteż koszt tej operacji jest liniowy zależny od łącznej liczby odkrytych reguł. Faza ta polega na przeskanowaniu zbioru reguł i ewentualne dodanie nowej w oparciu o następujące po sobie kompleksy.
4. Wybór finalnych reguł. Koszt tej operacji jest liniowo zależny od liczby kompleksów po operacji sklejanie (C_m).

Faza	Opis	Koszt
1	Znalezienie bazy selektorów	$ D * A $
2	Budowa zestawu reguł	$\sum_{i=2}^{max(length(c))} searchBeam^2 * D * cover(i)$
2	Sklejanie kompleksów	$ C $
2	Wybór finalnych reguł	$ C_m $ lub $2 C_m $ (w zależności od metody)

Tablica 5.9. Aproksymacja złożoności obliczeniowej algorytmu

Jak widać czas uczenia algorytmu jest liniowo zależny od liczby przykładów i liczby atrybutów w zbiorze. Najbardziej krytyczna zależność jest związana z parametrem *searchBeam*, którym to steruje użytkownik wpływając na czas i jakość obliczeń. Im większa wartość zdefiniowana przez parametr, tym więcej kandydatów jest generowanych i tym więcej bardziej szczegółowych reguł zostaje odkrytych.

5.9. Cechy algorytmu ADX

W podsumowaniu zebrane zostały główne cechy algorytmu/klasyfikatora ADX. W zależności od zastosowania niektóre z przedstawionych właściwości będą jego zaletami lub wadami.

- Właściwie liniowa zależność czasu uczenia od liczby przykładów oraz liczby atrybutów (patrz sekcja 7.1.2). Czas uczenia dla dużych zbiorów danych jest porównywalny lub niższy niż drzewa decyzyjnego (patrz wyniki w sekcjach 7.1.3 i 7.1.4).
- Algorytm nie jest czuły na sprzeczności w danych.
- Selekcja najlepszych atrybutów jest częścią procesu uczenia. Podczas procesu tworzenia bazy selektorów znajdowane są najbardziej odpowiednie selektory służące w kolejnych krokach do budowy reguł.
- Wiedza w postaci krótkich zwięzłych reguł jest czytelna dla człowieka. Użytkownik może ograniczać maksymalną liczbę finalnych reguł lub może skupić się na analizie tylko najistotniejszych z nich. Reguły można sortować pod względem jakości (miara Q).
- Parametr *searchBeam* ma bardzo duży wpływ na eksplorację i skalowalność algorytmu. Steruje on szerokością poszukiwań, im większa wartość parametru tym więcej reguł może zostać odkrytych kosztem dłuższego czasu poszukiwań. Im mniejsza wartość, tym czas uczenia jest mniejszy a zestaw reguł sprowadza się do najbardziej podstawowych (najważniejszych). Parametr *searchBeam* może zostać ustawiony w zależności od szybkości komputera lub potrzeb użytkownika.
- Jakość reguł wyznaczona na zbiorze trenującym jest istotnym czynnikiem służącym w trakcie procesu klasyfikacji. Nie tylko liczba reguł ale również ich jakość decyduje o nadaniu etykiety nowemu obiektowi.

- Zaprezentowana idea algorytmu zakłada działanie na danych symbolicznych i jako typowy symboliczny klasyfikator potrzebna jest wstępna dyskretyzacja. Jednak w implementacji techniki dyskretyzacji są wbudowane w proces uczenia i testowania klasyfikatora, a wyjściowe reguły (zapisywane do pliku) opierają się na numerycznych przedziałach. Użytkownik definiuje tylko w zbiorze wejściowym, które atrybuty są numeryczne, a które nominalne.

5.10. Oryginalność rozwiązania

Niewątpliwie oryginalnym rozwiązaniem w pracy jest sposób wykorzystania schematu budowania reguł asocjacyjnych (znanego z algorytmu Apriori [Agrawal *et al.*, 1994]) do indukcji reguł klasyfikacyjnych. Propozycja użycia schematu tego algorytmu pojawiła się już w publikacji [Stefanowski, 2001] (DOMApriori) lecz w ADX wprowadzono dodatkowo pewne modyfikacje zwiększające efektywność rozwiązania.

W celu tworzenia użytecznych reguł klasyfikacyjnych (dostatecznie długich i dokładnych), zaproponowano specyficzne miary oceny reguł wykorzystywane w trakcie ich budowy. W trakcie iteracyjnego procesu rozbudowy kompleksów istotnym elementem jest pokrywanie dostatecznej liczby przykładów, rokujące dalsze korzystne wydłużanie kompleksów. Czystość reguł na etapie budowy jest traktowana trochę mniej kategorycznie, ponieważ to zadanie spełnia odpowiednie wydłużanie kompleksów. Zapewnienie możliwości swobodnego wydłużania polega na doborze odpowiednich kompleksów z dużym wsparciem. W trakcie rozrostu bazy kompleksów priorytetem jest budowa sporej liczby dobrych reguł, lecz dopiero w fazie finalnej selekcji wybiera się te najlepsze z punktu widzenia klasyfikacji. Poprzez umiejętne związanie $pcov$ i $ncov$ w miarach oceny jakości Q reguł udało się uzyskać zamierzony efekt jednoczesnego pomiaru powszechności i czystości reguły. Powszechność w tym wypadku pozwala na wydłużanie kompleksów – strategia polega na dołączaniu kolejnych warunków do reguły by zmniejszać $ncov$ przy utrzymaniu wysokiego $pcov$. Czystość reguły wpływa natomiast na skuteczną klasyfikację.

Dругa istotna modyfikacja schematu wykorzystanego w ADX dotyczy wiązki ograniczającej liczbę kompleksów używanych do budowy kolejnych kandydatów. Zamiast zdefiniowanych w oryginalnym rozwiązaniu (również w DOMApriori) progów miar jakości (jak to jest w Apriori, gdzie definiuje się minimalne wsparcie i ufność) wprowadzono stałą liczbę najwyżej ocenionych kompleksów. Ten manewr spowodował niezwykle skok efektywności rozwiązania, ponieważ w każdej fazie ograniczamy z góry liczbę rodziców. W Apriori z góry nie wiadomo jakie przyjąć wartości wsparcia jako próg odcięcia, by otrzymać wynik w rozsądnym czasie. Eksperymenty potwierdzają, że w ADX zależność czasu indukcji rodziny reguł od liczby przykładów i atrybutów jest przeważnie liniowa dla badanych zbiorów (7.18, 7.19).

W ADX również nie wykorzystuje się popularnego schematu sekwencyjnego pokrywania, a rozrost reguł przeprowadza się dość swobodnie w kierunku lepszego opisu klasy pozytywnej eksplorując przy tym wiele potencjalnie dobrych hipotez. W fazie klasyfikacji, poprzez zaproponowany mechanizm wa-

zonego głosowania reguł (wag opartych na miarach jakości dopasowania do zbioru uczącego), uzyskuje się nieliniowe kształty granic podziałów pomiędzy klasami.

Zaproponowane silnie heurystyczne podejście, potwierdza swoją wysoką skuteczność i efektywność w eksperymentach porównawczych przedstawionych w rozdziale 7 (7.2, 7.5).

6. Selekcja istotnych atrybutów z wysokowymiarowych danych

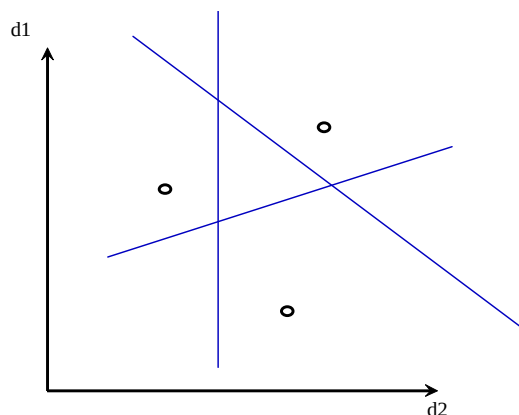
6.1. Opis problemu

Preselekcja informatywnych cech wykorzystanych do późniejszego procesu uczenia z nadzorem jest niezwykle istotnym i jednocześnie bardzo delikatnym zadaniem. W dalszej części pracy przedstawiony jest prosty koncepcyjnie lecz wymagający obliczeniowo sposób rozwiązania problemu selekcji atrybutów. Rzetelność prezentowanego podejścia leży w wielokrotnym konstruowaniu klasyfikatora (np. drzewa decyzyjnego lub klasyfikatora regulowego) dla wielu zbiorów trenujących losowo wybieranych ze źródłowych danych, gdzie poszczególne próbki zawierają jedynie podzbiór spośród wszystkich obserwowanych cech. Ranking atrybutów otrzymany finalnie z proponowanego algorytmu, może zostać z powodzeniem użyty do podniesienia jakości klasyfikacji przeprowadzonej dowolnym algorytmem. Prezentowany w dalszej części pracy algorytm intensywnie weryfikowano na danych [Golub *et al.*, 1999] (*leukemia problem*) oraz [Alizadeh *et al.*, 2000] (*lymphoma problem*), czego wynikiem jest spora część rozdziału 7.

Podstawowym wyzwaniem w trakcie analiz danych biologicznych jest ich specyficzny rozmiar: niewielka liczba przykładów (próbek) w stosunku do liczby obserwowanych czynników. Dla rozważanych danych opisujących ekspresję genów dostępnych jest przeważnie kilkadziesiąt rekordów (pacjenci) i kilka tysięcy atrybutów (poszczególne ekspresje obserwowanych genów). Celem większości analiz jest znalezienie wszystkich genów związanych (odpowiedzialnych) z występowaniem obserwowanego czynnika (np. typ choroby). Dodatkowo zadanie wyboru genów jest utrudnione poprzez szum występujący w danych, ponieważ właściwie każdy rekord uzyskuje się w osobnym procesie, wrażliwym na różne czynniki zewnętrzne (dwa kolejne procesy uzyskania obrazów ekspresji genów od pacjenta mogą dać różne od siebie wyniki).

We wspomnianej sytuacji, zadanie nadzorowanego uczenia różni się zasadniczo od typowego problemu z zakresu eksploracji danych, gdzie dysponuje się wystarczającą liczbą przykładów dla każdej z klas. Przeważnie celem takiego uczenia jest budowa klasyfikatora o możliwie najlepszych właściwościach predykcyjnych dla nieznanego mu danych. Selekcja atrybutów w powszechnym ujęciu jest raczej środkiem do osiągnięcia postawionego celu – stworzenia skutecznego klasyfikatora. W proponowanej metodzie dla wspomnianych danych jest raczej odwrotnie: budowa poszczególnych klasyfikatorów służy do wyboru informatywnych cech – genów. Informacja o zestawie potencjalnie zależnych atrybutów z obserwowanym zjawiskiem (np. typem białaczki) może

być z powodzeniem weryfikowana w dalszym procesie przez genetyków. Użycie różnych metod numerycznych pozwala na uniknięcie szukania przez nich na "ślepo" w ogromnej masie wszystkich obserwowanych genów i pozwala skoncentrować badania genetyków na rzetelnie wydzielonym podzbiorze. Dla specyficznych danych mikromacierzowych, każdy sensowny algorytm klasyfikacyjny może w zasadzie dostarczać bardzo podobnych wyników (mierzonych np. wskaźnikiem błędu) [Dudoit *et al.*, 2003]. Nie ma w tym właściwie nic niezwykłego, ponieważ z punktu widzenia klasyfikacji problem nie jest skomplikowany, a jedynie źle zdefiniowany.



Rysunek 6.1. Wizualizacja wymiaru Vapnika-Chervonenkisa dla przestrzeni dwuwymiarowej

Przykładowo bardzo prosto można zauważyć, iż $d + 1$ obserwacje w przestrzeni R^d (d wymiarowej) można dowolnie podzielić na dwa podzbiory (klasy) przy użyciu tylko jednej hiperpłaszczyzny. Formalnie mówiąc wymiar Vapnika-Chervonenkisa dla rodziny hiperpłaszczyzn jest $d + 1$. Rysunek 6.1 przedstawia wszystkie możliwe proste (sposoby podziału), które dzielą trzy dostępne obiekty w przestrzeni dwuwymiarowej na dwie dowolne klasy. Wynika stąd, że problem klasyfikacji dla dwóch klas jest zadaniem trywialnym jeśli np. dostępnych jest 101 przykładów (rekordów danych) w przestrzeni R^{100} . W przypadku danych pozyskanych z mikromacierzy mowa jest o 40–60 badaniach pacjentów i obserwowanych 6000–7000 różnych genów (czyli 60 obserwacji w przestrzeni R^{7000} !).

W konsekwencji, biorąc pod uwagę wrodzoną zmienność danych i fakt niewielkiej ilości dostępnych obserwacji, klasyfikacja staje się prawdziwym wyzwaniem "tylko" ze względu na otrzymanie klasyfikatora który będzie potrafił korzystać jedynie z informatywnych atrybutów. Wzmacnia to potrzebę związania budowy klasyfikatora z procesem selekcji cech.

W pracy [Dudoit *et al.*, 2003] Dudoit and Fridlyand przekonująco wykazują ogromne znaczenie uwzględnienia procesu selekcji cech w trakcie oceny jakości klasyfikatora. W szczególności jeśli procedura oceny jest oparta na walidacji krzyżowej (*cross-validation*), nie ulega kwestii że etap selekcji informatywnych genów powinien być przeprowadzony wewnątrz procesu walidacji krzyżowej. Bardzo dobrym przykładem skutków niepoprawnie przeprowadzonej oceny jakości wyników, może być eksperyment przeprowadzony przez

Simona opisany w pracy [Simon *et al.*, 2003]. Przygotowano tam 2000 zbiorów danych, każdy zawierający 20 różnych profili ekspresji (przykładów) dla 6000 genów losowo wygenerowanych z takim samym gaussowskim rozkładem. Profile zostały losowo podzielone na dwie równe klasy, toteż w danych nie wystąpiła żadna prawdziwa zależność pomiędzy klasą a obserwacjami. Rozważono następnie 3 różne eksperymenty, gdzie w dwóch z nich użyto walidacji krzyżowej (typu *leave-one-out*). W jednym usuwano pojedynczą obserwację po selekcji cech, a w drugim przed wyborem informatywnych genów.

1. Budowa klasyfikatora na pełnych danych — dla 98.2% symulowanych zbiorów danych uzyskano zerowy błąd reklasyfikacji (w tym eksperymentcie zbiór testowy i trenujący były tożsame).
2. Gdy sama klasyfikacja jest poddana walidacji krzyżowej wciąż 90.2% symulowanych zbiorów miało zerowy błąd klasyfikacji. Na pełnej próbie danych dokonano selekcji cech a następnie przeprowadzono proces walidacji krzyżowej.
3. Gdy selekcja genów jest również poddana walidacji krzyżowej mediana liczby źle zaklasyfikowanych profili/przypadków wynosiła 11 (z przedziału 0-20). W tym przypadku odrzucano jeden przykład po czym dokonywano selekcji cech i uczenia klasyfikatora a następnie badano jakość klasyfikacji na odrzuconym wcześniej przykładzie.

Wynik trzeciego eksperymentu sugeruje, iż tylko tam zastosowano właściwą metodologię, ponieważ dla czysto losowych danych nie ma mowy o prawidłowej predykcji, która to udała się w dwóch poprzednich eksperymentach tylko dzięki błędnie przeprowadzonemu procesowi. Podstawowym błędem w metodologii (wskazywanym w [Dudoit *et al.*, 2003] i [Simon *et al.*, 2003]) jest lekceważenie faktu, iż proces selekcji cech jest integralną częścią uczenia i nie może być wykonywany przy użyciu wszystkich dostępnych danych. Walidacja krzyżowa staje się wtedy elementem uczenia (ocenia dopasowanie wybranych wcześniej cech do całego zbioru) zamiast oceniać zdolności predykcyjne klasyfikatora na nieznanym danych.

Większość dotychczasowych prac skupia się jednak na wyborze genów, które są silnie zróżnicowane pod względem swojej ekspresji (nadekspresywne i podekspresywne) co nie jest tożsame z zadaniem selekcji "relatywnie istotnych" cech służących do późniejszej udanej predykcji. Ważne jest by dysponować jednak jakąś obiektywną miarą relatywnej istotności genów dla danego zadania klasyfikacji. Należy zwrócić tutaj dodatkowo uwagę także na to, że wybór istotnych cech o zdolnościach predykcyjnych znacznie się różni od ogólnie rozumianego zadania wyznaczania cech o silnie zróżnicowanej ekspresji, a w szczególności ekspresji genów (temat różnych analiz statystycznych dla zagadnień danych mikromacierzowych jest szeroko omawiany w [Speed, 2003] i [Smyth *et al.*, 2003]). W takim ogólnym zdefiniowaniu zadania interesujące geny (ponadprzeciętnie ekspresywne) są wybierane na podstawie zastosowania metod statystycznych – np. analizy średnich i przy uwzględnieniu statystycznej istotności otrzymanego rezultatu. Ten kierunek badań ekspresji genów ani nie uwzględnia klasyfikacji ani możliwych interakcji między genami, co, jak wykazały eksperymenty porównawcze przeprowa-

dzione w pracy oraz weryfikacja genetyków, prowadzi do nieco odmiennych wyników.

W bieżącym rozdziale jest przedstawiona propozycja procedury selekcji informatywnych cech, które następnie mogą posłużyć do budowy dowolnego klasyfikatora. Ścisłej mówiąc, algorytm na wyjściu dostarcza ranking obiektywnie ocenionych wszystkich obserwowanych cech wraz z informacją o wiarygodności wyników.

6.2. Opis badanych wysokowymiarowych zbiorów

Dane, które zostały użyte do przebadania właściwości algorytmu MCFS to zbiory analizowane w pracy [Golub *et al.*, 1999] oraz [Alizadeh *et al.*, 2000]. Zbiór Goluba zawiera 38 przykładów, które należą do dwóch klas – dwóch typów białaczki:

- białaczka limfoblastyczna (lymphoblastic leukemia – ALL); 27 przypadków,
- ostra białaczka szpikowa (acute myeloid leukemia – AML); 11 przypadków.

Każdy przypadek w zbiorze Goluba to zapis poziomów ekspresji 7129 genów dla pojedynczego pacjenta.

Drugi z wymienionych zbiorów [Alizadeh *et al.*, 2000] zawiera pomiary dla 4026 genów zebrane od 62 pacjentów. Pacjenci zostali w nim sklasyfikowani do trzech następujących klas:

- chłoniak oraz białaczka (lymphoma and leukemia – DLCL); 42 przykłady,
- chłoniak grudkowy (follicular lymphoma – FL); 9 przykładów,
- przewlekła limfocytarna białaczka (chronic lymphocytic leukemia – CLL); 11 przykładów.

6.3. Algorytm MCFS

6.3.1. Miary jakości klasyfikacji

Z procesem budowy dowolnego klasyfikatora nieodłącznie związana jest pewna miara jego oceny. Oceny tej przeważnie dokonuje się na dwóch zbiorach danych: danych trenujących (sprawdzając dopasowanie klasyfikatora), oraz na nieznanymi danych testujących (sprawdzając jego możliwości predykcyjne).

Z uwagi na charakter klasyfikatorów omawianych w tej pracy dalsze rozważania będą się ograniczać do miar oceny konstruowanych dla nominalnej zmiennej decyzyjnej. W celu formalnego zdefiniowania omówionych miar należy rozważyć macierz błędów (dla dwuklasowego problemu) zobrazowaną na rysunku 6.1, gdzie przyjęto następujące oznaczenia: T_1 , T_2 są liczbami poprawnie zaklasyfikowanych przykładów z klas odpowiednio 1 i 2, F_{12} to jest liczba przykładów z klasy 1 zaetykietowanych jako klasa 2. Niech $|D|$ oznacza

liczbę wszystkich przykładów czyli $|D| = T_1 + F_{12} + T_2 + F_{21}$, natomiast odpowiednie ilości przykładów obserwowanych i prognozowanych odpowiednio jako $O_1 = T_1 + F_{12}$, $P_1 = T_1 + F_{21}$ etc. .

	Prognozowane		
Obserwowane	T_1	F_{12}	O_1
	F_{21}	T_2	O_2
	P_1	P_2	

Tablica 6.1. Macierz błędów dla problemu o dwóch klasach

Najpopularniejszymi metodami pomiaru jakości klasyfikacji są następujące:

- *Accuracy* (Acc) – stosunek poprawnie zaklasyfikowanych przykładów do wszystkich (często wyrażany w procentach).

$$Acc = \frac{(T_1 + T_2)}{|D|} = 1 - Err \quad (6.1)$$

- Błąd predykcji (Err) – (ang. *Error Rate*) stosunek błędnie zaklasyfikowanych przykładów do wszystkich (często wyrażany w procentach).

$$Err = \frac{(F_{12} + F_{21})}{|D|} = 1 - Acc \quad (6.2)$$

- Korelacja – współczynnik korelacji liczony pomiędzy wektorami oryginalnych oraz predykowanych wartości decyzji (dla zmiennej decyzyjnej numerycznej).
- Błąd średniokwadratowy RMSE – błąd liczony na podstawie pierwiastka średniej wartości kwadratów różnic wartości oryginalnych i predykowanych (dla zmiennej decyzyjnej numerycznej).
- Statystyki nakierowane na jakość w ramach każdej z klas: *True Positive Rate* (TPR), *False Positive Rate* (FPR), *Precision*, *Recall*. Statystyki te oblicza się na podstawie macierzy błędów (*confusion matrix*) osobno dla każdej z klas. Warto zauważyć że TPR wyraża pokrycie pozytywne zbioru przykładów testowych przez klasyfikator natomiast FPR odpowiednio pokrycie negatywne.

$$TPR_1 = Recall_1 = \frac{T_1}{T_1 + F_{12}} \equiv pcov_1 \quad (6.3)$$

$$FPR_1 = \frac{F_{21}}{F_{21} + T_2} \equiv ncov_1 \quad (6.4)$$

$$Precision = \frac{T_1}{T_1 + F_{21}} \quad (6.5)$$

- Statystyka Kappa – miara ta zaproponowana w [Cohen, 1960] jest często używana w medycynie do oceny zgodności różnych klasyfikacji (np. dokonanych przez różnych lekarzy). Miara podobnie jak $wAcc$ jest wrażliwa na błąd dla niewielkiej klasy decyzyjnej.

$$kappa = \frac{Acc - k}{1 - k}, k = \frac{1}{|D|^2}(O_1 P_1 + O_2 P_2) \quad (6.6)$$

- Krzywa ROC (*Receiver Operating Characteristic*) – Krzywa konstruowana dla problemów o dwóch klasach, prezentująca zależność TPR od FPR . Klasyfikatory dyskretne które określają jedynie etykietę klasy mogą być przedstawione na przestrzeni ROC jako pojedyncze punkty. Dla klasyfikatorów których odpowiedzią jest prawdopodobieństwo przynależności do klasy (np. metody Bayesowskie) poprzez wprowadzenie odpowiedniego progu (*threshold*) można sztucznie zawyżać prawdopodobieństwo przynależności do jednej z klas. Przykładowo by obiekt został zaklasyfikowany do klasy A prawdopodobieństwo przynależności do tej klasy musi być wyższe niż analogiczne prawdopodobieństwo związane z klasą B powiększone dodatkowo o wartość ustalonego progu. Zależność funkcyjną pomiędzy TPR i FPR uzyskuje się zatem poprzez odpowiednie manewrowanie progami tak by wyznaczyć wiele punktów w przestrzeni ROC odpowiadającym różnym wartościom progu. Próg przyjmuje wtedy wartości w granicach pomiędzy sytuacją kiedy wszystkie przykłady klasyfikuje się jako pozytywne do momentu gdy wszystkie przykłady zostają uznane za negatywne.

Mimo, iż najwięcej informacji odnośnie dokładności klasyfikacji otrzymuje się na podstawie macierzy błędów i krzywej ROC, to wygodnie jest dysponować jedną liczbą, która w ustalony sposób definiuje jakość klasyfikacji. Oczywiście, sprowadzenie jakości klasyfikacji (szczególnie dla wieloklasowych problemów) do jednej wartości jest dużym uproszczeniem, lecz daje możliwość prostego i bardzo wygodnego porównania dwóch niezależnych systemów (np. najpopularniejsza miara *Accuracy* dla problemów z nominalną wartością decyzji). Zunifikowanie różnych aspektów jakości klasyfikatora w jednej mierze (np. *Accuracy*) jest uproszczeniem procesu optymalizacji parametrów klasyfikatora z wielokryterialnej do jednokryterialnej (jedno kryterium oceny). Należy tutaj jednak zaznaczyć, że w praktycznych zastosowaniach nie zawsze maksymalna wartość *Accuracy* (czy też minimalny błąd) jest najważniejszym kryterium oceny.

Miara *Accuracy* jakkolwiek popularna i uniwersalna ma jedną bardzo istotną wadę. Dla problemów o silnym zróżnicowaniu liczności klas wysokie *Acc* może być przyjmowane kosztem licznych błędów w ramach nielicznej klasy. W skrajnym przypadku gdy np. 90 przykładów należy do klasy 1 natomiast 10 do klasy 2 klasyfikator, który będzie zawsze orzekał klasę 1 będzie miał 90% skuteczności (*Accuracy*). By przeciwdziałać temu groźnemu zjawisku warto brać pod uwagę odpowiednie liczności klas tak by wprowadzić dodatkowy czynnik kary dla błędów w słabo reprezentowanej klasie. Miara oceny jakości klasyfikacji *weighted accuracy wAcc* (wzór 6.7) używana w algorytmie MCFS jest wolna od problemu silnie zróżnicowanych wielkości klas. Miara ta często w literaturze nazywana jest również *balanced accuracy*.

$$wAcc = \frac{1}{2} \left(\frac{T_1}{T_1 + F_{12}} + \frac{T_2}{T_2 + F_{21}} \right) \quad (6.7)$$

Faktycznie *wAcc* jest wartością średnią ze wszystkich możliwych TPR_i . W przypadku kiedy klasy mają dokładnie tyle samo reprezentantów w badanym zbiorze miary *Acc* i *wAcc* są sobie równe. Natomiast, im większe różni-

cowanie występuje w klasach, tym pojedynczy błąd predykcji dla nielicznej klasy silniej wpływa na łączną ocenę jakości.

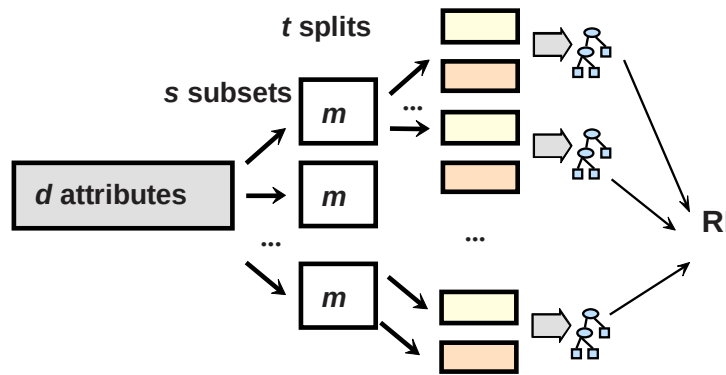
Dla bardziej ogólnego problemu, w którym mamy c klas, niech n_{ij} oznacza liczbę przykładów z klasy i zaklasyfikowanych jako klasa j . Inaczej mówiąc przyjmijmy że: $i, j = 1, 2, \dots, c$ i $\sum_{i,j} n_{ij} = n$, gdzie n to łączna liczba rozważanych przykładów. Dla tak zdefiniowanego zadania o dowolnej liczbie klas $wAcc$ będzie wyrażone poprzez wzór 6.8.

$$wAcc = \frac{1}{c} \sum_{i=1}^c \frac{n_{ii}}{n_{i1} + n_{i2} + \dots + n_{ic}}, \quad (6.8)$$

W zastosowaniach praktycznych dobór miary oceny klasyfikatora oraz akceptowalny poziom jego jakości zależy głównie od badanego problemu. Najchętniej w literaturze naukowej dla popularnych zbiorów służących do porównania różnych klasyfikatorów podaje się jednak *accuracy* lub błąd predykcji. Dla rozważanych w pracy – dość specyficznych problemów miary te są jednak nieadekwatne. Ekstremalnym przykładem doboru odpowiedniej oceny jakości klasyfikacji do problemu jest np. wykrywanie oszustw (ang. *fraud detection*), ponieważ w tego typu zadaniach zwykle przykłady z klasy pozytywnej (oszustwo) występują nieporównywalnie rzadziej niż negatywne (poprawna transakcja). W tej klasie problemów priorytetem jest minimalizacja miary *FPR* (klienci niesłusznie podejrzani o oszustwo) przy zadowalająco wysokim poziomie *TPR* (maksymalna ilość wykrytych oszustw).

6.3.2. Opis procedury

Ogólna koncepcja algorytmu MCFS (Monte Carlo Feature Selection) jest dość prosta, lecz wymaga budowy wielu klasyfikatorów – np. drzew decyzyjnych. Zakłada się, że odpowiednie atrybuty są istotne/informatywne jeśli odpowiednio często biorą udział w procesie klasyfikacji przykładów i taką „skłonność” atrybutu ogólnie nazwiemy relatywną istotnością (*relative importance*). Użycie drzew decyzyjnych zostało podyktowane ich niewątpliwymi zaletami: szybkością uczenia i testowania, klarownością opisu, wysoką jakością predykcji. Należy tu jeszcze raz dobitnie podkreślić, że w zadaniu nie skupiamy się na jak najlepszej klasyfikacji przykładów a na wyborze (stworzeniu rankingu) cech najbardziej istotnych z punktu widzenia decyzji.



Rysunek 6.2. Schemat blokowy podstawowej procedury MCFS

W podstawowym kroku procedury (patrz rys. 6.2) relatywna istotność jest estymowana poprzez konstruowanie tysięcy drzew (ogólniej klasyfikatorów) dla losowo wybranego podzbioru atrybutów. Mówiąc bardziej dokładnie spośród d cech, losowo wybiera się s podzbiorów o m atrybutach. Parametr m jest ustawiany tak by był dużo mniejszy niż liczba wszystkich dostępnych atrybutów ($m \ll d$). W celu wygody opisu nazwiemy ten proces dużą pętlą.

Dla każdego wylosowanego podzbioru cech buduje się następnie t drzew na losowo wybranych zbiorach trenujących, zapamiętuje użyte w klasyfikatorach cechy i na koniec testuje zapisując wynik predykcji. Podział na zbiór trenujący i testujący jest realizowany przy założeniu zachowania proporcji przykładów z poszczególnych klas zbioru wejściowego i domyślnie dane trenujące zawierają 66% losowo wybranych przypadków. Proces podziału na poszczególne zbiory uczące, budowę klasyfikatora oraz ocenę predykcji nazwiemy pętlą wewnętrzną. Faza ta ma na celu stabilizację wyniku w przypadku jeśli zestaw atrybutów jest sukcesywnie wybierany do budowy skutecznego klasyfikatora bez względu na podział na próby uczącą i testującą.

Łącznie w procedurze budowanych oraz poddanych ewaluacji jest st drzew. Oba parametry s i t powinny być wystarczająco duże by każda cecha miała realną szansę pojawienia się wielokrotnie w różnych losowych podzbiórach atrybutów. Jeśli spośród $d = 100$ atrybutów losujemy $m = 10$ ($m = 0.1d$) to warto tak ustawić s , t by każdy atrybut miał szansę wystąpić tyle razy ile jest atrybutów. Prawdopodobieństwo wylosowania atrybutu to $P_a = 0.1$ czyli $s=1000$. Duża pętla algorytmu sprzyja większej eksploracji (poszukiwaniu istotnych genów) natomiast pętla wewnętrzna/mała sprzyja bardziej eksploatacji (jeśli informatywne geny zostały wylosowane to zostaną wielokrotnie użyte do budowy kolejnych klasyfikatorów dla różnych prób uczących). Z tego powodu rozsądnie jest tak dobrać parametry s , t by $s \gg t$ np. $s = 1000$, a $t = 10$.

Wybór wielkości podzbioru m cech wybieranych dla każdej serii eksperymentów t powinien być rezultatem kompromisu pomiędzy potrzebą zabezpieczenia się przez maskowaniem dobrych cech przez najlepsze i naturalnym wymaganiem by wartość s nie była zbyt wielka. Rzeczywiście, mniejsza wartość m zmniejsza szansę (wpływa korzystnie) na maskowanie cech nawzajem, niestety wymusza jednocześnie dużą wartość s tak by wszystkie atrybuty miały realną szansę wielokrotnego pojawienia się w wybranym podzbiórce cech. Problem maskowania oraz próby niwelowania go będą omówione w dalszej części pracy. W literaturze problem ten porusza np. Breiman et al. w [Breiman et al., 1984].

6.3.3. Miara jakości atrybutu dla drzewa decyzyjnego

Miara jakości atrybutu pierwotnie została przez nas opracowana dla algorytmu C4.5 niemniej jednak jest na tyle uniwersalna, by mogła być z powodzeniem stosowana również dla implementacji innego algorytmu Sliq dostępnej w dmLab.

$$RI_{g_k} = \sum_{\tau=1}^{st} (wAcc(\tau))^u \sum_{n_{g_k}(\tau)} IG(n_{g_k}(\tau)) \left(\frac{\text{no. in } n_{g_k}(\tau)}{\text{no. in } \tau} \right)^v, \quad (6.9)$$

W poprzednim rozdziale ogólnie scharakteryzowano relatywną istotność jako częstość występowania danej cechy w poszczególnych klasyfikatorach w MCFS, dodatkowo jednak łączy ona w sobie istotność cechy w samym klasyfikatorze oraz jakość predykcji drzewa zbudowanego w oparciu o nią. Relatywna istotność cechy g_k , RI_{g_k} , będzie zatem zdefiniowana wzorem 6.9, który łączy w sobie te kryteria. Jeśli poszczególny podział jest przeprowadzony na atrybucie g_k (węzeł drzewa), to atrybut ten jest tym bardziej informatywny im większe są: przyrost informacji w węźle, liczba przykładów separowanych poprzez podział w tym węźle oraz wartość predykcji $wAcc$ zmierzona na zbiorze testującym dla drzewa zawierającego ten węzeł. Przyrost informacji może być mierzony za pomocą jednej z miar służących do oceny podziałów w węźle takich jak: *Gain Ratio*, *Information Gain* lub *Gini Index*. O ile użycie miary oceny podziału w węźle uwzględnia ocenę poszczególnych atrybutów na zbiorze uczącym to czynnik $wAcc$ odpowiednio podnosi RI_{g_k} cech, które weszły w skład drzewa skutecznie klasyfikującego nieznane przykłady.

We wzorze 6.9 sumowanie jest przeprowadzone po wszystkich st drzewach, a wewnątrz każdego τ -tego drzewa, po wszystkich węzłach $n_{g_k}(\tau)$, w których podział jest przeprowadzony na atrybucie g_k . Czynniki $IG(n_{g_k}(\tau))$ odpowiada za przyrost informacji dla węzła $n_{g_k}(\tau)$; $(\text{no. in } n_{g_k}(\tau))$ oznacza liczbę przypadków w węźle $n_{g_k}(\tau)$; $(\text{no. in } \tau)$ opisuje liczbę przypadków w korzeniu drzewa τ . Wartości u i v są dodatnimi parametrami równania. Warto podkreślić, iż dla wartości $u = 2$ drzewa z niskimi wynikami predykcji ($wAcc$) są karane znacznie mocniej niż w przypadku $u = 1$. Podobnie im większa jest wartość v tym mniejszy jest wpływ węzła $n_{g_k}(\tau)$ z danym stosunkiem $(\text{no. in } n_{g_k}(\tau))/(\text{no. in } \tau)$ na łączną wartość RI_{g_k} , chyba że $n_{g_k}(\tau)$ jest korzeniem drzewa.

Dla każdego dowolnie ustawionego dodatniego v , wpływ poszczególnego węzła na RI_{g_k} zmniejsza się monotonicznie z liczbą przykładów w tym węźle. Iloczyn stosunku ilości przykładów z przyrostem informacji dla danego węzła wpływa na obniżenie znaczenia wysokiej wartości $IG(n_{g_k}(\tau))$ w sytuacji gdy bezbłędnie separowane są znikome ilości przykładów. Przykładowo dla węzła, w którym podział pozwala na separację dwóch przykładów różnych kategorii od siebie, niewielka liczba przykładów będzie dodatkowo karać atrybut w węźle.

Warto również zauważyć, że dla wartości $u = v = 0$ i dla czynnika $IG(n_{g_k}(\tau))$ podniesionego do potęgi zerowej, otrzymuje się "surową" miarę relatywnej istotności dla cechy g_k . Miara ta oznaczona cRI_{g_k} (*crude RI*) będzie równa całkowitej liczbie podziałów dokonanych na g_k we wszystkich węzłach wszystkich st drzew, czyli opisuje jedynie częstotliwość wystąpień cechy.

W podanej procedurze istnieje pięć parametrów m , s , t , u i v , które należy ustawić w celu uruchomienia eksperymentu. We wcześniejszych rozważaniach podano informacje pomocne do ustawienia parametrów m , s , t . Ponadto

nie ma żadnych przeszkód by dostarczyć i porównać wiele rankingów, każdy oparty na RI_{g_k} z różnymi ustawieniami wartości par u i v (np. $u = 0, 1, 2$ i $v = 0.5, 1, 2$). Porównanie otrzymanych rankingów dla przykładowych danych pozwoli na ocenę wpływu u, v oraz rekomendację domyślnych ustawień.

W części eksperymentalnej przedstawiono wpływ wszystkich wyżej wymienionych parametrów i rozsądne domyślne ustawienia wynikające z tych doświadczeń.

6.3.4. Miara jakości atrybutu dla klasyfikatora ADX

Implementacja algorytmu MCFS w oparciu o drzewa decyzyjne wykazała wysoką efektywność zaproponowanej procedury oraz definicji relatywnej istotności cechy. Skoro klasyfikator jest w zasadzie "czarną skrzynką" z punktu widzenia procedury MCFS to wydaje się, że warto jest sprawdzić jak radzi sobie w zadaniu rangowania atrybutów zupełnie inny typ klasyfikatora. Kryteria jakie musi spełniać taki klasyfikator są dwa:

- odpowiednia szybkość działania (porównywalna z drzewem decyzyjnym),
- możliwość zdefiniowania miary RI_{g_k} .

Wydaje się zatem, że reguły klasyfikator ADX bardzo dobrze nadaje się do tego zadania. Skoro w MCFS buduje się tysiące klasyfikatorów to szybkość działania (uczenie i testowanie) pojedynczego klasyfikatora musi być na tyle duża by cała procedura miała szansę zakończyć się w rozsądnym czasie. Te właściwości klasyfikatora regułowego zostały zbadane a wyniki eksperymentów przedstawione w odpowiedniej sekcji pracy, natomiast tutaj jedynie stwierdzimy, że prędkość działania jest wystarczająca. Pozostaje zatem zaproponować miarę relatywnej istotności atrybutu g_k dla algorytmu ADX. Miara zdefiniowana w analogiczny sposób łączy w sobie jakość predykcji, częstość występowania cechy g_k , oraz jej udział w zbiorze reguł. Wzór 6.10 definiuje RI_{g_k} dla ADX.

$$RI_{g_k} = \sum_{\sigma=1}^{st} (wAcc(\sigma))^u \sum_{r_{g_k}(\sigma)} Q(r_{g_k}(\sigma)) (cov(r_{g_k}(\sigma)))^v, \quad (6.10)$$

Podobnie jak we wzorze 6.9 sumowanie jest przeprowadzone po wszystkich st klasyfikatorach regułowych, a wewnątrz każdego klasyfikatora σ , po wszystkich regułach r_{g_k} . Zapis $Q(r_{g_k}(\sigma))$ oznacza jakość reguły r_{g_k} , w której wystąpił atrybut g_k w klasyfikatorze σ . W ADX, jak podano w części opisującej sam klasyfikator, zdefiniowano kilka różnych możliwych do użycia metod liczenia jakości Q dla reguły, dlatego też w MCFS dobór Q sterowany jest poprzez odpowiedni parametr. Ostatni czynnik równania 6.10 uwzględnia pokrycie cov danej reguły. Dzięki temu dodatkowo można zróżnicować atrybuty występujące w regułach o podobnym Q – te z mniejszym pokryciem będą miały niższą wartość RI . Wpływ parametrów u i v jest identyczny w działaniu jak dla RI zdefiniowanego dla drzewa decyzyjnego.

Dla danych wysokowymiarowych o małej liczbie przykładów (np. opartych na ekspresji ludzkich genów) klasyfikator C4.5 przeważnie buduje bardzo proste drzewa. Większość z nich jest oparta na dwóch, trzech atrybutach co

zresztą nie jest zaskakujące ponieważ dysponując przestrzenią o wymiarze kilku tysięcy dobra separacja kilkudziesięciu przykładów jest zadaniem stosunkowo prostym. Prostota poszczególnych drzew wymusza budowę wielu tysięcy takich klasyfikatorów by można było ocenić wszystkie geny. W przeciwieństwie do drzew decyzyjnych, w ADX buduje się zestaw reguł (wcale nie minimalny). Pewna nadmiarowość reguł oraz specyfika algorytmu ADX prowadzi do bardziej rozmytego, niż w przypadku drzew, lecz pełniejszego opisu. Większa ilość informacji na temat poszczególnych atrybutów w pojedynczym klasyfikatorze pozwala przypuszczać, iż dla klasyfikatora ADX ranking najważniejszych cech powinien ustalać się w mniejszej liczbie iteracji algorytmu MCFS.

Kolejną niewątpliwą zaletą użycia algorytmu ADX w MCFS jest możliwość generowania osobnych rankingów cech dla każdej klasy. Dla każdej wartości decyzji budowany jest osobny zestaw reguł, który służy do budowy rankingu zbiorczego oraz specjalizowanego dla konkretnej klasy.

Podczas budowy rankingu niezależnego od klasy decyzyjnej suma po $(r_{g_k}(\sigma))$ odbywa się na wszystkich zestawach reguł. W przypadku budowy osobnych rankingów bierze się tylko zestawy reguł σ^c dla rozważanej wartości decyzji. Wzór 6.11 definiuje sposób liczenia relatywnej istotności RI cechy g_k dla klasy c .

$$RI_{g_k}^c = \sum_{\sigma^c=1}^{st} (wAcc(\sigma))^u \sum_{r_{g_k}(\sigma^c)} Q(r_{g_k}(\sigma^c)) (cov(r_{g_k}(\sigma^c)))^v, \quad (6.11)$$

6.3.5. Odległość rankingów cech

W opisie procedury MCFS opisywany jest zestaw trzech najważniejszych parametrów s , t i m , które użytkownik musi zadać przed uruchomieniem algorytmu. Z praktycznych obserwacji po przeprowadzeniu serii eksperymentów wydaje się, że można określić "bezpieczne" wartości owych parametrów, domyślne dające ustabilizowane rankingi niezależnie od problemu. Ich nierozsądne ustawienie może jednak doprowadzić do bardzo niedokładnych wyników (np. w przypadku nie wystarczającej liczby s). W części eksperymentalnej przedstawiono wpływ poszczególnych parametrów na działanie procedury MCFS dla wysokowymiarowych zbiorów danych. Dla ustalonego parametru m (np. 10–20% dostępnych cech) krytycznym wyborem jest ustawienie wartości s (definiującej liczbę losowanych podzbiorów cech), która powinna być na tyle duża by pozwolić na ustabilizowanie się rankingu. Pamiętajmy jednak, że w praktyce podstawowym celem postawionego zadania selekcji cech jest otrzymanie rankingu tylko najważniejszych atrybutów i przyjmując takie założenie można zdefiniować jasne kryterium zbieżności algorytmu MCFS.

Dla ustalonych m , t i s procedura jest uruchomiona tak, że w kolejnych momentach czasowych $s_1, s_2, s_3 \dots s$ zapamiętuje się kolejne rankingi cech (również dla ustalonych u i v). Przy czym $s_2 = s_1 + k$, $s_3 = s_1 + 2k = s_2 + k$, $s_n = s_1 + (n-1)k$, natomiast k przeważnie wynosi 10. Każdy kolejny ranking określony w momencie czasu s_n porównuje się z poprzednim $s_n - k$, a brak

drastycznych zmian w kolejnych porównywanych rankingach można uznać za stabilizację wyniku.

W praktyce istotny zbiór cech stanowi właściwie tylko czołówka rankingu, toteż porównuje się tylko zdefiniowane $p\%$ najwyżej ocenionych aktualnie atrybutów (około 10%; parametr *bestSetFractionSize*). Porównywanie całych rankingów dla wysokowymiarowych problemów nie pozwala raczej ocenić stabilizacji całości wyniku, ponieważ, o ile czołówka listy ustala się dość szybko, to większa część nieużytecznych atrybutów może przemieszczać się w rankingu dość gwałtownie (zwłaszcza dla niefortunnie wylosowanego podzbioru cech). Większa część atrybutów dla wspomnianych danych jest bezużyteczna i wartość RI dla tej grupy wynosi zero. Zdarza się jednak tak, że jedna z niewybranych do tej pory cech pojawia się w klasyfikatorze a co za tym idzie jej RI staje się większe od zera. W takim przypadku atrybut w rankingu wykonuje duży skok co wpływa na miarę oceny podobieństwa dwóch rankingów.

Począwszy od s_1 tworzy się kolejne listy atrybutów posortowanych według miary RI dla ustalonej pary (u, v) (patrz wzory 6.9, 6.10).

Odległość (dystans) pomiędzy dwoma rankingami definiuje wzór:

$$\text{Dist}(s_1, s_2) = \frac{1}{d_p} \sum_{g_k} |\text{rank}(g_k, s_1) - \text{rank}(g_k, s_2)|, \quad (6.12)$$

gdzie:

$$d_p = \frac{dp}{100} \quad (6.13)$$

W równaniu 6.12 sumuje się po najwyżej ocenionym p procencie d atrybutów g_k otrzymanych w s_2 ; $\text{rank}(g_k, r)$ określa pozycję w rankingu atrybutu g_k po ustaleniu rankingów. Stała d_p równa ilości rozważanych atrybutów jedynie normalizuje poszczególne odległości. Parametr p nie powinien być zbyt duży i ustalony w zakresie od 5 do 20 procent. Dowolna wartość z tego przedziału dla wysokowymiarowych danych genetycznych, daje prawie identyczne wyniki ponieważ tylko relatywnie mała grupa genów jest rzeczywiście informatywna. Warto zauważyć że po wylosowaniu s_n podzbiorów m cech maksymalnie da się ocenić $s_n m$ spośród wszystkich atrybutów, a prawdopodobieństwo otrzymania tej górnej granicy wynosi praktycznie zero. Z tego powodu w początkowej fazie działania MCFS dla małych s otrzymujemy wysokie wartości odległości rankingów, które z czasem ulegają zmniejszeniu, aż do momentu osiągnięcia pewnej stabilizacji na pewnym akceptowalnym poziomie bliskim zera. Jeśli w czołówce rankingu s_2 występuje atrybut g_k , taki który znajduje się poza rankingiem określonym przez d_p w s_1 , to przyjmujemy jego pozycję jako $d_p + 1$.

Definicja dystansu podana we wzorze 6.12 nie pozwala na porównywanie tej miary dla różnych wartości parametru *bestSetFractionSize*. Prosty zabieg normalizacji (wzór 6.14) – podzielenia wartości przez możliwą maksymalną, pozwala na łatwiejsze manipulowanie wynikami miary dystansu. Maksymalną wartość miary można określić na podstawie wyliczenia jej dla dwóch rankingów zawierających zupełnie różne listy cech, przy czym należy pamiętać że wartość dystansu (jak również odpowiedniej maksymalnej jego warto-

ści) zależy od długości listy rankingu. Obie wartości muszą zatem być liczone indywidualnie w zależności od ustawienia parametru *bestSetFractionSize* w algorytmie MCFS.

$$(\text{Dist})_{\text{norm}}(s_1, s_2) = \frac{\text{Dist}(s_1, s_2)}{\text{Dist}_{\text{max}}(s_1, s_2)} \quad (6.14)$$

Warto również zwrócić uwagę, że miara 6.12 jest równoważna ρ Spearmana, przy czym odległość rankingów bazuje na różnicach pozycji (w przeciwieństwie do kwadratów tych różnic) tak by wynik określał średnią odległość pozycji zajmowanych przez atrybuty w obu rankingach.

6.3.6. Kryterium zbieżności algorytmu

Przyjmując odległość rankingów jako miarę niepodobieństwa można sobie wyobrazić sytuację, kiedy kolejne jego wartości ustabilizują się na pewnym akceptowalnie niskim poziomie bliskim zeru, natomiast bieżąca iteracja s_n jest jeszcze daleka od zadanej przez użytkownika s_{max} . Oznacza to, że osiągnięto już stabilny wynik i kolejne pętle s nie wniosą już żadnej dodatkowej informacji, a zatem można przerwać działanie algorytmu.

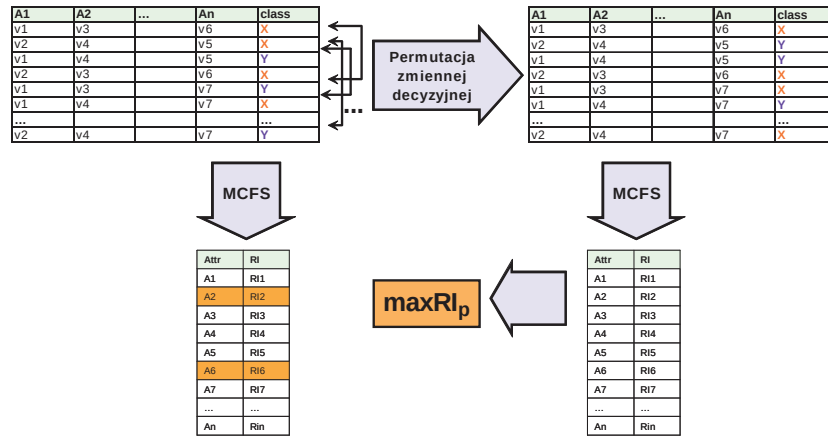
Skoro przedstawiono metodę badania zbieżności algorytmu to wydaje się, że kolejnym krokiem może być zaproponowanie kryterium, które oceni czy dalsze iteracje procedury MCFS są nadal potrzebne. Ogólna zasada opiera się na badaniu kolejnych odległości rankingów i stwierdzeniu, czy wartości te ustaliły się na niezmiennym niskim poziomie, co w konsekwencji prowadzi do wniosku o osiągnięciu stabilnego rezultatu. Domyślnie liczenie odległości pomiędzy kolejnymi rankingami dokonuje się co 10 kolejnych projekcji. Krzywa powstała poprzez połączenie poszczególnych punktów zostaje następnie wygładzona poprzez filtr uśredniający (np. okno wielkości 5), a następnie bada się zmianę $\Delta \text{Dist}(s_n, s_{n+10})$. Kryterium stopu algorytmu MCFS można sprowadzić zatem do ustalenia czy kolejne wartości $\Delta \text{Dist}(s_n, s_{n+10})$ są stałe i bliskie zera.

6.3.7. Selekcja istotnego zbioru atrybutów

Po analizie przedstawionej procedury MCFS można zauważyć, iż bez względu na jakość badanego zbioru finalnie po zbudowaniu st drzew otrzyma się pewien ranking cech. Niezależnie czy następujące po sobie rankingi były odległe od siebie, budowa kolejnych drzew powoduje zawsze ocenę użytych cech a tym samym zwiększa wartość ich RI_{gk} co pozwala na posortowanie listy atrybutów według RI i otrzymanie rankingu. Przy odpowiednio zadanym parametrze *bestSetFractionSize* oraz po ustaleniu się dystansu rankingów można mieć pewność, że część rankingu najwyżej ocenionych atrybutów jest obiektywnym wynikiem dla badanego zbioru danych. Trudno jest natomiast jednoznacznie stwierdzić (ustalić poziom odcięcia dla RI), które z atrybutów występujących na liście są rzeczywiście informatywne.

W praktyce dla dużych zbiorów wiele cech nie zostaje nigdy ocenionych ponieważ nigdy nie wyselekcjonowano ich do budowy klasyfikatorów i wtedy znajdują się one na spodzie rankingu gdzie $RI = 0$. Są jednak i takie atrybuty, które wybrano, na wskutek braku lepszych w aktualnie wylosowanym

podzbiórze cech, i mimo słabej jakości klasyfikatora zostały ocenione i uszeregowane w rankingu. W danych opisujących ekspresje genów przeważnie dostępnych jest kilka tysięcy atrybutów (spośród których kilkadziesiąt jest rzeczywiście istotnych), natomiast ocenie podlega ich znaczna część (każdy wybrany przynajmniej raz do budowy dowolnego klasyfikatora). Nie oznacza to wcale, iż wszystkie atrybuty z wartościami RI większymi od zera są informatywne, ponieważ część z nich mogła zostać użyta do zbudowania klasyfikatorów o jakości porównywalnej z losowym klasyfikatorem. Pewne jest jednak to iż jeśli dystans rankingów zbiegł do wartości bliskiej zera to w pozycje na liście rankingu wyznaczają hierarchię ważności cech.



Rysunek 6.3. Schemat selekcji istotnego zbioru cech

Do znalezienia poziomu RI granicy informatywności, można przeprowadzić dodatkową serię eksperymentów dla permutowanej za każdym razem wartości decyzji. Taki zabieg pozwala na zbudowanie analogicznego rankingu cech, o których z góry wiadomo, że nie są informatywne ponieważ brak w zbiorze realnej zależności z atrybutem decyzyjnym. Zbiór maksymalnych uzyskanych za każdym razem wartości RI (zbiór $\max RI_p$) dla tak przeprowadzonej serii eksperymentów dają silną podstawę do odrzucenia cech o wartościach podobnych lub niższych dla eksperymentu oryginalnego. Na rysunku 6.3 przedstawiono schemat omówionego sposobu wyboru cech, wszystkie atrybuty (zaznaczone kolorem pomarańczowym) o wartości RI większej niż $\max RI_p$ uznajemy za informatywne. Na schemacie tabele z wartościami RI nie przedstawiają rankingów cech (posortowanych po RI) lecz listę atrybutów posortowaną w.g. kolejności w danych.

Zbiór wartości $\max RI$ z przeprowadzonego szeregu eksperymentów z permutacją może być postrzegany jako rozkład pewnej zmiennej losowej. Wydaje się, że rozkład ten powinien oscylować wokół pewnej średniej wartości i mieć charakter rozkładu normalnego – co można sprawdzić przy pomocy np. testu Shapiro-Wilka.

Następnie wystarczy zastosować test-t w celu sprawdzenia, jaka dokładnie wartość RI odpowiada p – value na poziomie 0.05 dla rozkładu $\max RI$. Wszystkie cechy, które zostały ocenione wartością wyższą niż wyznaczone RI odcięcia w oryginalnym eksperymencie z bardzo dużym prawdopodobieństwem nieprzypadkowo zostały wysoko ocenione. Wyznaczona graniczna

wartość *RI* może w dość dobrze uzasadniony statystycznie sposób określać zbiór istotnych cech (poziom odcięcia).

6.3.8. Problem wzajemnego maskowania atrybutów

Po sformułowaniu głównych założeń procedury MCFS wydaje się, że można budować rankingi cech dla specyficznych wysokowymiarowych danych, dla których przeważnie minimalna liczba cech potrzebna do dobrej klasyfikacji jest znacznie mniejsza niż liczba wszystkich istotnych atrybutów. Oznacza to np. budowę rankingu dla problemu w którym walidacja krzyżowa daje niezwykle wysokie wyniki dla drzew budowanych na zaledwie 2 atrybutach (dla problemu o trzech klasach). Zbiór danych Alizadeha [Alizadeh *et al.*, 2000] badany w pracy w związku z podanym zagadnieniem jest dość osobliwy i wymaga bardziej wnikliwej analizy.

Po przeprowadzeniu procesu walidacji krzyżowej (np. 3 zbiory) dla wszystkich dostępnych cech dla tego zbioru wyraźnie widać, że doskonałe rezultaty klasyfikacji można uzyskać przy użyciu tylko dwóch genów: *GENE1622X* i *GENE2426X*. Pierwszy idealnie separuje klasę DLCL natomiast drugi oddziela od siebie dwie mniejsze. Po wyłączeniu ze zbioru danych tych atrybutów drzewo decyzyjne budowane jest w oparciu o dwa następne radząc sobie równie dobrze z problemem klasyfikacji. Po analizie okazuje się, że istnieje aż 8 genów idealnie oddzielających największą klasę DLCL od reszty przykładów, dodatkowo dla każdego z nich na drugim poziomie drzewa pojawia się szereg atrybutów bardzo skutecznie separujących klasy FL i CLL. Wydaje się zatem, że może to prowadzić dla zbioru Alizadeha do sytuacji gdzie finalny ranking zostanie silnie zdominowany przez cechy separujące klasę DLCL od reszty, dodatkowo zafałszowany poprzez wzajemne maskowanie atrybutów. Dlatego kolejne propozycje modyfikacji MCFS skupiają się na uwydatnieniu w końcowym rankingu cech, które są skuteczne w odróżnianiu dwóch pomniejszych klas od siebie (FL i CLL).

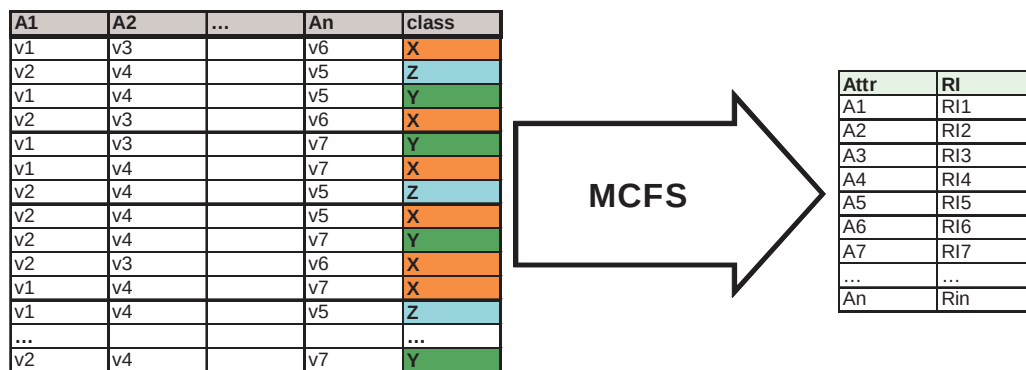
W szczególności właśnie dla danych Alizadeha uwidoczniła się potrzeba sprawdzenia kilku różnych trybów działania pierwotnej realizacji procedury MCFS. W modyfikacjach chodzi głównie o sprawdzenie czy tak specyficzne dane niekorzystnie wpływają na podejście klasyczne.

- Klasyfikacja w MCFS wszystkich klas (klasyczna wersja procedury).
- Klasyfikacja w MCFS wszystkich klas z kolejnym wyłączaniem ze zbioru cech najwyżej dotychczas ocenionych.
- MCFS uruchamiane osobno na zbiorach, gdzie jedna klasa jest przeciwstawiona pozostałym połączonym ze sobą.
- MCFS uruchamiane osobno na zbiorach, zawierających zawsze tylko parę klas.

W implementacji procedury MCFS parametrem sterującym trybem uruchomienia jest parametr: *mode*.

Wszystkie klasy - klasyczne podejście

W klasycznym podejściu walka z wzajemnym przesłanianiem równie informatywnych cech jest zawarta w losowaniu podzbioru cech w kolejnych



Rysunek 6.4. Schemat MCFS dla klasyfikacji 3 klas

pętlach algorytmu MCFS. Im mniejsza jest wartość parametru m (liczba losowo wybranych cech w podzbiorze) tym mniejsze jest prawdopodobieństwo pojawienia się kilku silnie skorelowanych ze sobą cech w jednym podzbiorze. W takim wypadku przy odpowiednio dobranym m w podzbiorach cech nie powinno występować na raz wiele cech informatywnych co z kolei implikuje wybór tylko tych ważnych do budowy pojedynczego klasyfikatora. Pojawienie się atrybutu w klasyfikatorze praktycznie zapewnia mu zwiększenie jego wartości oceny a tym samym znalezienie się w rankingu cech. Klasyczne podejście metody MCFS powinno uszeregować wszystkie dostępne atrybuty w takiej kolejności w jakiej rzeczywiście są one istotne ze względu na miarę oceny RI .

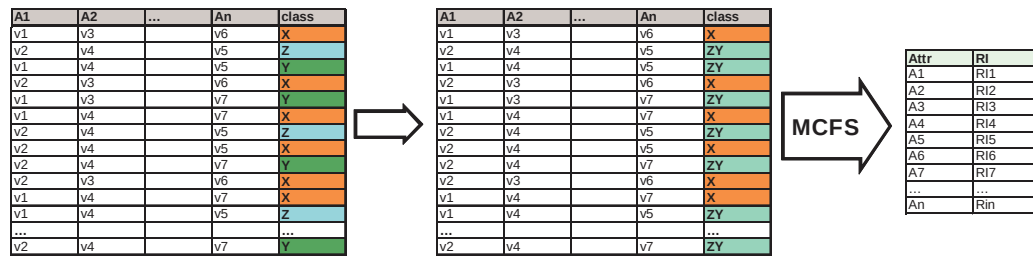
Wszystkie klasy - kolejne wyłączanie grupy najlepszych cech

Przy ustawieniu parametru *removeBestAttributes* = *true* można przeprowadzić serię eksperymentów MCFS, lecz po zakończeniu każdego z nich czołowe cechy z rankingu są usuwane ze zbioru dla kolejnego uruchomienia. Liczba tak zdefiniowanych powtórzeń jest definiowana przez parametr *runs* natomiast liczba usuwanych sprawdzonych i istotnych atrybutów poprzez *bestSetRemoveSize*. Podejście takie powinno dla specyficznych danych pozwolić na uwydatnienie w rankingu pewnych atrybutów, które być może są zdominowane przez te ocenione do tej pory najwyżej. Jeśli jednak klasyczne podejście pozwala na obiektywne wykrywanie rankingu (nawet dla specyficznych problemów w tej klasie) to kolejne odrzucane czołówki rankingów powinny dać w sumie ranking tożsamy z uzyskanym bez usuwania.

Jedna klasa przeciwstawiona pozostałym połączonym

W dwóch kolejnych możliwych trybach uruchomienia MCFS wykorzystuje się sprowadzenie problemu wieloklasowego do nowego problemu o binarnej decyzji. Należy jednak zaznaczyć, że taki zabieg wcale nie musi uprościć problemu ponieważ podczas gdy 3 klasy stosunkowo łatwo dają się separować w rozważanej przestrzeni to połączenie dwóch z nich może sformułować kompletnie nowe trudniejsze zadanie.

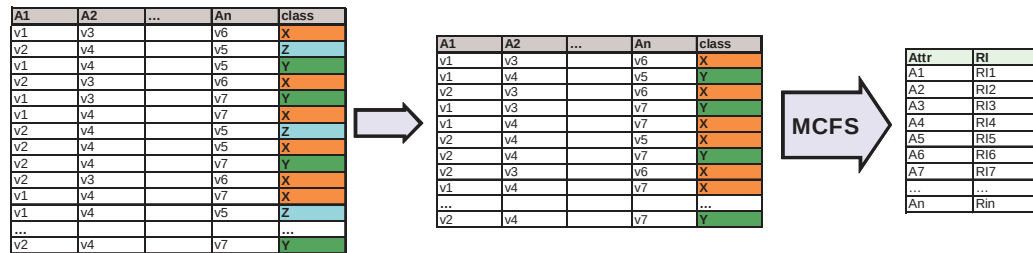
W tym omawianym trybie działania MCFS dla jednego zbioru danych o trzech wartościach decyzji automatycznie przeprowadza się trzy sub-eksperymenty. W każdym takim eksperymencie MCFS działa na zbiorze danych, w których



Rysunek 6.5. Schemat MCFS dla klasyfikacji 1 klasy przeciwko 2 połączonym ze sobą.

odpowiednio kolejne dwie klasy zostały jednakowo zaetykietowane. Przykładowo za pierwszym razem rozważamy klasy DLCL i FL+CLL, za drugim FL i DLCL+CL, natomiast za trzecim CL i DLCL+FL. Z każdego takiego eksperymentu otrzymujemy osobny ranking, w którym powinny znajdować się na czołowych miejscach cechy silniej związane z możliwościami separacji odpowiedniej klasy (samotnej w rozważanym zbiorze).

Wyłączenie ze zbioru jednej klasy



Rysunek 6.6. Schemat MCFS dla klasyfikacji tylko pary klas

Drugi sposób sprowadzenia problemu o 3 klasach do problemu z binarną decyzją jest zwyczajne usunięcie jednej z klas i rozważanie tylko pozostałych. W tym trybie również przeprowadza się 3 sub-eksperymenty przy czym w każdym inna klasa zostaje usunięta ze zbioru. Finalnie otrzymujemy również 3 rankingi, każdy reprezentujący inny zestaw cech które potrafią rozróżniać od siebie dwie klasy. Dla danych Alizadeha z racji na sporą grupę świetnie separujących genów klasę *DLCL* można się spodziewać, iż czołówki rankingów po usunięciu klasy CLL oraz FL będą podobne do siebie. Geny, które rozdzielają dobrze DLCL od obu klas (CLL i FL) będą dobrze oddzielać DLCL również od każdej z pozostałych klas z osobna.

Problem wzajemnego maskowania atrybutów został zbadany, a wyniki eksperymentów oraz stosowny komentarz zamieszczono w sekcji 7.2.4.

6.3.9. Problem silnie niezbalansowanych klas

W przypadku silnego zróżnicowania wielkości klas w zbiorze danych czołowe pozycje w rankingu cech mogą zostać zdominowane przez atrybuty, które dobrze separują największą spośród dostępnych klas. W implementacji algorytmu MCFS zastosowano parametr *balanceClasses*, który odpowiada

za automatyczny dobór wielkości klas w wylosowanych zbiorach uczącym i testującym. Podczas gdy parametr *splitRatio* określa stosunek podziału na oba zbiory to *balanceRatio* określa ile maksymalnie razy liczba przykładów z klasy może być większa od analogicznej liczby dla najmniejszej klasy.

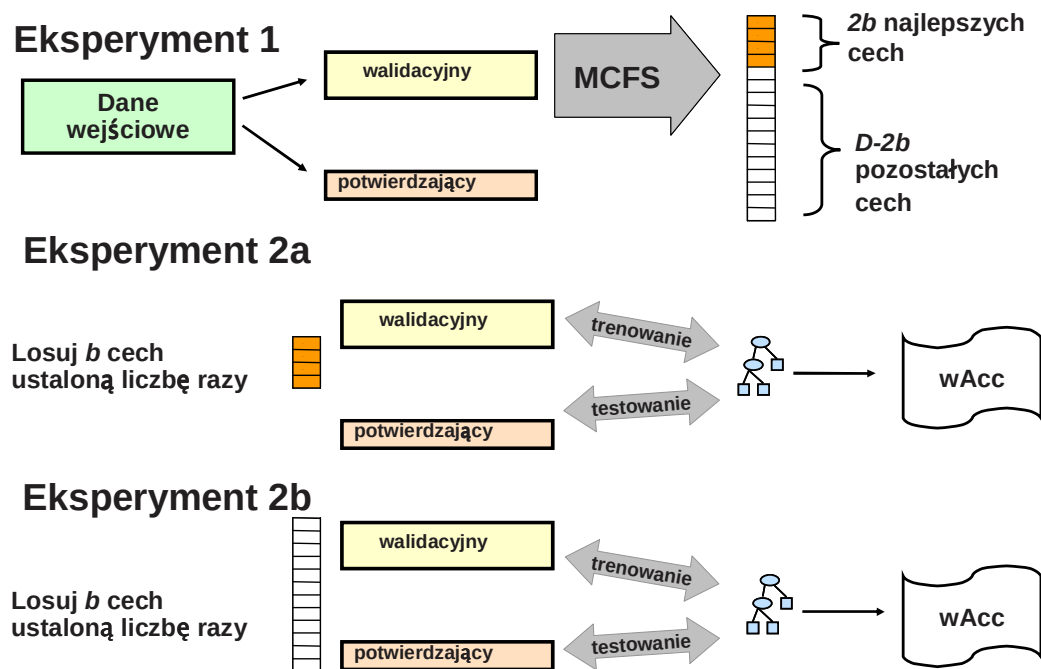
W przypadku gdy parametr *balanceClasses* został ustawiony jako 'true' to jeśli dla jakiegokolwiek klasy wartość $|D^c|$ (liczba przykładów należących do klasy c) jest większa od wartości $\text{balanceRatio}|\min(D^c)|$ to liczba przykładów podlegających podziałowi dla tej klasy będzie równa $\text{balanceRatio}|\min(D^c)|$. Taki zabieg pozwala na silniejsze zaakcentowanie stosunkowo bardzo małej klasy, ponieważ przy mniejszym zróżnicowaniu wielkości klas atrybuty istotne dla małej klasy będą chętniej wybierane poprzez poszczególne klasyfikatory. Mimo zaburzenia koncentracji klas decyzyjnych, finalnie otrzymany zestaw istotnych atrybutów jest nadal obiektywny ponieważ pojedyncze drzewa są budowane w oparciu o odmienne losowo wybrane przykłady z dominującej klasy. W sekcji poświęconej miarom oceny omówiono miarę *wAcc* (6.8), która w bardziej sprawiedliwy sposób ocenia wyniki klasyfikacji dla silnie niezbalansowanych zbiorów, tutaj natomiast prezentujemy prosty zabieg zwiększający szanse zaistnienia w klasyfikatorze cech charakteryzujących dla stosunkowo małych klas.

Warto tu jeszcze zaznaczyć, że podczas losowego podziału na zbiory trenujący i testujący w MCFS wymusza się zachowanie koncentracji klas w zbiorach. Obiekty do zbioru trenującego są losowane z rozkładem równomiernym w ramach każdej klasy osobno a ich ilość określa parametr *splitRatio* (losujemy $\text{splitRatio}|D^c|$ przykładów, gdzie $|D^c|$ oznacza liczbę przykładów w klasie c). Przy tak mało licznych zbiorach, jak to ma miejsce w danych ekspresji genów, losowanie spośród wszystkich obiektów mogłoby spowodować silne zaburzenie koncentracji poszczególnych klas w zbiorach uczącym i testującym.

6.4. Opis procedur walidacyjnych i potwierdzającej

Znając naturę badanych zbiorów (bardzo wiele wymiarów - atrybutów w stosunku do liczby dostępnych przykładów) należy bardzo wnikliwie i obiektywnie ocenić uzyskane rezultaty z punktu widzenia statystycznego. W tym celu można zaproponować włączenie do całego procesu dwóch różnych kroków walidacyjnych i dodatkowo jednego kroku potwierdzającego.

W ogólnym zarysie dla danego zbioru danych, pierwszy krok walidacyjny skupia się na powtórzeniu podstawowej procedury z powiedzmy 50 różnymi permutacjami etykiet klas zmiennej decyzyjnej. Celem takiego działania jest potwierdzenie czy uprzednio uzyskany wynik klasyfikacji (mierzony poprzez rozkład wartości *wAcc* na wszystkich *st* klasyfikatorach podczas MCFS) jest istotnie różny od tego jaki można uzyskać po losowej permutacji wartości decyzji (co skutkuje pełną niezależnością klasy od wartości zmiennych wejściowych). W taki sposób można sprawdzić czy istnieje ewidentny nieprzypadkowy związek pomiędzy wartościami cech wejściowych, a klasą decyzyjną. Stwierdzenie tego faktu w pierwszym kroku walidacyjnym uzasadnia celowość przeprowadzonego eksperymentu.



Rysunek 6.7. Schemat blokowy procedur walidacyjnej i potwierdzającej

Drugi krok walidacyjny skupia się na rzetelnym uzasadnieniu czy predykcja klasy decyzyjnej może zostać przeprowadzona przy użyciu tylko b cech spośród $2b$ wcześniej uznanych za najbardziej istotne (np. używając metody opisanej w sekcji 6.3.7). Takiej weryfikacji można dokonać poprzez konstrukcję kolejnych tysięcy klasyfikatorów na losowo wybranych b atrybutach z dostępnych najwyżej ocenionych $2b$ i wyniki porównać z rezultatami eksperymentu gdzie losujemy b cech ze zbioru $|D| - 2b$. Dla każdego podzbioru b atrybutów konstruuje się wiele zbiorów trenujących i testujących a następnie uruchamia proces uczenia klasyfikatora i jego oceny zdolności predykcyjnych. W ten sposób znowu otrzymujemy dwa rozkłady wartości $wAcc$ dla klasyfikacji opartej o b cech, raz losowanych z $2b$ najistotniejszych, natomiast po raz drugi losowanych z pozostałych $|D| - 2b$ atrybutów. Celem tego kroku jest sprawdzenie istotnej różnicy w jakości klasyfikacji przy użyciu: tylko cech uznanych za informatywne, oraz wszystkich pozostałych.

Pomysł który się kryje w drugim omawianym sposobie walidacji jest bardzo prosty. Rozkład ważonej dokładności ($wAcc$) klasyfikatorów dla losowo wybranych wektorów cech spośród $|D| - 2b$ służy jako założony rozkład pod warunkiem hipotezy używania nieinformatywnych cech. Rozkład $wAcc$ dla wektorów z cechami uznanymi za informatywne przeciwstawia się rozkładowi dla założonej wcześniej hipotezy w celu weryfikacji dokonanego wyboru atrybutów. Jeśli oba rozkłady miary jakości predykcji $wAcc$ (otrzymanych na danych testowych) statystycznie istotnie różnią się od siebie to oznacza to, iż zbiór cech uznanych za informatywne rzeczywiście odbiega od reszty.

Przedstawiona propozycja walidacji wyników właściwie nie stwierdza niczego poza dokładnym poziomem istotności otrzymanych rezultatów. Wielokrotne próbkowanie danych wprowadza wewnętrzną wzajemną zależność w całej procedurze czyniąc wynik być może zależnym od danych. Z tego

powodu by potwierdzić jakość otrzymanego wyniku można zaproponować dodatkowo krok potwierdzający obiektywne działanie procedury. Pełen wejściowy zbiór danych dzieli się na dwa podzbiory w stosunku np. 75% do 25% przykładów w każdym z nich. Pierwszy z nich nazwiemy zbiorem *walidacyjnym* natomiast drugi zbiorem *potwierdzającym*. Przy użyciu zbioru walidacyjnego dokonujemy budowy rankingu i selekcji istotnych atrybutów za pomocą procedury MCFS. Schemat blokowy tej operacji został przedstawiony na rysunku 6.4 i oznaczony etykietą "Eksperyment 1". W ramach procedury MCFS zbiór walidacyjny jest wielokrotnie próbkowany (zarówno losowa projekcja atrybutów, jak również podział na dane trenujące i testujące) i zgodnie z podanym już opisem algorytmu generuje się dla niego finalny ranking cech. Następnie dokonuje się drugiego kroku walidacyjnego (opisanego już w tej sekcji) tzn. buduje rozkłady $wAcc$ dla losowo wybranych b cech dla zbioru z $2b$ (Eksperyment 2a) i $|D| - 2b$ (Eksperyment 2b) cechami. Przy czym trenowania poszczególnych klasyfikatorów w tej fazie dokonuje się na zbiorze walidacyjnym natomiast testowania na nie używanym do tej pory zbiorze potwierdzającym. W ten sposób otrzymane oba rozkłady poddane analizie statystycznej pozwolą jednoznacznie stwierdzić wybór rzeczywiście informatywnych cech. Statystycznie istotna różnica obu rozkładów, przy jednocześnie wyższej medianie dla zbioru opartego na $2b$ najlepszych atrybutach niepodważalnie prowadzi do wniosku o prawidłowości rankingu, a tym samym skutecznym działaniu MCFS.

Wyniki przeprowadzenia poszczególnych kroków walidacji i potwierdzenia wyników eksperymentów znajdują się w sekcjach 7.2.1 i 7.2.7.

7. Eksperymenty

Wszystkie eksperymenty przeprowadzono na komputerze Pentium IV 3.2 GHz HT, 2GB RAM pracującym w systemie Windows XP.

7.1. Zadania klasyfikacji

7.1.1. Badanie wpływu parametrów algorytmu ADX

Wpływ parametrów ADX badano w oparciu o kilka zbiorów danych (tabela 7.1) na tyle różnych by można było obserwować zachowanie algorytmu w możliwie zróżnicowanych warunkach.

zbiór	l. obiektów	l. atrybutów	l. klas	źródło
Artificial05_500	4000	2	2	dane syntetyczne
alizadehdata	62	4026	3	[Alizadeh <i>et al.</i> , 2000]
golubdata	38	7129	2	[Golub <i>et al.</i> , 1999]
oszusci2007_2	15123	33	2	PKN Orlen
mushroom	8123	22	2	[UCI Repository]
iris	150	4	3	[UCI Repository]
AbaloneData	4177	8	3	[UCI Repository]
soybean	683	35	19	[UCI Repository]
tic_tac_toe	958	9	2	[UCI Repository]

Tablica 7.1. Zbiory danych użyte do badania wpływu parametrów ADX na jakość klasyfikacji

Omówienie sposobu przygotowania danych syntetycznych "Artificial05_500" znajduje się w sekcji 7.1.3. Zbiór "oszusci2007_2" to dane na temat odbiorów nagród w programie VITAY z klasą charakteryzującą oszustwo reprezentowaną przez 3072 przypadki. Reszta zbiorów pochodzi z ogólnie dostępnych źródeł i jest szeroko wykorzystywana w różnych eksperymentach.

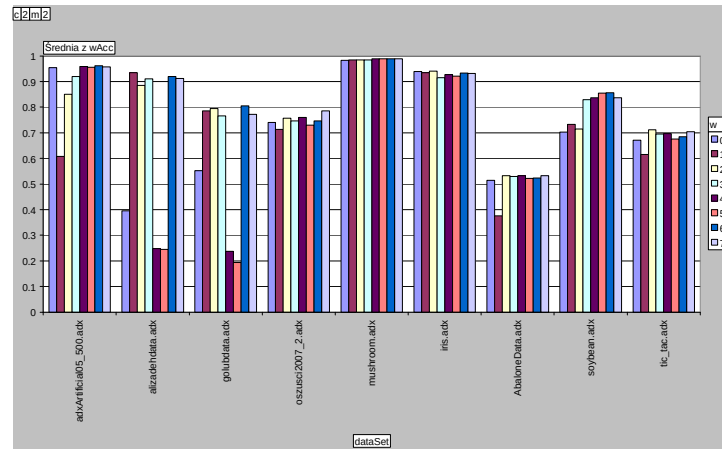
Badanie wpływu parametrów *scoreMethod*, *cleanCandidates*, *mergeCondition*

Przy ustalonych parametrach:

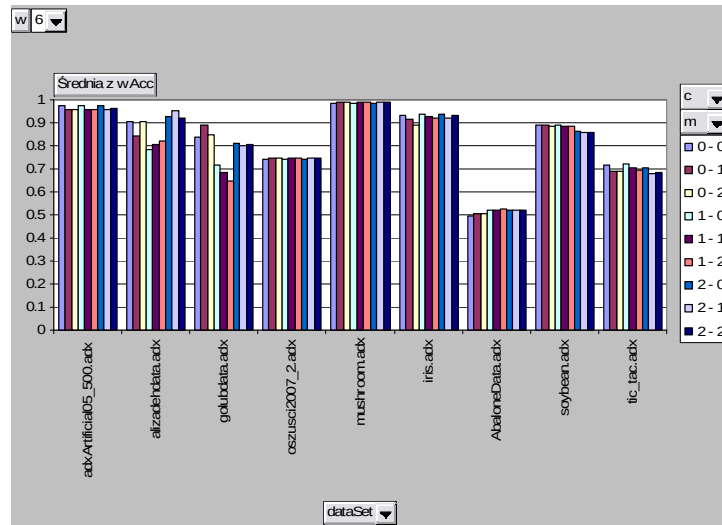
- *searchBeam* = 50
- *finalBeam* = [50]

- $qMethod = 1$
- $qMethodFinal = 2$
- $selSignificantMethod = 4$
- $maxEventsForSelection = 3000$
- $useSensitivity = 0$

zbadano wpływ na jakość klasyfikacji parametrów $scoreMethod$, $cleanCandidates$ oraz $mergeCondition$. Dla każdego ze zbiorów danych opisanych wcześniej w tabeli 7.1 i możliwych zestawów badanych parametrów przeprowadzono 10 kolejnych walidacji krzyżowych (każda oparta na podziale na 3 zbiory). Średni wynik Acc i $wAcc$ z każdej serii walidacji zapisano.



Rysunek 7.1. Średni wynik $wAcc$ dla różnych $scoreMethod$ przy ustalonych $cleanCandidates = 2$ i $mergeCondition = 2$



Rysunek 7.2. Średni wynik $wAcc$ dla ustalonego $scoreMethod = 6$

Analizując wyniki eksperymentów (wykresy 7.1 i 7.2) można stwierdzić, że nie ma optymalnego zestawu parametrów, który w przypadku wszystkich zbiorów pozwala uzyskać maksymalnie wysoką jakość klasyfikacji. Tak jak w

przypadku innych popularnych algorytmów można jednak arbitralnie wybrać pewien zestaw, który pozwala uzyskać wysoką jakość klasyfikacji. Wydaje się że przy ustawieniach $scoreMethod = 6$, $cleanCandidates = 2$ (usuwanie kandydatów którzy nie poprawili swojej jakości w stosunku do przynajmniej jednego z rodziców) i $mergeCondition = 0$ (faza łączenia kompleksów zablokowana) uzyskuje się najlepsze efekty i takie nastawy parametrów będą używane jako domyślne.

Pewnym zaskoczeniem mogą być uzyskane wyniki dla wyłączonej fazy sklejanie kompleksów. Reguły po sklejeniu pokrywają jednocześnie większy obszar rozpatrywanej przestrzeni, a tym samym mogą wpływać na lekkie rozmycie decyzji. Reguły które pokrywają mniejsze obszary są bardziej wyspecjalizowane i mogą dokładniej opisać zależności w danych. Po sklejeniu dwóch kompleksów nie mamy już do dyspozycji dwóch zestawów wskaźników (np. różnych prawdopodobieństw klasy pozytywnej dla dwóch odmiennych obszarów) dla dwóch reguł lecz jeden zestaw opisujący większy zsumowany obszar. Skuteczna metryka stosowana w metodzie dyskretyzacji pozwala na uzyskanie przedziałów wartości atrybutu na tyle odpowiednich by ponowne ich łączenie w fazie sklejanie nie pozwalało podnieść skuteczności klasyfikacji. Reasumując sklejanie wpływa na większe uogólnienie wiedzy natomiast przy jego braku uzyskujemy więcej dokładniejszych reguł a tym samym lepsze dopasowanie.

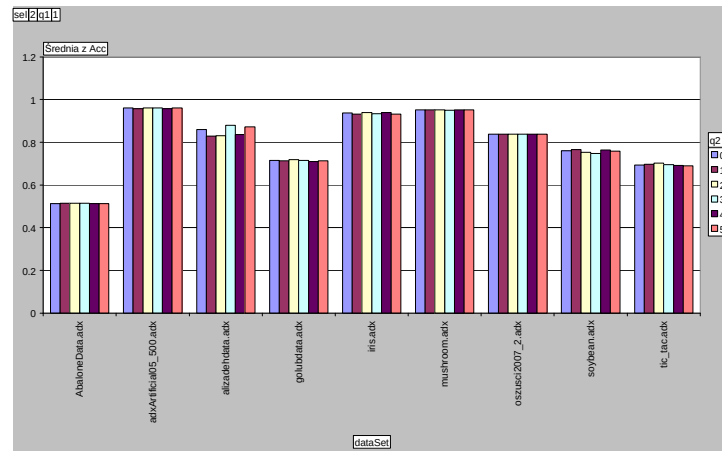
Badanie wpływu parametrów $qMethod$, $qMethodFinal$, $selSignificantMethod$

Przy ustalonych parametrach:

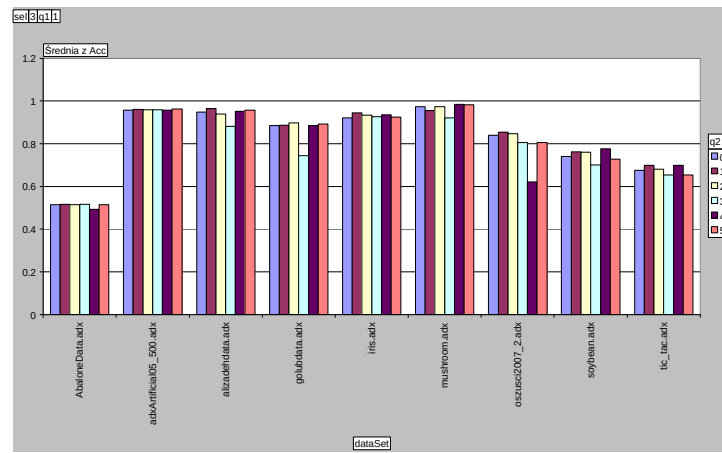
- $searchBeam = 50$
- $finalBeam = [50]$
- $qMin = 0.1$
- $scoreMethod = 6$
- $cleanCandidates = 2$
- $mergeCondition = 0$
- $maxEventsForSelection = 3000$
- $useSensitivity = 0$

Zbadano wpływ na jakość klasyfikacji parametrów $qMethod$, $qMethodFinal$ oraz $selSignificantMethod$. Dla każdego ze zbiorów danych opisanych wcześniej w tabeli 7.1 i możliwych zestawów badanych parametrów przeprowadzono 10 kolejnych walidacji krzyżowych (każda oparta na podziale na 3 zbiory). Średni wynik Acc i $wAcc$ z każdej serii takich walidacji zapisano.

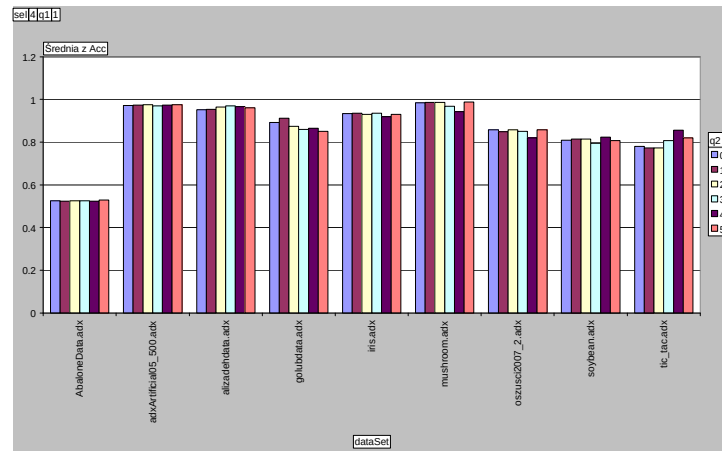
Dla wszystkich zbiorów zbadano wszystkie kombinacje trzech parametrów ($qMethod$, $qMethodFinal$, $selSignificantMethod$). Na rysunkach 7.3, 7.4 i 7.5 przedstawiono wyniki klasyfikacji dla ustalonych $qMethod$ i $selSignificantMethod$ w celu prezentacji wpływu metody oceny jakości Q kompleksu w fazie finalnego wyboru reguł. Kryteria wyboru finalnego zbioru kompleksów przy ustawieniu $selSignificantMethod = 1$ lub 2 nie uwzględniają $qMethodFinal$ dlatego nie ma potrzeby ich prezentacji. Wartość $qMethod = 1$ została tutaj wybrana arbitralnie lecz dla wszystkich trzech wykresów jej zmiana nie powoduje silnych zmian charakterystyki słupków.



Rysunek 7.3. Średni wynik $wAcc$ dla ustalonych $qMethod = 1$ oraz $selSignificantMethod = 2$

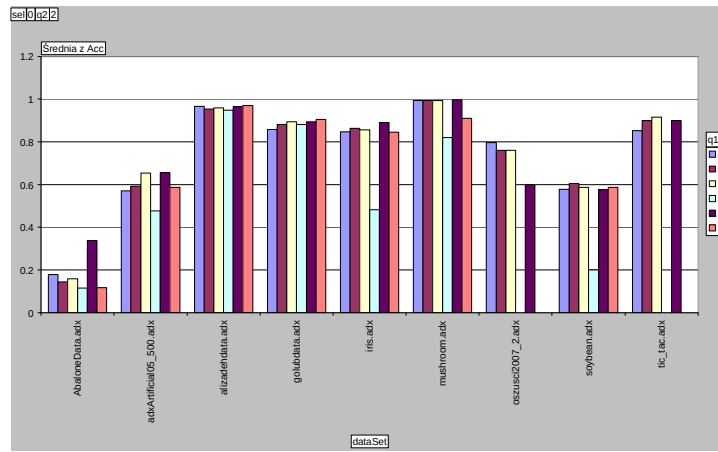


Rysunek 7.4. Średni wynik $wAcc$ dla ustalonych $qMethod = 1$ oraz $selSignificantMethod = 3$

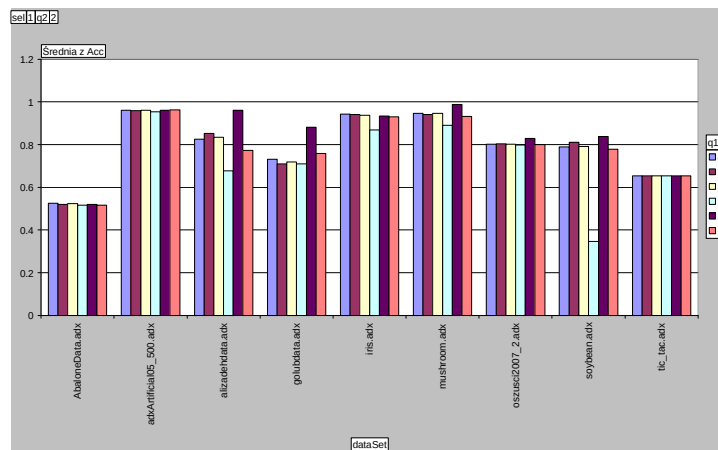


Rysunek 7.5. Średni wynik $wAcc$ dla ustalonych $qMethod = 1$ oraz $selSignificantMethod = 4$

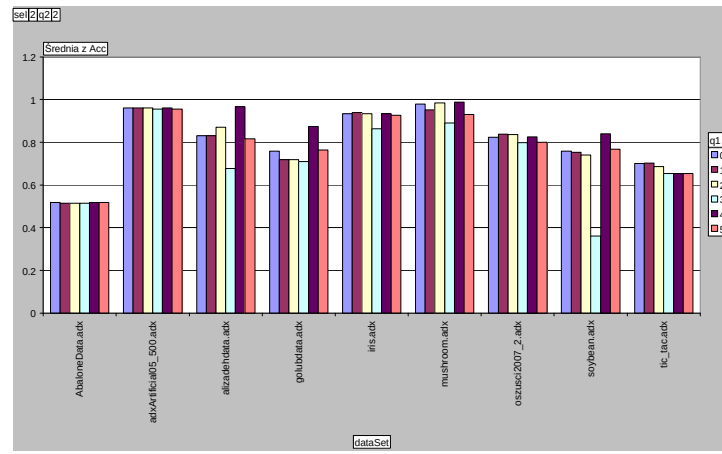
Wpływ miary $qMethodFinal$ jest zależny od metody wyboru finalnego zestawu $selSignificantMethod$. W przypadku najbardziej zaawansowanej miary, która skupia się na wyborze kompleksów najlepiej klasyfikujących podzbiór przykładów trenujących, wpływ $qMethodFinal$ jest jak widać znikomy – ogranicza się on jedynie do utworzenia rankingu kompleksów. Decyzja o wyborze w tej metodzie nie opiera się już na mierze Q . W metodach finalnej selekcji gdzie Q ma większe znaczenie można już zauważyć słabość prymitywnej miary oceny opartej jedynie na jednym z czynników $pcov$ lub $ncov$. Połączenie obu czynników, które ma na celu minimalizację $ncov$ i maksymalizację $pcov$ ($qMethodFinal = 0, 1, 2, 5$) powoduje już stabilizację wyniku w zależności od zbioru danych. Wobec bardzo zbliżonych wyników trudno jest dokonać wyboru najlepszej miary dla $qMethodFinal$, niemniej jednak autor dla dalszej części eksperymentów przyjął $qMethodFinal = 2$ jako ustawienie domyślne.



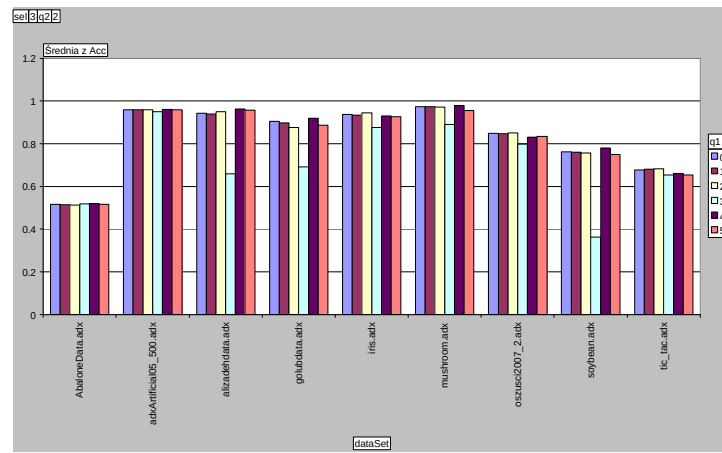
Rysunek 7.6. Średni wynik $wAcc$ dla ustalonych $qMethodFinal = 2$ oraz $selSignificantMethod = 0$



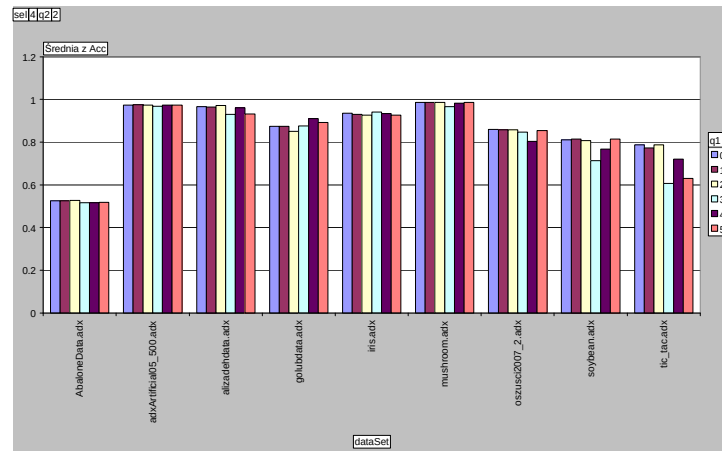
Rysunek 7.7. Średni wynik $wAcc$ dla ustalonych $qMethodFinal = 2$ oraz $selSignificantMethod = 1$



Rysunek 7.8. Średni wynik $wAcc$ dla ustalonych $qMethodFinal = 2$ oraz $selSignificantMethod = 2$

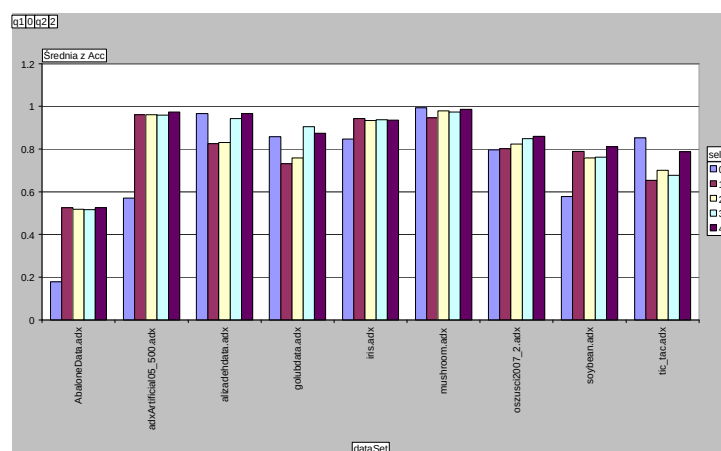


Rysunek 7.9. Średni wynik $wAcc$ dla ustalonych $qMethodFinal = 2$ oraz $selSignificantMethod = 3$

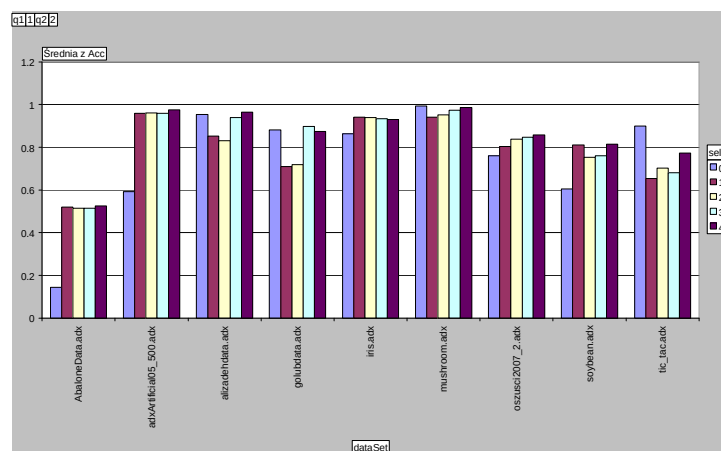


Rysunek 7.10. Średni wynik $wAcc$ dla ustalonych $qMethodFinal = 2$ oraz $selSignificantMethod = 4$

Na kolejnych rysunkach 7.6, 7.7, 7.8, 7.9, 7.10 przedstawiono wyniki klasyfikacji dla ustalonych $qMethodFinal = 2$ i odpowiednio parametru $selSignificantMethod$ w celu prezentacji wpływu metody oceny jakości Q w fazie budowy kompleksów. Najlepsze i najbardziej stabilne wyniki dostajemy w przypadku $selSignificantMethod = 3, 4$ i $qMethod = 0, 1, 2, 5$. Porównanie do najprostszej strategii gdzie w każdej fazie dbamy tylko o minimalizację $ncov$ ($qMethod = 4$ – minimalizacja $ncov$, $selSignificantMethod = 0$ – w finalnym zestawie pozostawiamy tylko kompleksy o zerowym $ncov$) pokazuje że bardziej wyrafinowane dają znacznie lepsze wyniki jakości klasyfikacji. Warto jednak zwrócić uwagę na ustawienie $qMethod = 4$, miara Q minimalizująca $ncov$ która w szczególności w prostszych metodach wyboru finalnego zestawu radzi sobie wyjątkowo dobrze. Wydaje się, że najkorzystniejszą miarą oceny Q w trakcie budowy kompleksów jest $qMethod = 0, 1$ ponieważ niezależnie od metody wyboru finalnego zestawu przeważnie uzyskuje najwyższe wyniki jakości klasyfikacji.



Rysunek 7.11. Średni wynik $wAcc$ dla ustalonych $qMethod = 0$ oraz $qMethodFinal = 2$



Rysunek 7.12. Średni wynik $wAcc$ dla ustalonych $qMethod = 1$ oraz $qMethodFinal = 2$

Dla ustalonych $qMethod = 0,1$ i $qMethodFinal = 2$ na rysunkach 7.11, 7.12 przedstawiono wyniki klasyfikacji w zależności od metody finalnego wyboru zestawu reguł. Najbardziej zaawansowana strategia wyboru $selSignificantMethod = 4$ pozwala na uzyskanie najlepszej klasyfikacji dla większości badanych zbiorów, w szczególności dla dość dużego zbioru rzeczywistych danych "oszusci2007_2" uzyskuje przewagę. Tym bardziej, że dla małych zbiorów gdzie zbiór kompleksów podlegający finalnej selekcji jest ograniczony niezależnie od metody mamy dużą szansę wyboru podobnego finalnego zestawu. Różnice w metodzie finalnej selekcji powinny bardziej uwidaczniać się (i tak się dzieje) dla dużych bardziej skomplikowanych zbiorów ponieważ zbudowanych i ocenionych kompleksów jest stosunkowo znacznie więcej. Warto zwrócić uwagę na zbiór "tic.tac.toe" gdzie przy $selSignificantMethod = 0$ uzyskuje się najbardziej prawidłową klasyfikację. W przypadku tego zbioru do najlepszej klasyfikacji potrzeba wszystkich pokrywających jedynie przykłady pozytywne. Po tej serii eksperymentów można uznać że parametrami domyślnymi algorytmu pozostaną $selSignificantMethod = 4$ i $qMethod = 1$, $qMethodFinal = 2$.

Badanie wpływu parametrów $searchBeam$, $finalBeam$

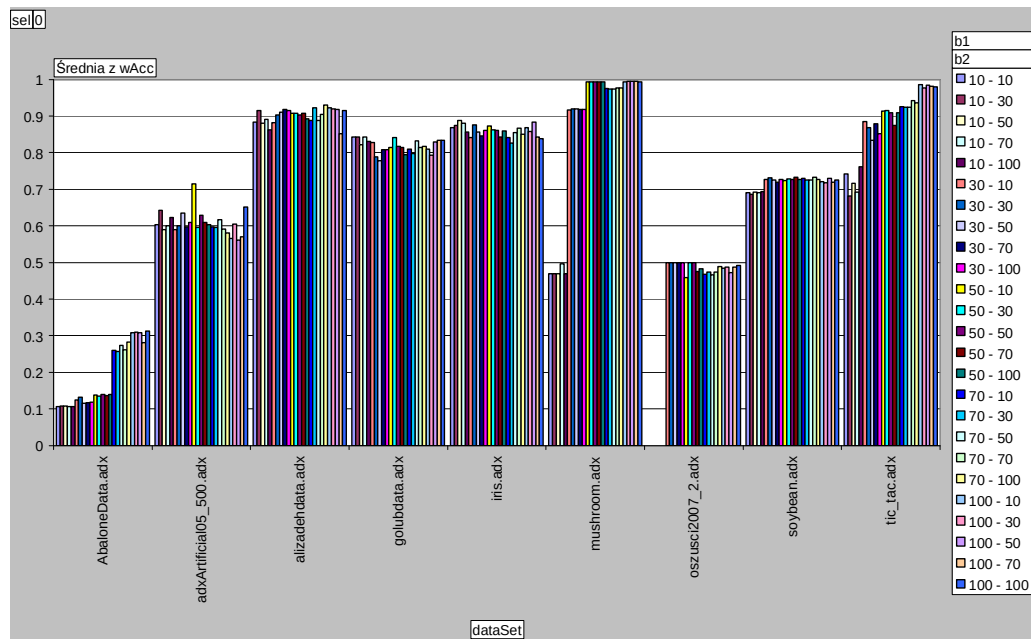
Przy ustalonych parametrach:

- $qMethod = 1$
- $qMethodFinal = 2$
- $qMin = 0.1$
- $scoreMethod = 6$
- $cleanCandidates = 2$
- $mergeCondition = 0$
- $maxEventsForSelection = 3000$
- $useSensitivity = 0$

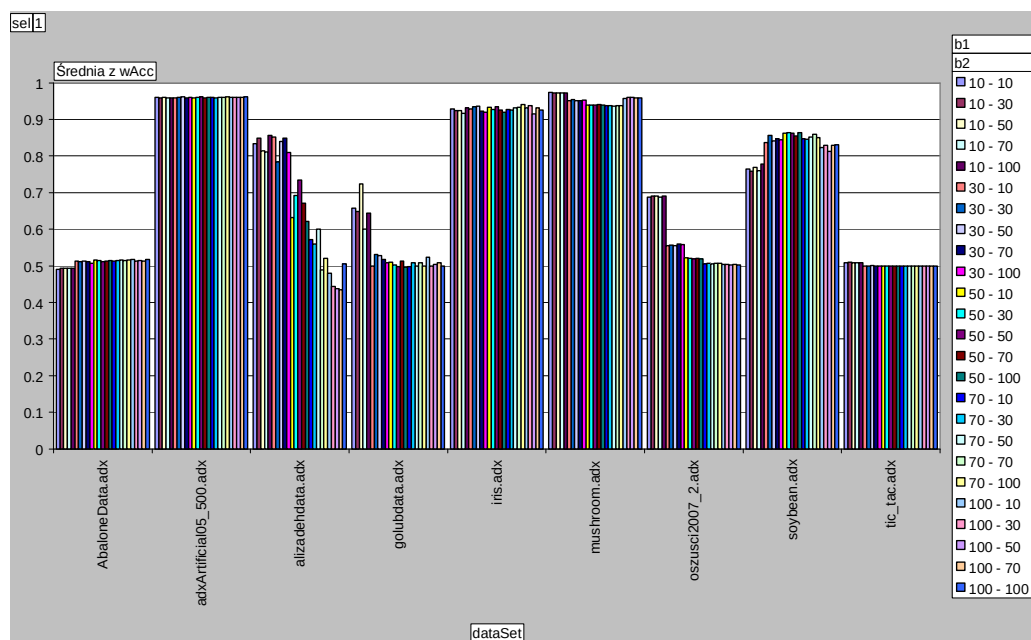
Zbadano wpływ na jakość klasyfikacji parametrów $searchBeam$, $finalBeam$ w zależności od $selSignificantMethod$. Podobnie jak poprzednio dla każdego ze zbiorów danych opisanych wcześniej w tabeli 7.1 i możliwych zestawów badanych parametrów przeprowadzono 10 kolejnych walidacji krzyżowych (każda oparta na podziale na 3 zbiory), a średni wynik Acc i $wAcc$ serii zapamiętano. Na rysunkach przedstawiono zależność jakości klasyfikacji od odpowiedniej pary wartości parametrów $searchBeam$ oraz $finalBeam$.

W przypadku metody selekcji $selSignificantMethod = 0$ w finalnym zbiorze reguł znajdują się wszystkie reguły o zerowym $ncov$. Na rysunku 7.13 zaprezentowane średnie $wAcc$ w zależności od $searchBeam$ i $finalBeam$. Użyta metoda wyboru reguł nie używa $finalBeam$ co ma dobre odzwierciedlenie na wykresie. Uwidacznia się natomiast przeważnie schodkowy wzrost $wAcc$ w zależności od zwiększania liczby budowanych kandydatów w trakcie budowy reguł. Dla tej metody selekcji finalnego zbioru wniosek jest dość oczywisty iż zwiększanie wiązki poszukiwań ($searchBeam$) pozwala na uzyskanie klasyfikacji na wyższym poziomie.

W przypadku gdy finalna selekcja opiera się na wybraniu tylko tych reguł, które przyjmują $pcov > ncov$ (rys. 7.14). Dla 3 zbiorów danych dla których można zbudować ogromną liczbę reguł zwiększanie $searchBeam$ pogarsza

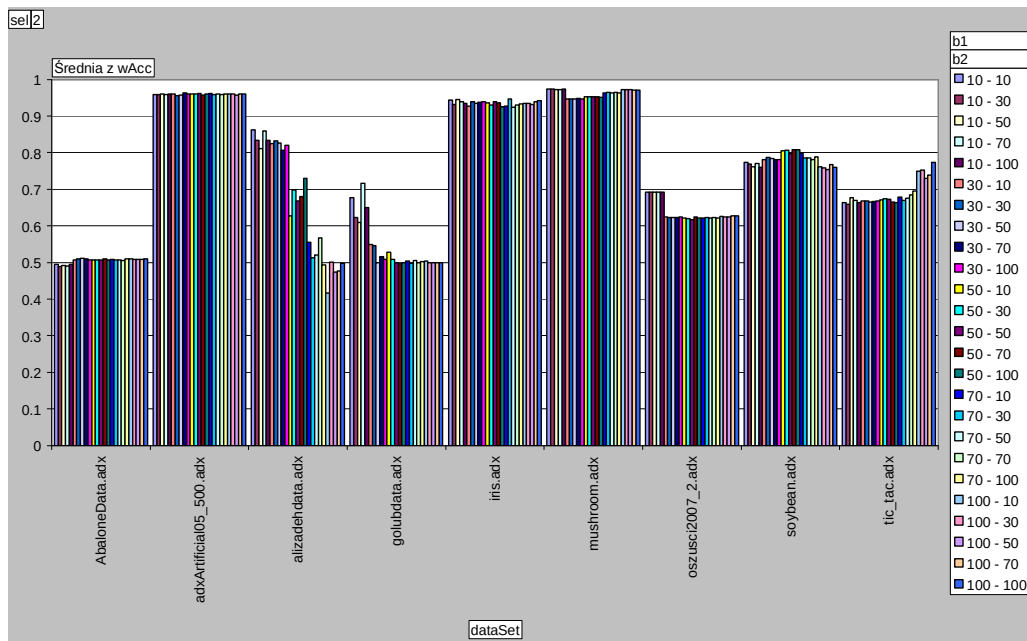


Rysunek 7.13. Średni wynik $wAcc$ dla ustalonego $selSignificantMethod = 0$ w zależności od $searchBeam$, $finalBeam$



Rysunek 7.14. Średni wynik $wAcc$ dla ustalonego $selSignificantMethod = 1$ w zależności od $searchBeam$, $finalBeam$

jakość klasyfikacji. Przy jednoczesnym bardzo liberalnym kryterium wyboru finalnego zestawu budowa dużej liczby reguł wpływa pośrednio na ich liczbę w zbiorze finalnym. Gdy budujemy więcej reguł, to więcej z nich spełnia kryterium $pcov > ncov$ i tym samym bierze później udział w klasyfikacji. Jak widać wpływa to bardzo niekorzystnie i wpływa na rozmycie podejmowanych decyzji.

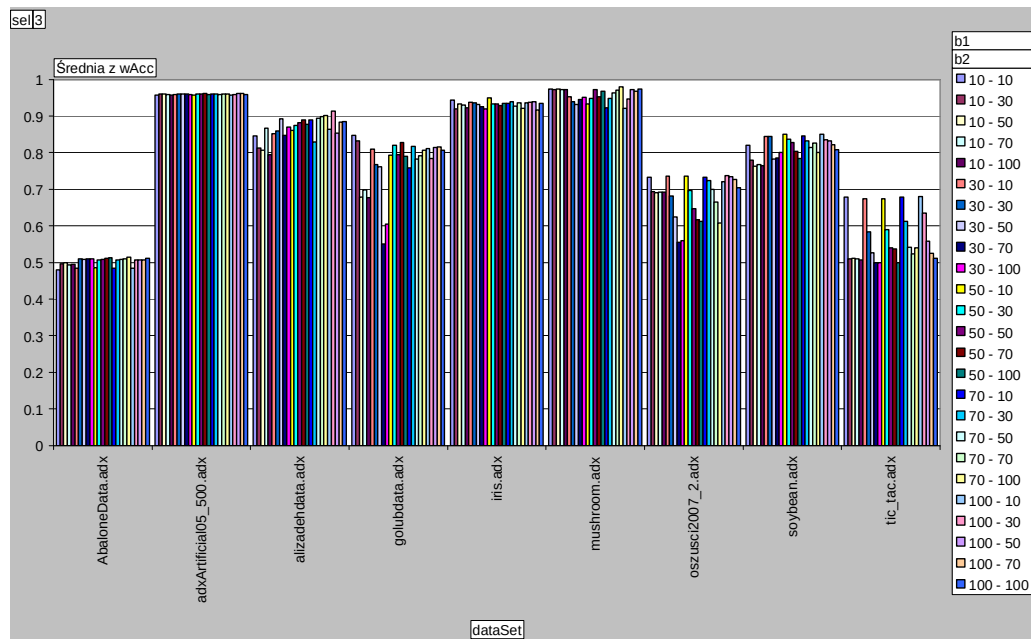


Rysunek 7.15. Średni wynik $wAcc$ dla ustalonego $selSignificantMethod = 2$ w zależności od $searchBeam$, $finalBeam$

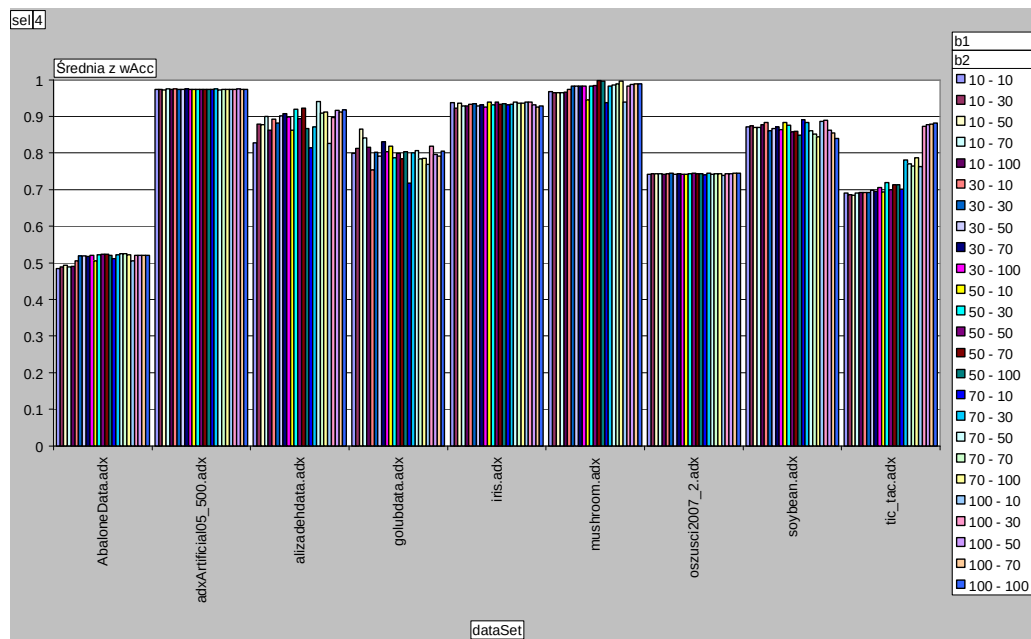
Gdy do finalnego zbioru trafiają reguły ($selSignificantMethod = 2$, rys. 7.15) o Q mniejszym niż założone sytuacja jest podobna jak poprzednio. Wyznaczony próg odcięcia nie jest ustawiony restrykcyjnie (należałoby go dobierać eksperymentalnie) co znów powoduje spadek jakości klasyfikacji przy zwiększaniu $searchBeam$. Odwrotna bardzo ciekawa tendencja uwiadamia się w przypadku zbioru "tic_tac_toe", który wymaga nieco odmiennej strategii parametrów niż reszta zbiorów (szerzej o tym w komentarzu odnośnie rys. 7.17).

Dla $selSignificantMethod = 3$ (wybieramy ściśle określoną liczbę finalnych reguł najlepszych pod względem Q) oba parametry $searchBeam$ i $finalBeam$ mają już poważne znaczenie. Dla kilku zbiorów na wykresie można zaobserwować swoiste "piki" które związane są z odpowiednią wartością $searchBeam$ bądź $finalBeam$. Generalnie znów lepsze wyniki klasyfikacji osiąga się gdy finalny zbiór reguł zawiera ich raczej mniej (w przypadku "tic_tac_toe" wyraźnie widać szpilki dla $finalBeam = 10$). Co interesujące w przypadku tej metody selekcji właściwie nie widać zmiany w jakości klasyfikacji w zależności od badanego przedziału $searchBeam$.

Porównując metodę wyboru finalnego zestawu ($selSignificantMethod = 4$, rys. 7.17) z poprzednimi widać zdecydowanie lepsze wyniki klasyfikacji oraz ich większą stabilność. Parametr $finalBeam$ dla tej metody jedynie ogranicza maksymalną liczbę reguł w finalnym zbiorze, natomiast to metoda już decyduje czy będzie ich mniej. Właściwie nawet dla niewielkiej wartości $searchBeam$ i $finalBeam$ metoda jest w stanie wybrać najlepsze dostępne reguły do poprawnej klasyfikacji. Przy stosunkowo znacznie większym obszarze poszukiwań reguł $searchBeam = 100$ dla większości badanych zbiorów nie uwiadamia się dramatyczny wzrost jakości klasyfikacji. Jedynie dla zbioru



Rysunek 7.16. Średni wynik $wAcc$ dla ustalonego $selSignificantMethod = 3$ w zależności od $searchBeam$, $finalBeam$



Rysunek 7.17. Średni wynik $wAcc$ dla ustalonego $selSignificantMethod = 4$ w zależności od $searchBeam$, $finalBeam$

”tic_tac_toe” uwidacznia się wyraźna tendencja wzrostu poprawności klasyfikacji wraz ze zwiększaniem wiązki poszukiwań.

Obserwowane różnicowanie wyników dla zbiorów reprezentujących ekspresje genów jest związane z dwoma aspektami. Można zauważyć, że dla niewielkich wartości $finalBeam$ klasyfikacja raczej wypada gorzej, co jest związane z niedostatecznym zasobem wiedzy - pełniejszy opis pozwala na

lepszą klasyfikację. Drugim elementem, który wprowadza lekką niestabilność wyników to fakt, iż dla tak nietypowego zbioru dysponujemy dużą liczbą selektorów o najwyższej jakości, która przeważnie przekracza *searchBeam*. Wtedy zgodnie z algorytmem losuje się najlepsze z dostępnych selektorów by zostały rodzicami w kolejnej fazie.

Eksperymenty wykazały, że metoda wyboru finalnego zestawu kompleksów ma niezwykle silny wpływ na klasyfikację i zgodnie z oczekiwaniami metoda *selSignificantMethod* = 4 okazała się najlepsza dla badanych zbiorów. Oba parametry *searchBeam* i *finalBeam* mają bardzo duży wpływ na czas uczenia (zostanie przedstawione to w kolejnej sekcji) i przy wyborze ich wartości domyślnych należy o tym pamiętać. Zwiększenie *searchBeam* zwiększa szanse znalezienia lepszych selektorów, natomiast zwiększanie *finalBeam* daje większą swobodę metodzie selekcji do wyboru najwłaściwszego finalnego zestawu.

Wybór wartości domyślnych dla ADX to kompromis pomiędzy oczekiwaną jakością klasyfikacji a czasem uczenia dla nieznanego zbioru danych. Autor proponuje *searchBeam* = 50 i *finalBeam* = 50 lecz przykład zbioru "tic_tac_toe" pokazuje, że w drugim kroku warto zbadać większe wartości pamiętając o potrzebnym dłuższym czasie uczenia.

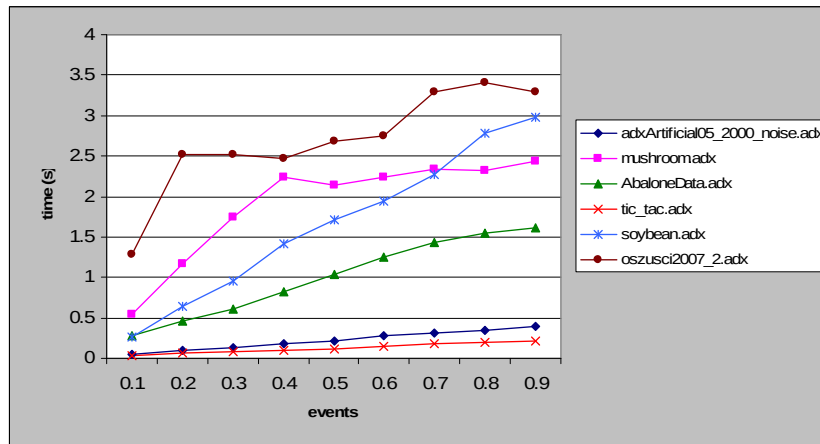
7.1.2. Badanie złożoności obliczeniowej

W tej sekcji zostaną przedstawione wyniki eksperymentów pomiaru czasu uczenia w zależności od ilości przykładów, atrybutów, oraz parametru *searchBeam*. Eksperymenty przeprowadzono przy domyślnych ustawieniach algorytmu ADX na komputerze którego parametry omówiono we wstępie sekcji:

- *searchBeam* = 50
- *finalBeam* = [50]
- *qMethod* = 1
- *qMethodFinal* = 2
- *scoreMethod* = 6
- *cleanCandidates* = 2
- *mergeCondition* = 0
- *maxEventsForSelection* = 3000
- *useSensitivity* = 0

Czas uczenia prezentowany na rys. 7.18 uwzględnia jedynie etap indukcji reguł i ich finalnej selekcji bez wstępnej dyskretyzacji atrybutów ciągłych. Używany algorytm dyskretyzacji (metoda zaproponowana przez Faiada i Iraniego) został reimplementowany na podstawie [Fayyad *et al.*, 1992] gdzie przedstawiono również wyniki badań jego właściwości.

W przypadku badania zależności czasu uczenia od ilości przypadków dla każdego zbioru danych losowano odpowiednią część przykładów wielokrotnie (10 razy) a następnie wynik uśredniano. Poszczególne wyniki mogą różnić się lekko od siebie co jest głównie związane z różnymi próbkami danych oraz w mniejszym stopniu z działającymi równolegle innymi procesami na komputerze badawczym.



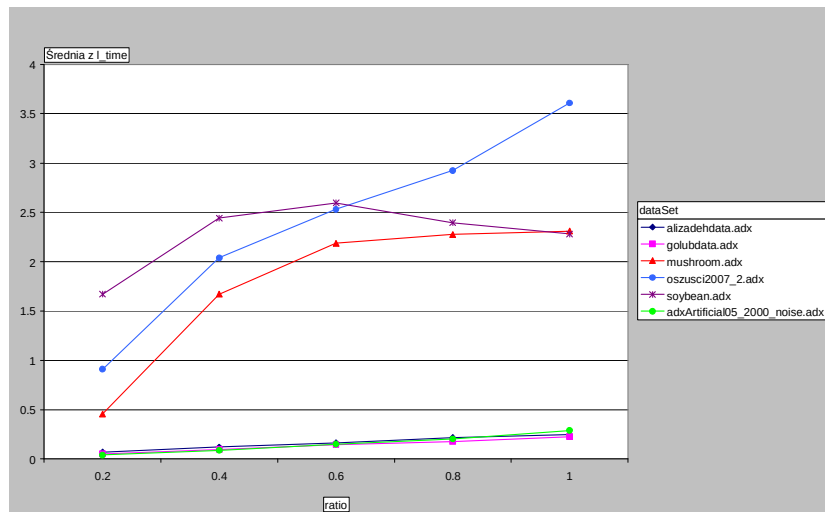
Rysunek 7.18. Średni czas uczenia ADX w zależności od liczby przykładów

Dla badanych zbiorów zależność czasu uczenia od ilości przykładów ma przeważnie liniowy charakter. Dla większych zbiorów wykres jest raczej liniowy przedziałami z zauważalnym mniejszym nachyleniem (nasyceniem) przebiegu. Nasycenie to jest związane z ustawieniami parametrów ograniczającymi nadmierną budowę kompleksów kandydatów (*searchBeam*) oraz proces selekcji finalnego zestawu (*finalBeam*, *maxEventsForSelection*).

W przypadku drugiej serii eksperymentów zbadano zależność czasu uczenia algorytmu ADX od wielkości zbioru dostępnych atrybutów. Dla każdego ze zbiorów 30 razy losowano odpowiedni podzbiór atrybutów, a następnie trzykrotnie przeprowadzono etap uczenia na losowo wybranych 2/3 przykładów i testowano na pozostałej 1/3. Wielkość kolejnych losowanych podzbiorów atrybutów ustalono na odpowiednio [0.2, 0.4, 0.6, 0.8, 1.0] całości dostępnych cech. Na rysunku 7.19 przedstawiono uśredniony czas uczenia (z 30 powtórzeń) dla badanych zbiorów, które zawierały wystarczającą łączną liczbę atrybutów do przeprowadzenia niniejszego eksperymentu. Liczba powtórzeń użyta w tym eksperymencie jest większa ponieważ starano się zmniejszyć wpływ losowego doboru atrybutów. Dwie realizacje uczenia oparte na różnych zbiorach tej samej liczby atrybutów mogą dać odmienne wyniki. Jest to konsekwencją zróżnicowania informatywności poszczególnych cech w zbiorach danych.

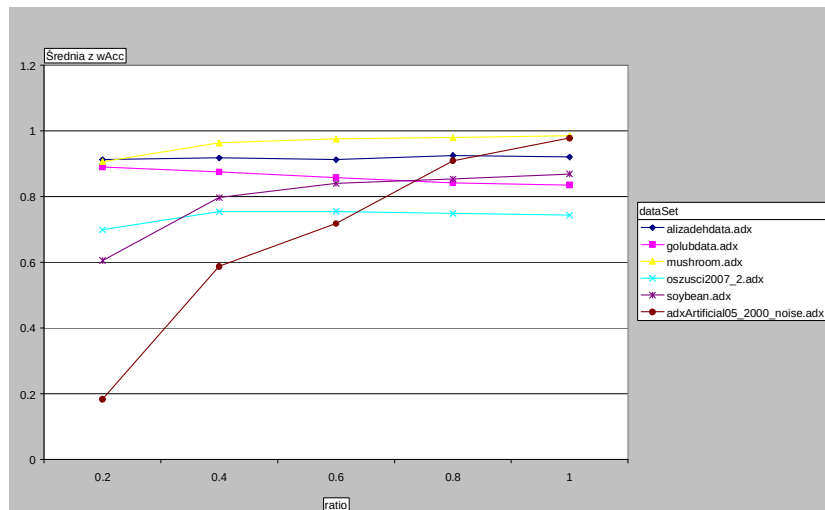
Zbiór "adxArtificial05_2000_noise" to dane syntetyczne o 30 atrybutach i 16 tysiącach przykładów reprezentujących dwie klasy. Sposób jego generacji został opisany w sekcji 7.1.3.

W przypadku większości badanych zbiorów danych średni czas uczenia rośnie liniowo wraz ze wzrostem ilości atrybutów co jest zgodne z założeniami algorytmu. W przypadku dwóch ("soybean" i "mushroom") spośród badanych zbiorów czas uczenia przestaje rosnąć lub nawet maleje w punkcie gdzie dostępnych jest 0.6 spośród wszystkich atrybutów. Dokładniejsza analiza poszczególnych przypadków pozwoliła na wyjaśnienie tego fenomenu. Przy mniejszej liczbie dostępnych atrybutów budowane zbiory kandydatów dla dłuższych reguł są większe ponieważ w ramach najwyżej ocenionych kompleksów (ograniczonych przez *searchBeam*) łatwiej jest znaleźć rodzi-



Rysunek 7.19. Średni czas uczenia ADX w zależności od liczby atrybutów

ców spełniających kryteria tworzenia nowego kandydata. W tym przypadku więcej jest dostępnych kompleksów o wystarczającej części wspólnej pozwalającej na wygenerowanie nowego kandydata. W przypadku większej ilości atrybutów często najwyższej oceniony zbiór kompleksów jest na tyle różnorodny że faza budowy kompleksów (w każdym kroku dłuższych o 1) kończy się znacznie wcześniej.



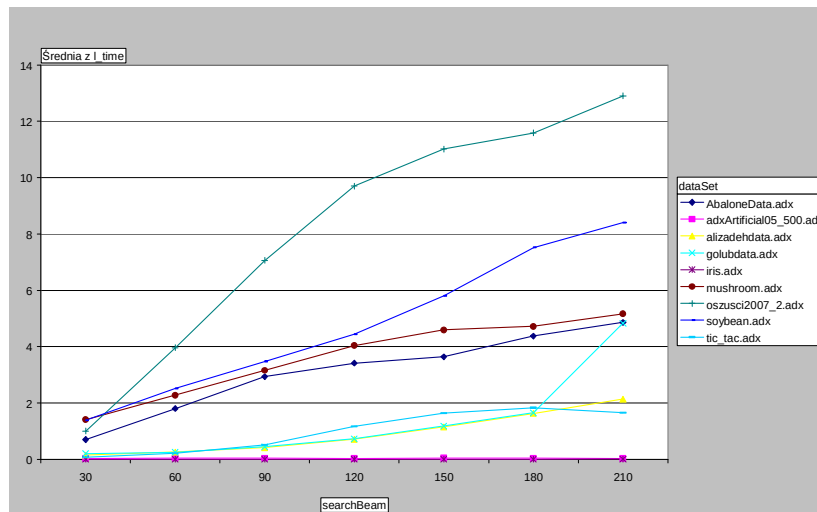
Rysunek 7.20. Średnie wAcc klasyfikacji ADX w zależności od liczby atrybutów

Mówiąc ogólniej, gdy dostępnych atrybutów wejściowych jest mniej (trudniejsze zadanie), wtedy algorytm dokonuje sprawdzenia większej przestrzeni możliwych reguł co pozwala mu na lepsze dostosowanie się do napotkanego problemu. W przeprowadzonym eksperymencie przy mniejszej liczbie atrybutów należy spodziewać się raczej trudniejszego zadania z punktu widzenia klasyfikacji ponieważ selekcja cech ma tutaj charakter czysto losowy. Na rysunku 7.20 przedstawiono średnie wAcc dla eksperymentu z rysunku 7.19.

Dla zbiorów danych "soybean" i "mushroom" mimo nieliniowej charakterystyki czasu uczenia od ilości atrybutów, jakość predykcji powyżej wartości 0.6 w zasadzie nie ulega zmianie. Oznacza to, iż dla tych dwóch zbiorów mimo pewnego utrudnienia, w postaci usunięcia części atrybutów, jakość klasyfikacji nie ulega zmianie przy nieco dłuższym czasie uczenia.

Ostatnia seria eksperymentów w tej sekcji ma na celu wykreślenie zależności czasu uczenia od wartości parametru *searchBeam* (rys. 7.21). Tym razem również użyto domyślnego zestawu parametrów ADX i 10-krotnie przeprowadzono trenowanie i testowanie na każdym ze zbiorów. Wartości na wykresie oznaczają średni czas indukcji reguł przeprowadzonej na losowej części 2/3 całego zbioru danych.

Z przytoczonego opisu algorytmu ADX można się spodziewać, iż w najgorszym przypadku przy *searchBeam* = 50 należy za każdym razem zbadać 1225 możliwych wygenerowanych kompleksów kandydatów. Liczba ta wynika z ilości możliwych kombinacji (symbol Newtona: $\binom{50}{2}$) przy założeniu budowy spełniania wszystkich kryteriów łączenia kompleksów rodziców. W praktyce jednak taka sytuacja zachodzi rzadko i charakterystyka ma dla większości zbadanych zbiorów raczej liniowy przebieg, niż krzywej odpowiadającej funkcji $\binom{x}{2}$. Kryteria doboru rodziców do budowy kompleksu kandydata powodują znaczne ograniczenie tworzenia dużej liczby niepotrzebnych kandydatów w każdej z kolejnych iteracji. Przeważnie zbiór kandydatów o długości 2 jest największy lecz również w tym wypadku praktycznie nie występuje potrzeba oceny wszystkich możliwych par, ponieważ zabrania się budowy kandydatów w oparciu o rodziców, którzy bazują na tym samym atrybucie.

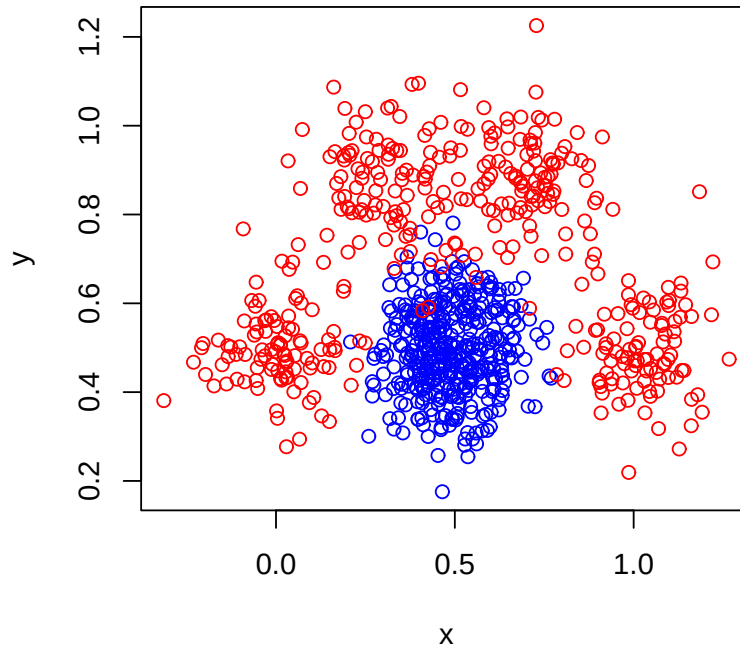


Rysunek 7.21. Średni czas uczenia ADX w zależności od parametru *searchBeam*

Podsumowując, czas indukcji reguł w algorytmie ADX skaluje się liniowo w zależności od liczby atrybutów i przykładów użytych w fazie trenowania. Zbadany czas uczenia jest w praktyce przeważnie funkcją liniową parametru *searchBeam*, a jak pokazały inne eksperymenty *searchBeam* = 50 jest bardzo dobrą wartością domyślną i zwiększanie jej podnosi jakość klasyfikacji tylko w przypadku wybranych zbiorów danych.

7.1.3. Klasyfikacja danych syntetycznych

Do przeprowadzenia eksperymentu przygotowano w środowisku **R** [R-project] w wersji 2.4.1 dane syntetyczne zawierające $n = 800, 4000, 8000, 16000$ przykładów podzielonych na dwie równej wielkości klasy.



Rysunek 7.22. Wizualizacja wygenerowanych danych syntetycznych dla zbioru $n = 800$

Każdy obiekt został opisany przez trzy atrybuty: dwa numeryczne wyznaczające jego położenie w przestrzeni 2D oraz trzeci opisujący klasę (*blue*, *red*). Wartości x i y dla klasy *blue*, zawierającej $\frac{n}{2}$ przykładów, zostały wygenerowane przy użyciu rozkładu normalnego dla średniej 0.5 i odchylenia standardowego 0.1, natomiast przykłady klasy *red* to cztery podobne rozkłady (każdy po $\frac{n}{8}$ przykładów) tak rozmieszczone by wspólnie otaczały klasę *red*. Wizualizacja tak przygotowanego zbioru dla $n = 800$ znajduje się na rysunku 7.22.

Reguły dla zbioru $n = 800$ zbudowane przez ADX na wszystkich danych są następujące:

```
#Rules for decision = blue
#Complex Size: 1
#Complexes Number: 0
#Complex Size: 2
#Complexes Number: 4
x=(0.25748622;0.35888457] and y=(-Infinity;0.7059473] p: 0.087
```

```

n: 0.002 q: 0.0847 pr: 0.9722
x=(0.35888457;0.6413076] and y=(-Infinity;0.7059473] p: 0.832
n: 0.012 q: 0.8097 pr: 0.9852
x=(0.6413076;0.7087706] and y=(-Infinity;0.7059473] p: 0.054
n: 0.002 q: 0.0523 pr: 0.9565
x=(0.7087706;0.7711703] and y=(-Infinity;0.7059473] p: 0.014
n: 0.002 q: 0.0124 pr: 0.8571

#Rules for decision = red
#Complex Size: 1
#Complexes Number: 3
x=(-Infinity;0.25748622]p: 0.340 n: 0.002 q: 0.3366 pr: 0.9927
x=(0.7711703;Infinity] p: 0.302 n: 0.0 q: 0.3025 pr: 1.0
y=(0.7059473;Infinity] p: 0.492 n: 0.007 q: 0.4813 pr: 0.9850
#Complex Size: 2
#Complexes Number: 4
x=(0.25748622;0.35888457] and y=(0.7059473;Infinity] p: 0.072
n: 0.0 q: 0.0724 pr: 1.0
x=(0.6413076;0.7087706] and y=(0.7059473;Infinity] p: 0.052
n: 0.0 q: 0.0524 pr: 1.0
x=(0.7087706;0.7711703] and y=(0.7059473;Infinity] p: 0.072
n: 0.0 q: 0.0724 pr: 1.0
x=(0.7711703;Infinity] and y=(-Infinity;0.7059473] p: 0.242
n: 0.0 q: 0.2425 pr: 1.0

```

Wartości: p, n opisujące reguły oznaczają odpowiednio pokrycia pozytywne, negatywne, q jakość reguły, natomiast pr prawdopodobieństwo wystąpienia rozpatrywanej klasy. Warto tutaj zaznaczyć, że reguły zostały wygenerowane w oparciu o domyślny zestaw parametrów co oznacza wyłączenie sklejanie kompleksów. Krótka analiza wygenerowanych reguł pozwoli zauważyć, że w obu zestawach reguł sklejenie spowodowałoby redukcję liczby reguł: dla decyzji *blue* do jednej, natomiast dla decyzji *red* do łącznie sześciu. Wyłączenie sklejanie pozwala jednak na otrzymanie kompleksów bardziej specyficznych pokrywających mniejsze przestrzenie co okazuje się lepsze rezultaty. Sklejanie wpływa jednak niekorzystnie na jakość klasyfikacji poprzez rozmywanie decyzji związanej z pokryciem przez poszczególne kompleksy.

Dla każdego z podanych w tabeli 7.2 algorytmów przeprowadzono 10-krotnie walidację krzyżową (3 zbiorową), a następnie uśredniony wynik zapisano. Wszystkie implementacje algorytmów porównawczych pochodzą z [Weka 3] w wersji 3.4.11 i uruchomiono je przy ustawieniach domyślnych w **R** przy wykorzystaniu pakietu RWeka.

Prawie każdy z badanych algorytmów radzi sobie bardzo dobrze z problemem a różnice w jakości klasyfikacji są minimalne, jedynie SVM w implementacji WEKA nie jest skuteczne dla tego zbioru. Zadanie nie powinno właściwie sprawiać żadnego problemu klasyfikatorowi SVM lecz przy wartościach domyślnych parametrów w zastosowanej implementacji nie udało się uzyskać lepszego rezultatu. Czas potrzebny na przeprowadzenie pełnego

n	J48	1-NN	NaiveBayes	RandomForest	SVM	ADX
800	97.9	96.8	96.7	97.6	73.5	97.4
time(s)	0.19	0.37	0.14	0.51	0.51	0.08
4000	97.6	97.0	96.0	97.3	72.6	97.3
time(s)	0.62	3.08	0.31	2.14	1.05	0.44
8000	97.7	97.0	96.6	97.2	72.6	98.0
time(s)	1.3	12.3	0.5	5.1	2.2	1.16
16000	97.6	96.7	96.3	NaN	72.8	97.8
time(s)	2.9	56.9	1.4	32.3	4.8	3.74

Tablica 7.2. Accuracy i czas pojedynczej *cv* dla zbioru danych syntetycznych

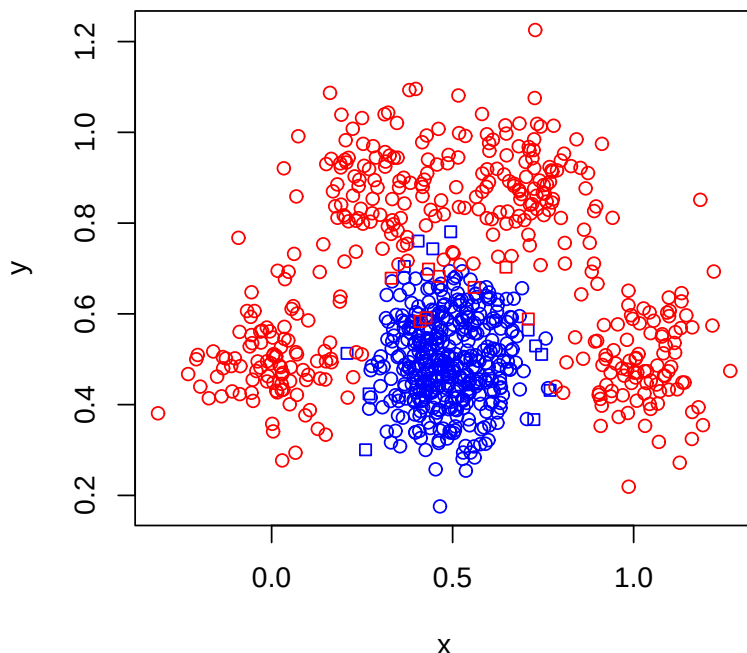
eksperymentu 3 zbiorowej walidacji krzyżowej jest bardziej zróżnicowany i tutaj algorytm ADX wyróżnia się pod względem efektywności. Czas mierzony tutaj obejmuje jeden pełen process walidacji krzyżowej łącznie z podziałem zbioru przykładów, uczeniem, klasyfikacją oraz ewentualną dyskretyzacją. Na rysunku 7.23 przedstawiono wynik klasyfikacji jednej z przeprowadzonych walidacji krzyżowych dla $n = 800$, przy czym kształt obiektu charakteryzuje poprawność nadania etykiety.

W kolejnej porcji eksperymentów każdy z przygotowanych wcześniej zbiorów został zakłócony poprzez dodanie 30 atrybutów z wartościami o rozkładzie równomiernym z zakresu $[0;1]$. Przeprowadzenie klasyfikacji w takich warunkach powinno dostarczyć informację czy ADX jest wrażliwy na nieinformatywne atrybuty oraz w jakim stopniu te atrybuty ingerują w budowane reguły. Tabela 7.3 zawiera wyniki eksperymentów klasyfikacji dla zaburzonych danych - eksperymenty przeprowadzono w identycznych warunkach jak poprzednio (uśredniony wynik 10 walidacji krzyżowych 3 zbiorowych).

Dodanie nowych kolumn silnie zakłóciło jakość klasyfikacji jedynie w przypadku k-NN. Jest to głównie związane z charakterem klasyfikatora, który znajduje najbliższych sąsiadów na podstawie wszystkich dostępnych cech, a wiemy że ponad 90% z nich ma charakter czysto losowy i nie związany z etykietą klasy. W przypadku ADX jakość klasyfikacji pozostała niemal identyczna co łatwo wytłumaczyć analizując poszczególne zestawy reguł. W żadnym z nich nie użyto zbędnych atrybutów a drobne zmiany średniego *accuracy* wynikają jedynie z losowości walidacji krzyżowej.

7.1.4. Klasyfikacja danych rzeczywistych

Ostatni z eksperymentów badających właściwości predykcji ADX poświęcono danym rzeczywistym, spośród których sporą grupę stanowią dane ogólnie znane z zasobów UCI [UCI Repository]. Dodatkowo dzięki udziałowi autora w praktycznych zastosowaniach metod data mining przeprowadzono testy przy użyciu zbiorów danych komercyjnych. W tej serii eksperymentów



Rysunek 7.23. Wynik walidacji krzyżowej dla zbioru $n = 800$, symbol $\square$ oznacza niepoprawną klasyfikację

oprócz zbiorów przedstawionych w tabeli na początku rozdziału (tabela 7.1) dodatkowo użyto danych przedstawionych w tabeli 7.4.

Dane "BP" określają wystąpienie niepożądanego piasku w trakcie prac eksploatacyjnych studni wydobywczej ropy lub gazu. Zjawisko to jest po części związane z zastanymi warunkami geologicznymi, choć może zostać zneutralizowane przy użyciu odpowiednio przeprowadzonego procesu odwiertu i wydobywania. Przykładów kiedy zaobserwowano wystąpienie piasku jest w danych stosunkowo niewiele (14 przykładów), co stanowi dodatkowe utrudnienie z racji silnego zróżnicowania wielkości klas.

Dane "Geant" to próba klientów hipermarketu podzielonych na 6 segmentów o wielkości od 919 do 7598. Każdy klient został opisany przy użyciu cech demograficznych jak również cech charakteryzujących ich przyzwyczajenia zakupowe (np. udział zakupów w weekend czy udział produktów dla dzieci w profilu). Zadaniem klasyfikatora jest poprawna klasyfikacja poszczególnych klientów do odpowiedniego segmentu/grupy.

Podobnie jak w sekcji 7.1.3 działanie algorytmu ADX przedstawiono na tle innych popularnych metod klasyfikacji, które również zbadano w oparciu o te same dane rzeczywiste. Każdy ze zbiorów danych poddano 10 krotnie procesowi walidacji krzyżowej 3 zbiorowej dla każdego z algorytmów, a następnie wyniki uśredniono. Implementacje klasyfikatorów: Naive Bayes (NB), drzewa C4.5 (J48), Random Forest (RF), SVM (SMO), kNN (IBk) pochodzą z pakietu WEKA w wersji 3.4.11, które uruchomiono przy użyciu domyślnych

n	J48	1-NN	NaiveBayes	RandomForest	SVM	ADX
800	97.4	56.8	95.7	94.3	70.3	97.3
time(s)	0.9	1.9	0.6	2.4	2.4	0.36
4000	97.1	54.2	95.9	97.0	72.1	97.3
time(s)	5.9	33.3	3.2	16.2	28.7	4.05
8000	97.1	56.7	96.5	97.4	72.3	98.0
time(s)	13.7	122.2	6.4	36.9	134.2	13.69
16000	97.1	55.8	96.3	97.4	72.6	97.8
time(s)	30.2	438.2	12.5	82.4	604.2	50.21

Tablica 7.3. Accuracy i czas pojedynczej *cv* dla zbioru danych syntetycznych zaburzonych losowymi cechami

zbiór	l. obiektów	l. atrybutów	l. klas	źródło
BP	134	32	2	BP
hypothyroid	3772	30	4	[UCI Repository]
ionosphere	351	35	2	[UCI Repository]
splice.asx	3190	61	3	[UCI Repository]
vote	435	17	2	[UCI Repository]
zoo	101	18	7	[UCI Repository]
Geant	16920	59	6	Géant Polska

Tablica 7.4. Dodatkowe zbiory danych użyte do porównania klasyfikatorów

parametrów (w przypadku kNN jest to 1NN). Autorska implementacja ADX również była testowana przy użyciu stałych domyślnych parametrów wypracowanych w ramach uprzednio prezentowanych eksperymentów. Implementacja algorytmu LEM2 pochodzi z pakietu RSES ([RSES], [Bazan *et al.*, 2002]) w wersji 2.2.2, który podobnie jak pozostałe algorytmy zrealizowano w języku JAVA. W przypadku LEM2 w systemie RSES brakuje raportowania czasu działania algorytmu LEM2, dlatego czas ten szacowano przy pomocy oprogramowania systemu operacyjnego Windows XP monitorującego obciążenie procesora (perfmon.exe).

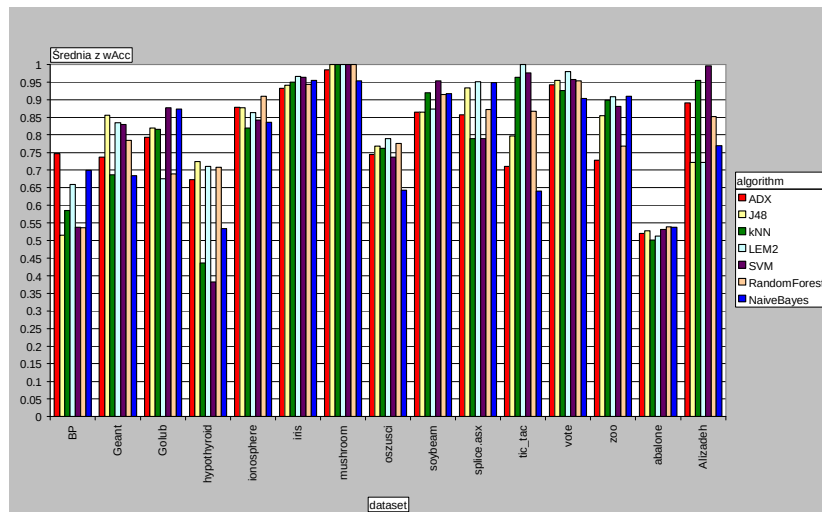
Uśrednione wyniki czasu i jakości klasyfikacji zebrano w tabeli 7.5. Czas podany w zestawieniu to średni czas pojedynczego procesu walidacji krzyżowej przeprowadzonej przy podziale na 3 równej wielkości podzbiory. Czas ten łączy w sobie czas ewentualnej dyskretyzacji, uczenia i testowania, co wydaje się porównaniem możliwie sprawiedliwym ponieważ zbiera wszystkie elementy działania klasyfikatora.

Jakość klasyfikacji (patrz wykres 7.24 i tabela 7.5) osiągnięta przez klasyfikator ADX jest porównywalna z zasadniczymi technikami używanymi w

dataset	wskaźnik	ADX	J48	kNN	LEM2	SVM	RF	NB
BP	Acc	0.829	0.884	0.857	0.913(0.76)	0.867	0.881	0.84
BP	wAcc	0.747	0.515	0.586	0.659	0.538	0.536	0.699
BP	CV(s)	0.05	0.03	0.07	1.2	0.76	0.12	0.02
Geant	Acc	0.816	0.896	0.762	0.902(0.56)	0.87	0.853	0.761
Geant	wAcc	0.736	0.855	0.686	0.834	0.83	0.784	0.685
Geant	CV(s)	31	50	1739	14321	483	100	1759
Golub	Acc	0.874	0.839	0.879	0.754(0.95)	0.929	0.811	0.926
Golub	wAcc	0.793	0.82	0.815	0.675	0.877	0.689	0.873
Golub	CV(s)	0.7	1.2	1.2	47	1.5	4.1	1
hypothyroid	Acc	0.977	0.995	0.915	0.993(0.85)	0.936	0.991	0.953
hypothyroid	wAcc	0.673	0.724	0.437	0.71	0.383	0.708	0.533
hypothyroid	CV(s)	2.2	0.4	38.8	132	17.4	3.9	0.3
ionosphere	Acc	0.873	0.897	0.864	0.884(0.81)	0.879	0.927	0.827
ionosphere	wAcc	0.878	0.877	0.82	0.864	0.842	0.91	0.835
ionosphere	CV(s)	0.2	0.25	0.47	1.53	0.32	0.49	0.06
iris	Acc	0.932	0.941	0.95	0.967(0.94)	0.964	0.943	0.955
iris	wAcc	0.932	0.941	0.95	0.966	0.964	0.943	0.955
iris	CV(s)	0.01	0.01	0.02	1	1.93	0.04	0.01
mushroom	Acc	0.986	1	1	1(0.92)	1	1	0.954
mushroom	wAcc	0.985	1	1	1	1	1	0.953
mushroom	CV(s)	2.3	0.3	139.1	3.5	19	3.5	0.2
osusci	Acc	0.857	0.867	0.835	0.887(0.63)	0.863	0.887	0.479
osusci	wAcc	0.744	0.768	0.762	0.79	0.737	0.776	0.643
osusci	CV(s)	6.4	24.6	831.9	5006	196.5	40.4	2.9
soybean	Acc	0.819	0.889	0.909	0.969(0.83)	0.925	0.912	0.906
soybean	wAcc	0.864	0.865	0.919	0.873	0.953	0.914	0.918
soybean	CV(s)	2.3	0.1	1.4	8	121	1.6	1.4
splice.asx	Acc	0.861	0.934	0.734	0.949(0.45)	0.734	0.892	0.953
splice.asx	wAcc	0.857	0.934	0.789	0.951	0.79	0.873	0.948
splice.asx	CV(s)	1.9	0.5	51.8	29	54.4	5.5	0.2
tic.tac	Acc	0.769	0.833	0.975	1(0.83)	0.983	0.896	0.709
tic.tac	wAcc	0.71	0.797	0.964	1	0.976	0.867	0.64
tic.tac	CV(s)	0.16	0.05	0.85	0.5	1.22	0.32	0.02
vote	Acc	0.941	0.957	0.922	0.98(0.83)	0.957	0.957	0.901
vote	wAcc	0.942	0.955	0.926	0.98	0.958	0.954	0.904
vote	CV(s)	0.2	0.03	0.31	0.45	0.13	0.18	0.01
zoo	Acc	0.838	0.934	0.95	0.978(0.87)	0.952	0.906	0.95
zoo	wAcc	0.728	0.855	0.899	0.908	0.881	0.768	0.91
zoo	CV(s)	0.16	0.01	0.06	1.07	1.43	0.09	0.01
abalone	Acc	0.524	0.527	0.501	0.53(0.47)	0.545	0.533	0.519
abalone	wAcc	0.52	0.527	0.502	0.512	0.531	0.538	0.537
abalone	CV(s)	1.64	1.46	16.42	129	1.24	5.45	0.25
Alizadeh	Acc	0.948	0.837	0.977	0.837(0.95)	0.998	0.931	0.894
Alizadeh	wAcc	0.89	0.722	0.955	0.722	0.996	0.852	0.77
Alizadeh	CV(s)	0.77	1.44	1.68	21	1.15	4.55	1.21

Tablica 7.5. Wyniki jakości i szybkości działania badanych klasyfikatorów na danych rzeczywistych

predykcji. Prawie dla wszystkich zbiorów danych ADX wypada relatywnie bardzo dobrze plasując się pośród algorytmów najlepiej klasyfikujących. Analizując poszczególne wyniki klasyfikacji w sekcji opisującej badanie wpływu parametrów ADX (w szczególności wykresy 7.13, 7.14, 7.15, 7.16, 7.17) można



Rysunek 7.24. Średnie wAcc dla badanych klasyfikatorów - 10 krotna walidacja krzyżowa 3 zbiorowa

zauważyć, że odpowiedni dobór parametrów pozwala na znaczne poprawienie wyników w nielicznych zbiorach gdzie ADX wypadł słabiej. Dla zbioru "tic.tac.toe" przy $searchBeam = 100$, $finalBeam = 10$ i ustawieniu $selSignificantMethod = 0$ uzyskano średnie wAcc=0.985.

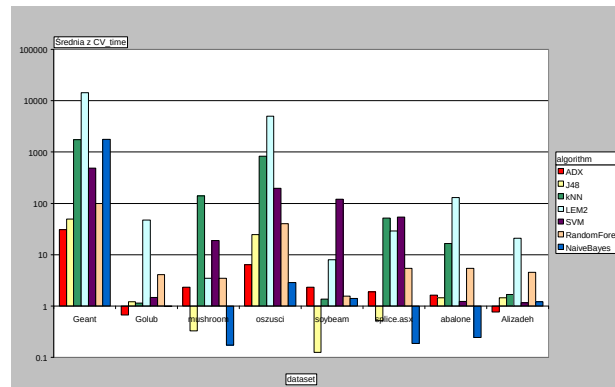
Dla zbioru "Soybean" przy $searchBeam = 50$, $finalBeam = 30$ i parametrze $selSignificantMethod = 1$ uzyskano średnie wAcc=0.864.

Dla zbioru "zoo" przy $searchBeam = 100$, $finalBeam = 10$ i ustawieniu $selSignificantMethod = 4$ uzyskano średnie wAcc=0.811. Natomiast dla zbioru "oszusi" przy $qMethodFinal = 4$ uzyskano średnie wAcc=0.790.

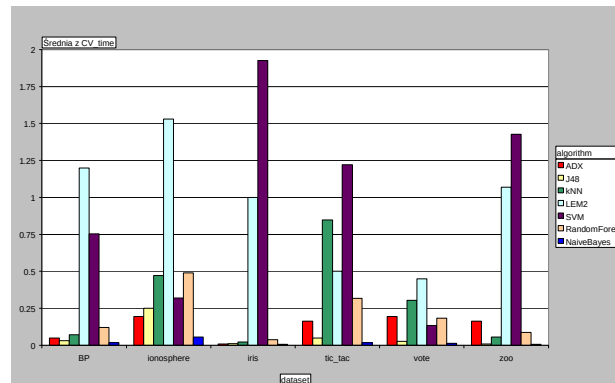
W przypadku algorytmu LEM2 wyniki w tabeli 7.5 są dodatkowo opisane pokryciem przykładów przez reguły. Oznacza to, że wskaźniki Acc i wAcc są liczone tylko w oparciu o przykłady sklasyfikowane (również w ten sposób wynik jest podawany w implementacji w RSES). Reszta obiektów przy braku nadanej etykiety nie jest uznawana za błędny wynik predykcji. W ten sposób szacowana jakość predykcji jest faktycznie zawyżona, a w skrajnym przypadku poprawna klasyfikacja jednego przykładu pozwala na osiągnięcie 100% klasyfikacji. W ADX brak nadania etykiety jest rzadkim zjawiskiem, lecz jeśli zajdzie to w końcowych wynikach uznaje się każdy nieoznaczony przykład jako niepoprawnie zaklasyfikowany.

Średni czas działania algorytmów w procesie walidacji krzyżowej zaprezentowano na wykresach 7.25 i 7.26 przy czym pierwszy z nich zbiera informacje dla większych zbiorów danych na skali logarytmicznej.

Szczególnie dla większych zbiorów liniowa skalowalność ADX silnie pozytywnie odróżnia go od pozostałych technik. Dla zbiorów "oszusi" i "geant" czas działania ADX to kilkadziesiąt sekund w porównaniu do kilkudziesięciu minut czy nawet kilku godzin w stosunku do najwolniejszych algorytmów. W przypadku mniejszych zbiorów algorytm ADX nadal jest jednym z szybszych lecz przeważnie różnica kilku sekund nie jest w praktyce bardzo istotna. Oczywiście w procesie optymalizacji parametrów klasyfikatora lub procedurze MCFS krótszy czas pojedynczego działania pozwala na znaczną redukcję



Rysunek 7.25. Średni czas walidacji krzyżowej (3 zbiory) dla dużych zbiorów danych - skala logarytmiczna



Rysunek 7.26. Średni czas walidacji krzyżowej (3 zbiory) dla mniejszych zbiorów danych

łącznego czasu procesu. Podsumowując algorytm ADX uzyskuje bardzo dobre wyniki klasyfikacji przy znakomitej skalowalności co czyni go niezwykle atrakcyjną pozycją szczególnie w środowisku biznesowym gdzie bardzo często wymaga się generowania czytelnej wiedzy w oparciu o niezwykle duże tablice danych.

7.2. Zadanie selekcji istotnych cech

We wszystkich eksperymentach tam gdzie nie zaznaczono inaczej jako algorytmu klasyfikacyjnego w metodzie MCFS użyto drzewa C4.5 (w wersji 8) jego implementacji z systemu WEKA 3-4-1 [Weka 3] (J48). Parametry klasyfikatora były w każdym eksperymencie ustawione na domyślne, czyli minimalna liczba przykładów w liściu na 2 ($minNumObj=2$), a $ConfidenceFactor=0.25$ w trybie przycinania drzewa.

Algorytmu ADX używano w tej części eksperymentów przy stałych ustawieniach dających dobre wyniki dla pojedynczej klasyfikacji oraz pozwalających uzyskać bardzo krótki czas uczenia i walidacji:

- $finalBeam = [20]$,
- $searchBeam = 50$,
- $complexGenerality = 0$,
- $qMethod = 1$,
- $qMethodFinal = 2$,
- $selSignificantMethod = 3$,
- $scoreMethod = 1$.

Wyczerpujący opis parametrów implementacji ADX znajduje się w rozdziale 8, a wyjaśnienie ich znaczenia w algorytmie a rozdziale 5.

W eksperymentach użyto oryginalnego zbioru danych Goluba ([Golub *et al.*, 1999]) bez wstępnego jego przetworzenia (w przeciwieństwie np. do prac Dudoit *et al.* [Dudoit *et al.*, 2002], [Dudoit *et al.*, 2003]). W przypadku drugiego zbioru danych (Alizadeh *et al.*), finalną selekcję przeprowadzono na dwóch postaciach zbioru: jednej z brakami danych i drugiej, w której braki danych wypełniono w taki sam sposób jak w [Tibshirani *et al.*, 2003].

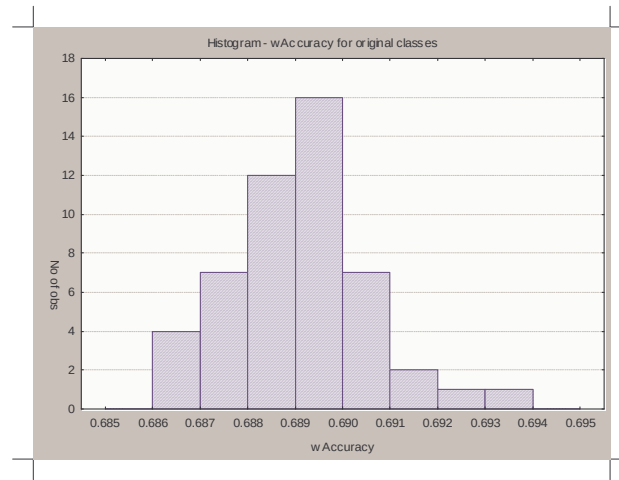
7.2.1. Test wiarygodności danych

W celu stwierdzenia wiarygodności danych dla obu zbiorów przeprowadzono po dwie serie eksperymentów budowy tysięcy klasyfikatorów na odpowiednich podzbiorach, raz dla oryginalnej wartości decyzji, a za drugim razem dla wartości decyzji poddanej permutacji (zgodnie z pierwszym krokiem walidacji danych opisanym w sekcji 6.4).

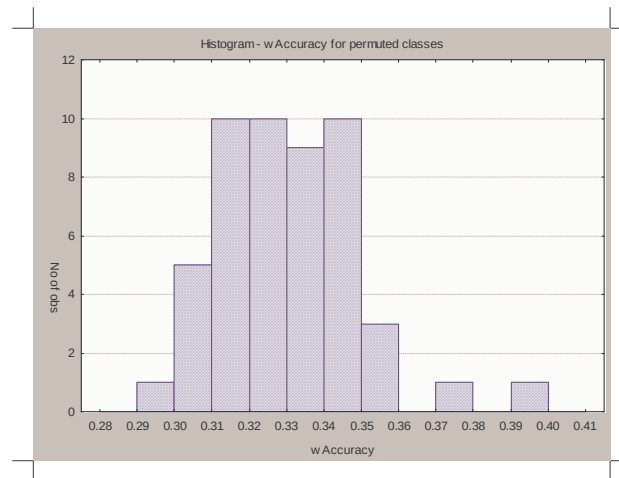
Dla danych Alizadeha (chłoniak) przeprowadzono 10 000 projekcji (wyboru atrybutów) losowo wybranych 1800 genów. Każdy taki podzbiór był następnie losowo dzielony 50 razy na trenujący i testujący. Łącznie utworzono 500 000 drzew decyzyjnych, a ich miary jakości predykcji na poszczególnych zbiorach testowych Acc i $wAcc$ zapisano.

W drugiej serii przeprowadzono podobny eksperyment dla 50 różnych losowych permutacji klasy decyzyjnej. Dla każdej permutacji zbudowano 200 losowych zbiorów z 1800 genów (projekcji), a następnie 50 razy dzielono na podzbiory trenujący i testujący. W sumie zbudowano również 500 000 drzew i tak jak poprzednio zachowano miary jakości predykcji na zbiorach testowych.

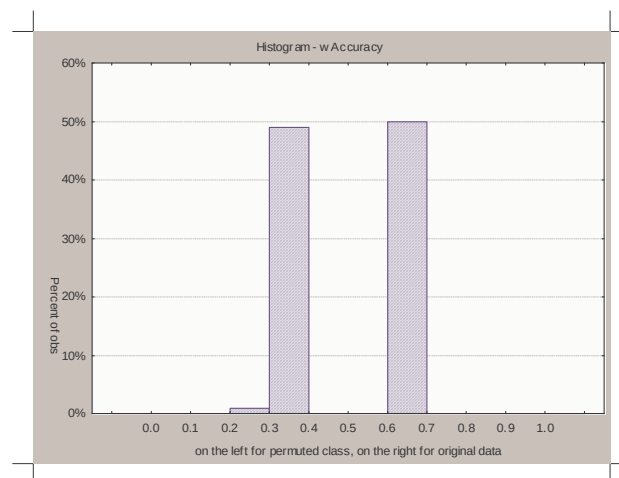
Dla drugiego zbioru danych (dane Goluba – białaczka) przeprowadzono identyczną procedurę. Dla oryginalnej wartości decyzji przeprowadzono 15 000 projekcji losowo wybranych 2000 genów, a następnie 30 razy dzielono na



Rysunek 7.27. Rozkład $wAcc$ dla oryginalnej wartości decyzji – zbiór Alizadeha (chłoniak)

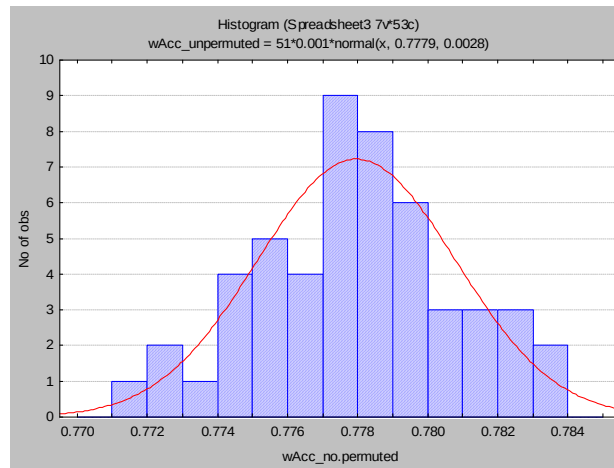


Rysunek 7.28. Rozkład $wAcc$ dla permutowanej wartości decyzji – zbiór Alizadeha (chłoniak)

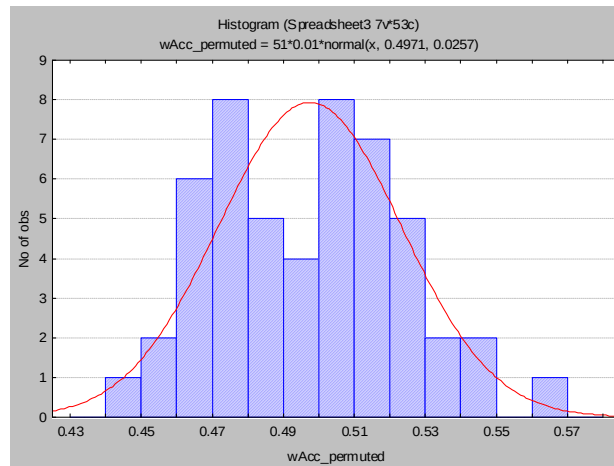


Rysunek 7.29. Porównanie obu rozkładów $wAcc$ – zbiór Alizadeha (chłoniak)

podzbiory trenujący i testujący (łącznie otrzymano 450 000 wyników predykcji). Dla 50 różnych permutacji wartości decyzji za każdym razem losowano 300 podzbiorów z 2000 genami i również dzielono następnie 30 krotnie na podzbiory trenujący i testujący. Łącznie znowu otrzymano 450 000 drzew i odpowiadające im Acc oraz $wAcc$.

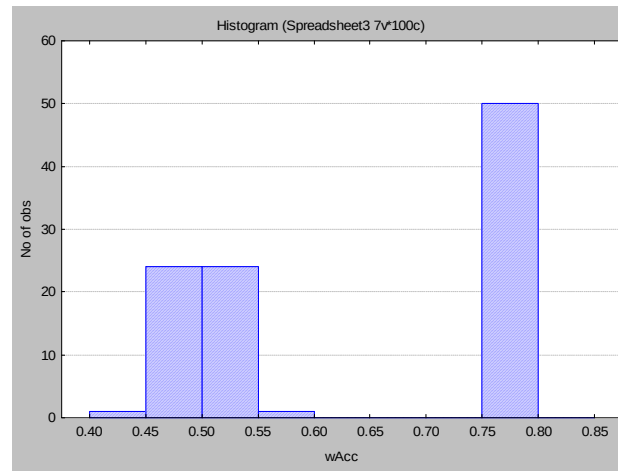


Rysunek 7.30. Rozkład $wAcc$ dla oryginalnej wartości decyzji – zbiór Goluba (białaczka)



Rysunek 7.31. Rozkład $wAcc$ dla permutowanej wartości decyzji – zbiór Goluba (białaczka)

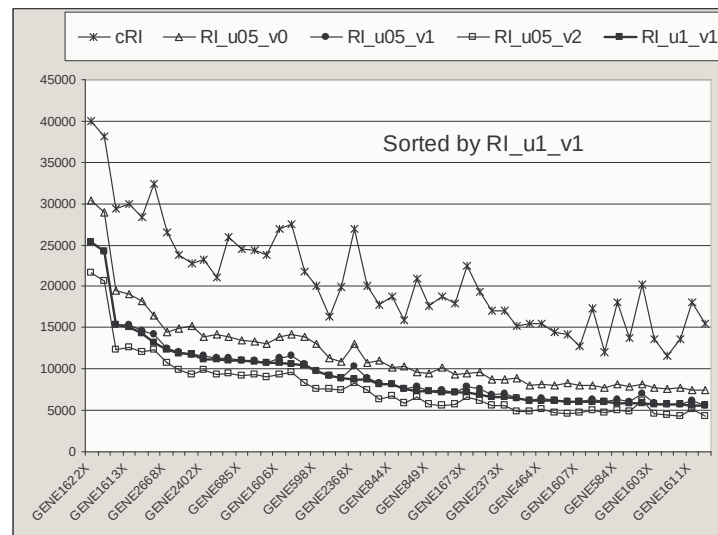
Porównanie dwóch rozkładów miary $wAcc$ obok siebie (dla obu badanych zbiorów) na rysunkach 7.29 i 7.32 wyraźnie pokazuje silne przesunięcie dla zbioru o oryginalnej wartości decyzji w stosunku do danych o losowej (permutowanej) klasie. Daje to solidne podstawy do stwierdzenia, że występujące zależności w danych są dalekie od przypadkowych i zdefiniowana klasa jest istotnie statystycznie zależna od dostępnych wartości atrybutów. Tym samym uzasadnia wiarygodność wyników otrzymanych w dalszej części pracy dla badanych zbiorów (chłoniak, białaczka).



Rysunek 7.32. Porównanie obu rozkładów $wAcc$ – zbiór Goluba (białaczka)

7.2.2. Badanie wpływu parametrów u, v

Dla zdefiniowanej miary jakości cRI_{g_k} (6.9) zbadano wpływ różnych wartości pary (u, v) . Do porównania dodano również surową miarę cRI (opartą wyłącznie na częstotliwości wystąpień poszczególnych cech we wszystkich zbudowanych klasyfikatorach).



Rysunek 7.33. Rankingi dla różnych wartości par parametrów (u, v) (dane Alizadech – chłoniak)

Rysunek 7.33 przedstawia porównanie otrzymanych wartości cRI_{g_k} dla następujących par (u, v) : $(0.5, 0)$, $(0.5, 1)$, $(0.5, 2)$ i $(1, 1)$, gdzie wyniki posortowano według RI_{g_k} dla $u = v = 1$. Dla danych Alizadeha różnice w kolejności genów, choć łatwo dostrzegalne, nie zmieniają silnie obrazu rankingu. Parametry u, v dodatkowo w subtelny sposób promują cechy o odpowiednio lepszych właściwościach predykcyjnych lub bardziej istotnych w klasyfikatorze. Wydaje się, że nie ma jednak żadnych istotnych powodów by silniej wpływać na jeden z czynników w cRI_{g_k} , dlatego też można uznać $u = v = 1$ za dobre

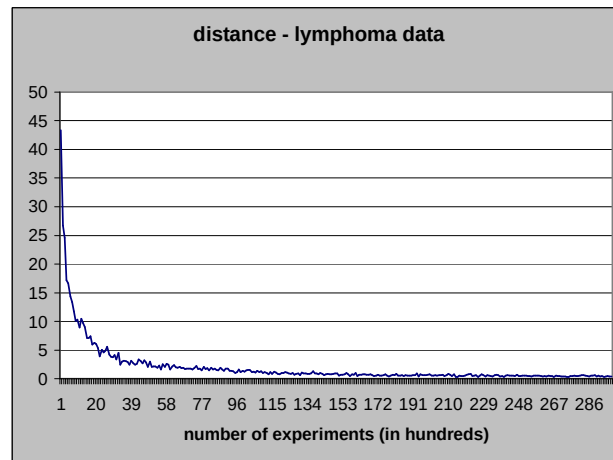
wartości domyślne i przy takim ich ustawieniu przeprowadzono pozostałe eksperymenty.

7.2.3. Badanie wpływu parametrów s , t , m

W przypadku każdego zbioru danych trudno deterministycznie dokładnie określić jakie wartości parametrów: s , t i m będą wystarczające do osiągnięcia stabilnego wyniku. Miara 6.12 definiująca dystans pozwala na bieżąco śledzić przebieg eksperymentu i stwierdzić stabilizację rankingu. W tej sekcji zbadany zostanie wpływ poszczególnych parametrów sterujących s , t , m na zbieżność algorytmu. Do eksperymentów użyto klasyfikatora C4.5 z domyślnymi parametrami uczenia.

Badanie zbieżności algorytmu w zależności od parametru s

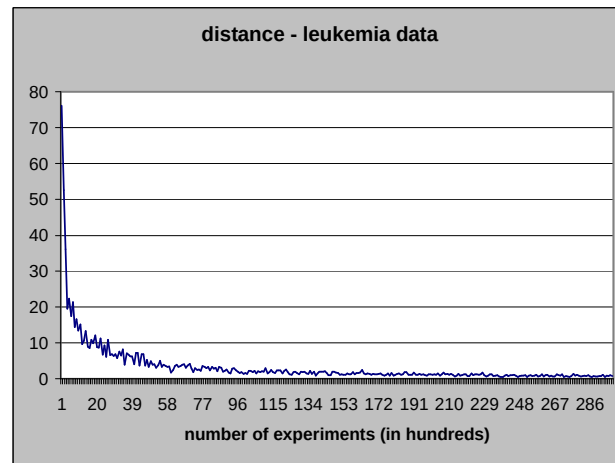
Wyniki analizy dystansów rankingów dla obu badanych zbiorów ([Alizadeh *et al.*, 2000], [Golub *et al.*, 1999]) przy stałych ustawieniach: $m = 300$, $t = 10$ i $u = v = 1$, zostały przedstawione na rysunkach 7.34 oraz 7.35. Każde drzewo klasyfikacyjne było trenowane każdorazowo na losowo wydzielonych 66% przykładach, a testowane na pozostałych (z zachowaniem proporcji w klasach). Losowano zatem 3000 razy po 300 genów (spośród 7129 w przypadku danych Goluba i 4026 dla danych Alizadeh), a dla każdego takiego podzbioru przeprowadzono 10 różnych procesów uczenia i testowania.



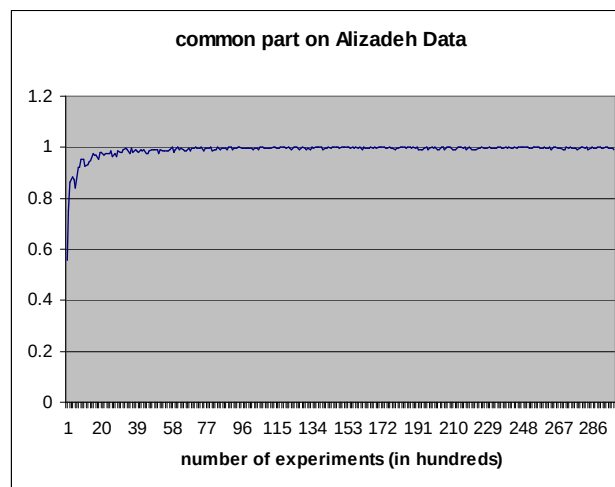
Rysunek 7.34. Dystans rankingów (dla 5% najwyżżej ocenianych cech) jako funkcja kolejnych wartości s – dane Alizadeh (chłoniak)

Oprócz rankingu w tym samym eksperymencie również obserwowano część wspólną kolejnych rankingów (rysunki 7.36, 7.37), która nie bierze pod uwagę kolejności genów, a jedynie opiera się na porównaniu dwóch najwyżżej ocenionych 5% zbiorów cech. Wartość 1 na wykresach oznacza dwa identyczne rankingi atrybutów w kolejnych iteracjach s_k i s_{k+10} procedury MCFS (każde 10 iteracji s to budowa 100 klasyfikatorów ponieważ $t = 10$).

Z wyników eksperymentów przedstawionych na rysunkach 7.35 i 7.34 wynika, że do osiągnięcia stabilnego wyniku (przy ustalonych warunkach) dla obu zbiorów danych w zupełności wystarczy $s = 3000$ ($st = 30000$ drzew).



Rysunek 7.35. Dystans rankingów (dla 5% najwyższych ocenianych cech) jako funkcja kolejnych wartości s – dane Golub (białaczka)

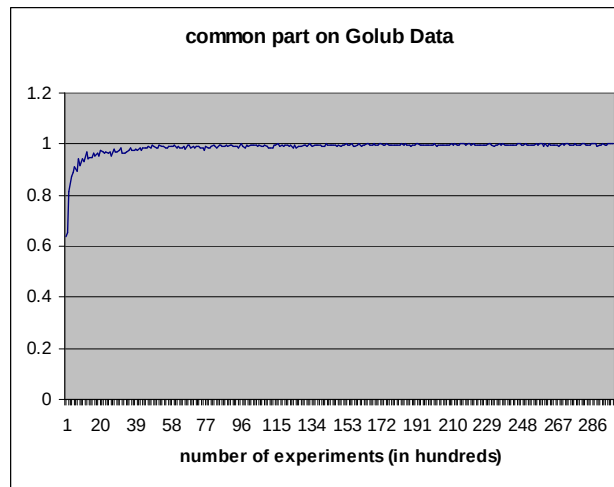


Rysunek 7.36. Część wspólna rankingów (dla 5% najwyższych ocenianych cech) jako funkcja kolejnych wartości s – dane Alizadeh (chłoniak)

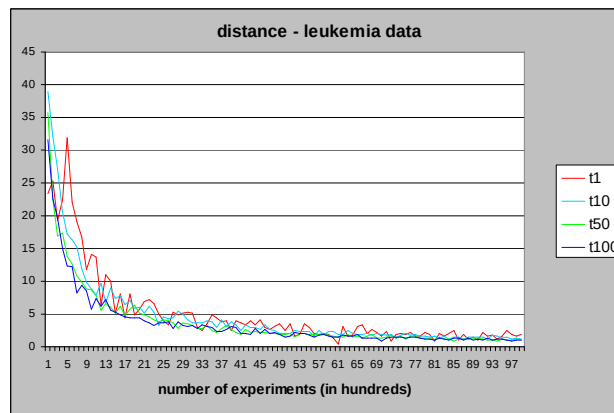
Badanie zbieżności algorytmu w zależności od parametru t

Dla ustawionych standardowo parametrów: $s = 3000$, $m = 300$, $u = v = 1$, $splitRatio = 0.66$, $bestSetFractionSize = 0.05$ zbadano wpływ parametru t na algorytm MCFS dla obu zbiorów danych. W eksperymencie zbadano następujące wartości t : 1, 5, 10, 30, 50, 100, 200, a jako klasyfikatora użyto implementacji drzewa decyzyjnego J4.8(C4.5).

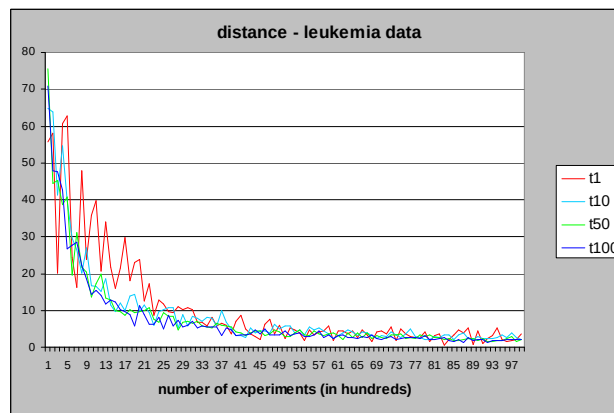
Na rysunkach 7.38 i 7.39 przedstawiono wykresy początkowego przebiegu krzywej dystansu w zależności od s dla wybranych wartości t . Na obu rysunkach uwidacznia się jedynie odmienny przebieg krzywej dla $t=1$ natomiast pozostałe mają znacznie bardziej gładkie przebiegi i są do siebie bardzo podobne. Dość intuicyjne wprowadzenie do algorytmu MCFS kilkukrotnego wewnętrznego losowania podzbiorów trenującego i testującego dla wylosowanego podzbioru cech ma jak widać bardzo korzystny wpływ na stabilizację wyniku – ustalanie się rankingu cech. Począwszy od wartości $t = 10$ przebiegi



Rysunek 7.37. Część wspólna rankingów (dla 5% najwyżżej ocenianych cech) jako funkcja kolejnych wartości s – dane Golub (białaczka)

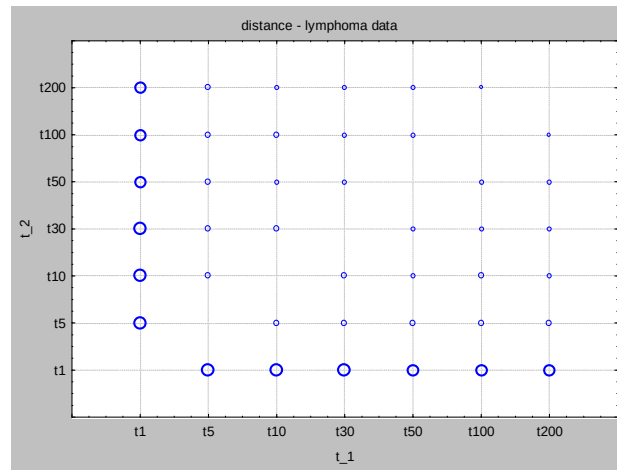


Rysunek 7.38. Dystans rankingów (dla 5% najwyżżej ocenianych cech) jako funkcja kolejnych wartości s dla wybranych t – dane Alizadeh (chłoniak)

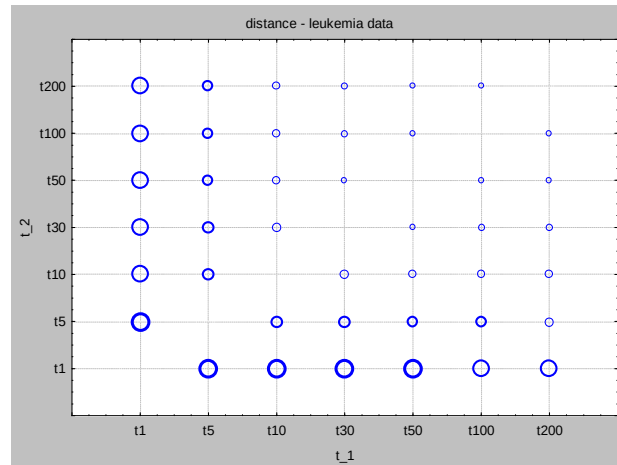


Rysunek 7.39. Dystans rankingów (dla 5% najwyżżej ocenianych cech) jako funkcja kolejnych wartości s dla wybranych t – dane Golub (białaczka)

znacznie się wygładzają i właściwie większe wartości t nie zmieniają procesu stabilizacji.



Rysunek 7.40. Porównanie wartości dystansu pomiędzy rankingami otrzymanymi finalnie dla różnych wartości parametru t – dane Alizadeh(chłoniak)

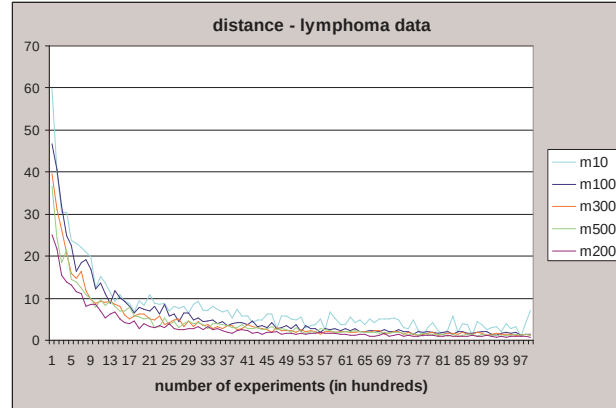


Rysunek 7.41. Porównanie wartości dystansu pomiędzy rankingami otrzymanymi finalnie dla różnych wartości parametru t – dane Golub (białaczka)

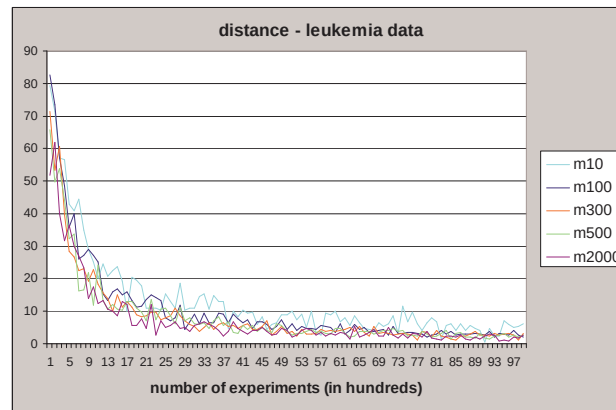
Na dwóch kolejnych rysunkach (patrz 7.40, 7.41) przedstawiono wartość $Dist_{norm}$ (liczoną pomiędzy finalnymi rankingami) jako wielkość bąbelka, natomiast osie x i y wyznaczają eksperymenty przeprowadzone przy odpowiedniej wartości t . Wartość $t = 1$ jest jak widać niewystarczająca ponieważ finalny ranking jest stosunkowo daleki od wszystkich pozostałych otrzymanych przy wyższych wartościach t . Począwszy od wartości $t = 10$ finalne rankingi są wzajemnie bardzo podobne do siebie (niska wartość $Dist_{norm}$) i dalsze zwiększanie wartości tego parametru nie przynosi żadnych znaczących zmian. Duża wartość parametru t silnie wpływa na sumaryczny czas obliczeń ponieważ determinuje liczbę łączną liczbę budowanych i ocenianych klasyfikatorów (st). Minimalna wartość t dla której oba przebiegi są wystarczająco gładkie to $t = 10$ i taka też wartość tego parametru jest domyślna i najczęściej używana w pozostałych eksperymentach.

Badanie zbieżności algorytmu w zależności od parametru m

Przy standardowych ustawieniach parametrów: $s = 3000$, $t = 10$, $u = v = 1$, $splitRatio = 0.66$, $bestSetFractionSize = 0.05$ oraz przy różnych ustawieniach m uruchomiono algorytm MCFS dla obu zbiorów danych. W eksperymencie zbadano następujące wartości m : 10, 20, 50, 100, 200, 300, 500, 1000, 2000 a jako klasyfikator użyto implementacji drzewa decyzyjnego J4.8(C4.5).



Rysunek 7.42. Dystans rankingów (dla 5% najwyższych ocenianych cech) jako funkcja kolejnych wartości s dla wybranych m – dane Alizadeh (chłoniak)



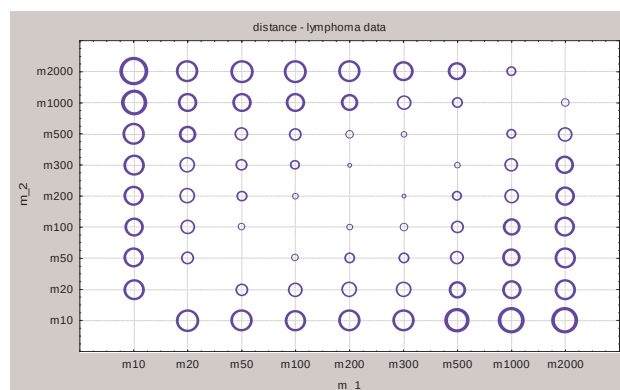
Rysunek 7.43. Dystans rankingów (dla 5% najwyższych ocenianych cech) jako funkcja kolejnych wartości s dla wybranych m – dane Golub (białaczka)

Na rysunkach 7.42 i 7.43 uwidacznia się pewna stała tendencja stabilizacji krzywej dystansu w zależności od m – dla wyższych m jej przebieg jest łagodniejszy. W celu lepszej prezentacji wyników na rysunkach przedstawiono tylko wybrane krzywe dla $m=[10,100,300,500,2000]$ oraz zakres s od 0 do 100 lecz eksperyment przeprowadzono dla wszystkich podanych wcześniej m i w pełnym zakresie s do 3000.

Duże m sprzyja większemu zróżnicowaniu wartości RI , ponieważ kilka najbardziej informatywnych cech będzie bardzo często losowana do poszczególnych podzbiorów co z kolei wpłynie na zwiększanie ich miary oceny kosztem wyboru innych cech również ważnych lecz już w mniejszym stopniu.

Mniejsze wartości parametru m przedziewdziają maskowaniu cech lecz jednocześnie zmniejszają realne szanse na pojawienie się informatywnych cech w poszczególnych podzbiorach. Dla zbyt niskiej wartości m do podzbiorów trafiają przeważnie tylko mało użyteczne atrybuty i tym samym część wybiera się do budowy klasyfikatora, a następnie odpowiednio zwiększa ich miarę RI mimo ich nikłej jakości.

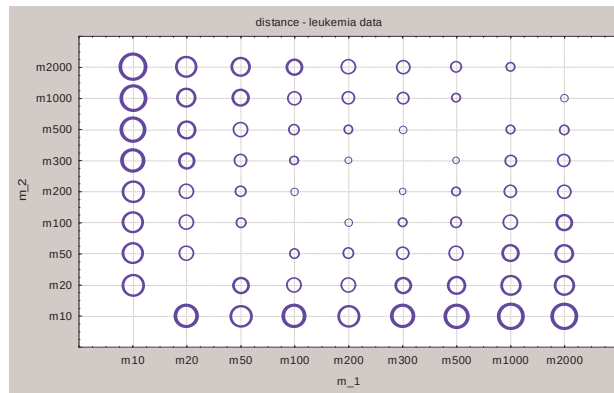
Należy zwrócić uwagę, iż dla wartości $m=300/500$ krzywe dystansu (na rysunkach 7.42 i 7.43) nie odbiegają kształtem od krzywej otrzymanej dla $m = 2000$, co świadczy o zapewnieniu istnienia informatywnych cech w losowanych podzbiorach danych. Jednocześnie porównując wykresy bąbelkowe (7.44 i 7.45) widać, iż dystans poszczególnych finalnych rankingów jest najmniejszy dla wartości 300, 500. Na wspomnianych rysunkach przedstawiono odległość $Dist_{norm}$ jako wielkość bąbelka w zależności od odpowiedniej pary wartości m . Oczywiście dla skrajnych wartości m finalne rankingi znacznie się różnią od siebie lecz wzajemne wysokie podobieństwo rankingów otrzymuje się dla m o wartości kilkaset. Dobrym parametrem domyślnym będzie zatem $m=300$ dla którego krzywa dystansu dość szybko się stabilizuje i jednocześnie jest to wartość na tyle niska by dobrze zapobiegać maskowaniu cech.



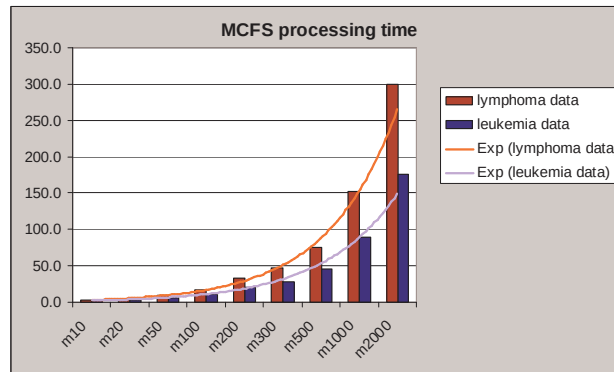
Rysunek 7.44. Porównanie wartości dystansu pomiędzy rankingami otrzymanymi finalnie przy różnych wartościach parametru m – dane Alizadeh(chłoniak)

Dodatkowym dość ważnym kryterium doboru wartości m jest czas obliczeń, ponieważ w przypadku klasyfikatora J4.8 rośnie on wykładniczo w stosunku do parametru m (porównaj rysunek 7.46). Przy stałych w tym eksperymencie ustawieniach oraz dla $m=300$ na komputerze Pentium IV (3.2GHz, HT, 2GB RAM) czas potrzebny do wygenerowania finalnego rankingu to odpowiednio 46 (dane chłoniak) i 28 (dane białaczka) minut.

Dobór większości parametrów jest wynikiem pewnego kompromisu ze względu na założone kryteria. W przypadku wysokowymiarowych danych można założyć, że tylko kilka procent atrybutów jest rzeczywiście istotna. Dlatego warto tak ustawić parametr m by prawdopodobieństwo wylosowania przynajmniej jednego istotnego atrybutu do podzbioru było wystarczająco wysokie. Dla danych o 5000 atrybutach zakładając, że istotny jest 1% cech (czyli 50) dla $m=100$ średnio mamy szansę wylosować 1 atrybut istotny w pojedynczym podzbiorze cech. Pozwala to dość łatwo oszacować wartość



Rysunek 7.45. Porównanie wartości dystansu pomiędzy rankingami otrzymanymi finalnie przy różnych wartościach parametru m – dane Golub (białaczka)



Rysunek 7.46. Porównanie czasu działania (w minutach) MCFS dla różnych wartości parametru m

parametru m i przyjąć jako domyślną dla dowolnego wysokowymiarowego problemu na poziomie kilku procent wejściowych atrybutów.

7.2.4. Przeciwdziałanie wzajemnemu maskowaniu cech

Dla dość specyficznego zbioru danych opisującego przypadki chłoniaka [Alizadeh *et al.*, 2000] przeprowadzono szereg eksperymentów w celu sprawdzenia istotności problemu wzajemnego maskowania genów. Pomysły, które pomagają przeciwdziałać temu problemowi zostały wcześniej omówione w sekcji 6.3.8.

Wszystkie klasy - klasyczne podejście

W klasycznym podejściu MCFS do klasyfikacji zawsze rozważa się wszystkie klasy decyzyjne (w tym wypadku 3) bez jakichkolwiek wprowadzonych zmian. Dla 4026 cech występujących w danych użyto procedury MCFS przy następujących ustawieniach parametrów:

- ilość cech w pojedynczym podzbiorze $m = 100$,
- ilość losowych podzbiorów cech $s = 60000$,
- ilość losowych podziałów podzbiorów na dane trenujące i testujące $t = 30$,

- stosunek przykładów trenujących do testujących 2/3 (40 obiektów do uczenia – 22 do testowania).
- wykładniki potęg $u = v = 1$.

Parametr d specjalnie ustawiono na dość niskim poziomie w celu przeciwdziałania maskowaniu informatywnych genów przez inne. Przy niskiej wartości tego parametru zwiększa się prawdopodobieństwo tego, że za każdym razem będzie wylosowana zupełnie inna konfiguracja cech, co z kolei zmniejszy szansę jednoczesnego częstego pojawiania się porównywalnych (podobnych) cech. Finalnie w trakcie tego eksperymentu zbudowano imponującą liczbę $s \cdot t = 1800000$ klasyfikatorów (drzew).

Dla tak zdefiniowanej metody MCFS otrzymano ranking cech posortowanych według RI – 45 najwyżej ocenionych przedstawiono poniżej.

<i>FL, DLCL, CLL</i>	<i>FL, DLCL, CLL</i>	<i>FL, DLCL, CLL</i>
1. GENE1622X	16. GENE1606X	31. GENE1673X
2. GENE1602X	17. GENE669X	32. GENE2553X
3. GENE1553X	18. GENE622X	32. GENE2097X
4. GENE1613X	19. GENE598X	32. GENE2373X
5. GENE1610X	20. GENE1661X	32. GENE646X
6. GENE2426X	21. GENE651X	32. GENE694X
7. GENE2668X	22. GENE2368X	32. GENE464X
8. GENE530X	23. GENE524X	32. GENE650X
9. GENE653X	24. GENE642X	32. GENE626X
10. GENE2402X	25. GENE844X	32. GENE1607X
11. GENE1647X	26. GENE834X	32. GENE2404X
12. GENE639X	27. GENE1605X	32. GENE1632X
13. GENE685X	28. GENE849X	32. GENE584X
14. GENE537X	29. GENE588X	32. GENE1648X
15. GENE542X	30. GENE640X	32. GENE1672X

Jedna klasa przeciwstawiona pozostałym połączonym

W tym eksperymencie przy identycznych ustawieniach jak poprzednio oryginalny zbiór danych poddano pewnej modyfikacji i uruchomiono podstawową procedurę MCFS trzykrotnie. Dla problemu o trzech klasach za każdym razem klasyfikacja budowana jest w oparciu o 2 klasy, z których jedna powstała poprzez połączenie dwóch. I tak dla rozważanego zbioru [Alizadeh *et al.*, 2000] łączono dwie kolejnie klasy ($DLCL + CLL \iff FL$, $DLCL + FL \iff CLL$, $CLL + FL \iff DLCL$) pozostawiając trzecią bez zmian. Problem o trzech wartościach decyzji sprowadzono zatem to problemu o binarnej decyzji. Sumaryczny ranking z tak przeprowadzonego eksperymentu zbudowano w oparciu o najwyżej ocenione geny z trzech osobnych rankingów.

$FL \iff DLCL + CLL$	$CL \iff DLCL + FL$	$DLCL \iff CLL + FL$
1. GENE1672X	1. GENE2668X	1. GENE1622X
2. GENE1673X	2. GENE2426X	2. GENE1602X
3. GENE1625X	3. GENE669X	3. GENE1553X
4. GENE1192X	4. GENE844X	4. GENE530X
5. GENE1583X	5. GENE537X	5. GENE1610X
6. GENE3704X	6. GENE639X	6. GENE1647X
7. GENE1676X	7. GENE542X	7. GENE1613X
8. GENE1610X	8. GENE685X	8. GENE2402X
9. GENE2309X	9. GENE2368X	9. GENE653X
10. GENE30X	10. GENE651X	10. GENE1661X
11. GENE3285X	11. GENE524X	11. GENE622X
12. GENE3705X	12. GENE2553X	12. GENE598X
13. GENE3286X	13. GENE454X	13. GENE1606X
14. GENE2416X	14. GENE694X	14. GENE642X
15. GENE3283X	15. GENE771X	15. GENE1662X

Klasyfikacja dwóch spośród trzech klas

Podobnie jak poprzednio przy stałe tych samych podstawowych parametrach procedury (m, s, t, u, v) przeprowadzono eksperyment polegający na trzykrotnym uruchomieniu MCFS dla wyłącznie pary klas. Za każdym razem wyłączano z rozważań jedną z klas tak by otrzymać w kolejnych rankingach cechy, które szczególnie dobrze separują od siebie obiekty o dwóch różnych wartościach decyzji. Bez względu na ułożenie w obiektów w przestrzeni cech w źródłowym problemie usunięcie jednej z klas wprowadza pewne uproszczenie, które wydaje się powinno pomóc w wyborze atrybutów. Tutaj również wybrano 15 najwyżej ocenionych cech dla każdego binarnego problemu (DLCL vs. FL, CLL vs. FL, etc.) tak by finalnie otrzymać zbiór 45 cech.

$DLCL \iff CLL$	$DLCL \iff FL$	$CLL \iff FL$
1. GENE640X	1. GENE1606X	1. GENE2426X
2. GENE694X	2. GENE1673X	2. GENE2368X
3. GENE832X	3. GENE1622X	3. GENE816X
4. GENE563X	4. GENE1613X	4. GENE2244X
5. GENE717X	5. GENE1611X	5. GENE3497X
6. GENE733X	6. GENE1602X	6. GENE2554X
7. GENE760X	7. GENE2402X	7. GENE2391X
8. GENE626X	8. GENE1605X	8. GENE3792X
9. GENE669X	9. GENE1610X	9. GENE2364X
10. GENE2346X	10. GENE1672X	10. GENE459X
11. GENE655X	11. GENE584X	11. GENE2547X
12. GENE537X	12. GENE2404X	12. GENE669X
13. GENE639X	13. GENE530X	13. GENE528X
14. GENE617X	14. GENE2356X	14. GENE2378X
15. GENE849X	15. GENE1619X	15. GENE586X

Wszystkie klasy - kolejne wyłączanie grupy najlepszych cech

W tym podejściu jest realizowana trochę odmienna metoda przeciwdziałania maskowaniu cech. Procedura MCFS jest w uruchamiana wielokrotnie (w przeprowadzonym eksperymencie 10 razy) z kolejnym wyłączaniem kilku najlepszych cech (tutaj 5). Pozostałe parametry zachowano bez zmian w stosunku do klasycznego ustawienia z różnicą dla $s = 6000$. Łączna ilość zbudowanych drzew jest zatem tożsama z pierwszym eksperymentem i finalnie wynosi również $10 \cdot s \cdot t = 1800000$ drzew.

<i>FL, DLCL, CLL</i>	<i>FL, DLCL, CLL</i>	<i>FL, DLCL, CLL</i>
1. GENE1602X	1. GENE542X	1. GENE849X
2. GENE1622X	2. GENE598X	2. GENE694X
3. GENE1553X	3. GENE2402X	3. GENE2373X
4. GENE1613X	4. GENE669X	4. GENE1673X
5. GENE2426X	5. GENE622X	5. GENE464X
1. GENE1610X	1. GENE2368X	1. GENE2097X
2. GENE685X	2. GENE844X	2. GENE563X
3. GENE2668X	3. GENE1661X	3. GENE1632X
4. GENE530X	4. GENE651X	4. GENE626X
5. GENE653X	5. GENE834X	5. GENE717X
1. GENE537X	1. GENE642X	1. GENE1607X
2. GENE524X	2. GENE640X	2. GENE2404X
3. GENE1647X	3. GENE646X	3. GENE2553X
4. GENE639X	4. GENE588X	4. GENE650X
5. GENE1606X	5. GENE1605X	5. GENE2346X

Porównanie otrzymanych wyników

Tabela 7.6 przedstawia porównanie wspólnych genów wyselekcjonowanych przy użyciu odpowiednich trybów uruchomienia MCFS. Maksymalnie podobne (29 identycznych atrybutów) rankingi otrzymano dla trybu gdzie jednocześnie rozważamy wszystkie klasy oraz gdzie łączymy dwie klasy przeciwstawiając je trzeciej. Najsilniej natomiast różni się metoda gdzie rozważamy jednocześnie tylko dwie klasy od trybu łączenia klas – tylko 16 wspólnych cech w rankingach. Należy zwrócić uwagę na niemalże identyczne zbiory cech dla trybów 1 i 4. Przy kolejnym usuwaniu czołówki rankingu (5 cech) otrzymujemy właściwie ten sam wynik, co pozwala wstępnie stwierdzić iż MCFS w pierwotnej postaci dobrze radzi sobie z problemem wzajemnego maskowania cech.

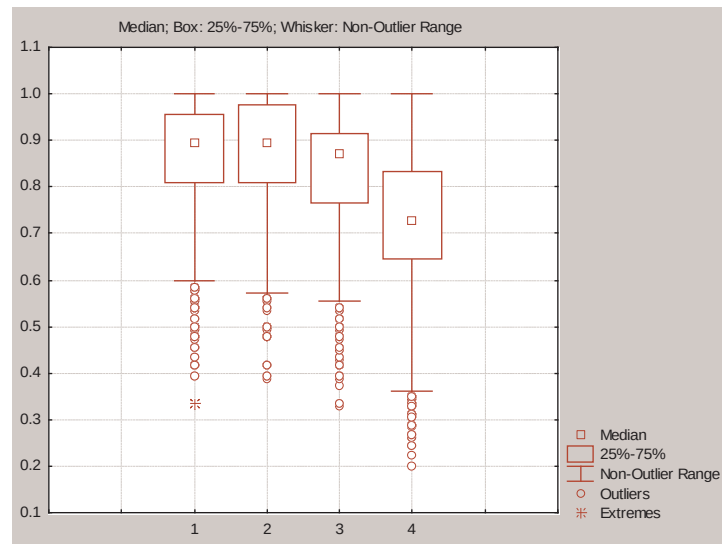
Do porównania wyników różnych trybów działania MCFS dla każdego z nich wybrano najbardziej informatywne geny (tak jak poprzednio) i przeprowadzono kolejną serię eksperymentów. Bazując na 45 atrybutach zbudowano kolejne rozkłady wartości oceny predykcji dla losowego podziału obiektów na zbiory trenujące (40 przykładów) i testujące (pozostałe 22 przykłady). Liczba 45 cech została wybrana arbitralnie w tej fazie prac eksperymentalnych, lecz konsekwentnie użyta dla wszystkich otrzymanych wcześniej rankingów. Dla każdego zbioru najistotniejszych cech zbudowano milion klasyfikatorów (drzew C4.5), a ich ważoną dokładność ($wAcc$) zachowano. W celu porów-

tryb	1	2	3	4
1	x	29	21	42
2	29	x	16	29
3	21	16	x	23
4	42	29	23	x

Tablica 7.6. Porównanie ilości wspólnych cech w odpowiednich parach rankingów (45 cech); 1: klasyfikacja 3 klas; 2: jedna klasa przeciwstawiona pozostałym połączonym; 3: klasyfikacja tylko dwóch klas jednocześnie; 4: kolejne usuwanie 5 najwyżżej ocenionych cech

nanania otrzymanych rezultatów zbudowano również rozkład wartości $wAcc$ w oparciu o losowo wyselekcjonowane geny (45 spośród 4026).

Rezultaty dla otrzymanego $wAcc$ dla różnych trybów MCFS zestawiono na wykresie ramkowym 7.47. Porównanie rozkładów pokazuje, że bez względu na zastosowaną metodę selekcji finalnego zbioru cech udało się za każdym razem dokonać znaczenie lepszej predykcji na wybranych 45 atrybutach niż zupełnie losowych.



Rysunek 7.47. $wAcc$ dla zbiorów o 45 atrybutach; 1: klasyfikacja 3 klas; 2: jedna klasa przeciwstawiona pozostałym połączonym; 3: klasyfikacja tylko dwóch klas jednocześnie 4: losowy wybór cech

Pierwsza metoda (przetwarzanie wszystkich klas jednocześnie) i druga (gdzie przeciwstawiamy jedną klasę pozostałym połączonym) pozwalają na uzyskanie najwyższej mediany wyników $wAcc$. Co ciekawe druga metoda jest nieznacznie lepsza (3 kwartył nieco wyżej). Wynika stąd, iż dla badanego zbioru danych w metodzie drugiej udało się trochę lepiej wybrać geny separujące od siebie dwie małe klasy (11 obiektów w CLL i 9 w FL) zamiast kolejnych, które wciąż bardzo dobrze oddzielają klasę DLCL od reszty. Do-

datkowo dla podejścia 2 otrzymano również mniej wartości $wAcc$ odstających od rozkładu co pokazuje trochę większą stabilność. Sporą wadą drugiego podejścia jest jednak czas obliczeń, ponieważ w przypadku zbioru o 3 wartościach decyzji MCFS uruchamia się 3 krotnie za każdym razem łącząc ze sobą inną parę klas. W przypadku pierwszego trybu problemu tego typu nie ma (otrzymujemy jeden ranking dla jednego uruchomienia), dodatkowo dysponując dużą uniwersalnością rozwiązania (bez względu na liczbę klas), które okupione jest pomijalnie słabszym wynikiem.

Podsumowując, problem który wydawało się, że może stanowić zagrożenie w obiektywnej ocenie cech dla nietypowego zbioru Alizadeha jest właściwie marginalny i podstawowy algorytm MCFS radzi sobie doskonale z tego typu danymi. Pozostałe metody zaimplementowano w celach badawczych i nadal są dostępne w programie, choć tryb przetwarzania wszystkich klas jednocześnie pozostał jako tryb domyślny.

7.2.5. Finalna selekcja istotnego zbioru atrybutów

W celu wyboru punktu odcięcia w rankingu, przy którym formułujemy postulat o informatywności genów, posłużono się zbadaniem otrzymanych wartości RI w identycznych warunkach eksperymentalnych lecz przy wymuszeniu braku zależności wartości decyzji.

Dla obu zbiorów badanych ([Alizadeh *et al.*, 2000], [Golub *et al.*, 1999]) w tej fazie zastosowano następujące parametry w MCFS:

- $s = 3000$,
- $t = 10$,
- $m = 300$,
- $u = v = 1$.
- $splitRatio = 0.66$.

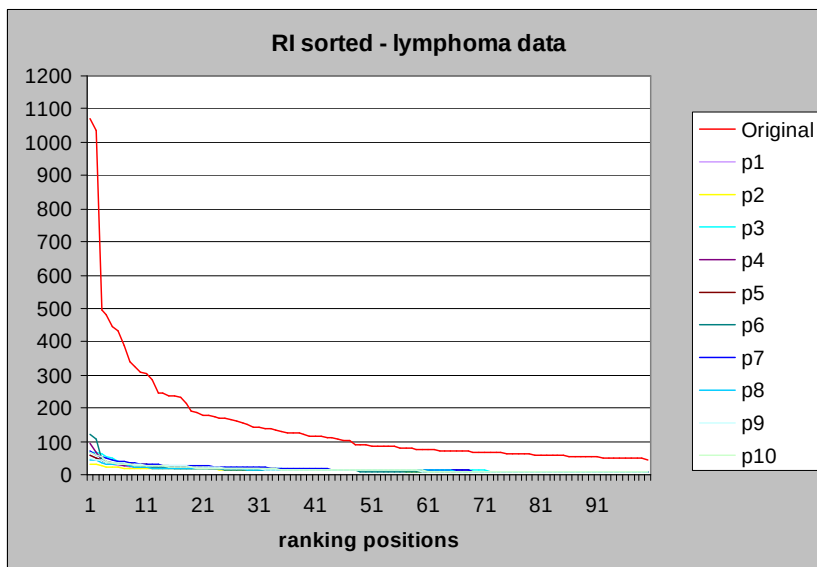
Następnie przeprowadzono dla każdego z rozważanych zbiorów 30 eksperymentów dla permutowanej zmiennej decyzyjnej i jednorazowo referencyjny dla oryginalnej klasy.

W tabeli 7.7 zestawiono maksymalne wartości RI ($maxRI$) odpowiednio otrzymane dla oryginalnej wartości decyzji i kolejnych jej permutacji (od p1 do p30) – dane chłoniak. Maksymalna zaobserwowana wartość RI wynosi 119.8 i wyznacza ona punkt odcięcia na poziomie 38 cech dla rankingu o oryginalnej wartości decyzji. Oznacza to, iż 38 atrybutów ma swoją wartość RI równą lub większą dla referencyjnego eksperymentu. Jeśli natomiast uznamy rozkład wartości $maxRI$ za normalny to p-wartość=0.05 dla takiego rozkładu jest w punkcie 86.09, co z kolei odcina dokładnie 50 najwyższych ocenionych cech. Postulat o normalności rozkładu sprawdzamy przy pomocy testu Shapiro-Wilka dla którego p-wartość wyniosła 0.72 (testowany rozkład jest z bardzo dużym prawdopodobieństwem rozkładem normalnym).

W tabeli 7.8 analogicznie jak poprzednio zebrano maksymalne wartości – $maxRI$ dla danych Goluba. Tylko 4 cechy w rankingu referencyjnym mają wartość RI większą niż maksymalna zaobserwowana 305.8 dla permutowanej zmiennej. Wartość 305.8 jest jednak obserwacją silnie odstającą od reszty wartości co sugeruje niezwykle 'szczęśliwą' permutację dla jednej z cech w

oryg. 1069.4		
p1. 58.6	p11. 66.2	p21. 87.7
p2. 31.9	p12. 57.7	p22. 61.9
p3. 68.6	p13. 108.3	p23. 101.0
p4. 94.3	p14. 63.0	p24. 114.8
p5. 60.0	p15. 61.8	p25. 81.2
p6. 119.8	p16. 49.1	p26. 84.7
p7. 73.1	p17. 86.1	p27. 83.2
p8. 45.8	p18. 100.1	p28. 96.5
p9. 47.5	p19. 88.3	p29. 79.9
p10. 69.0	p20. 119.8	p30. 68.6

Tablica 7.7. Zestawienie $maxRI$ dla zbioru danych chłoniak (Alizadeh)



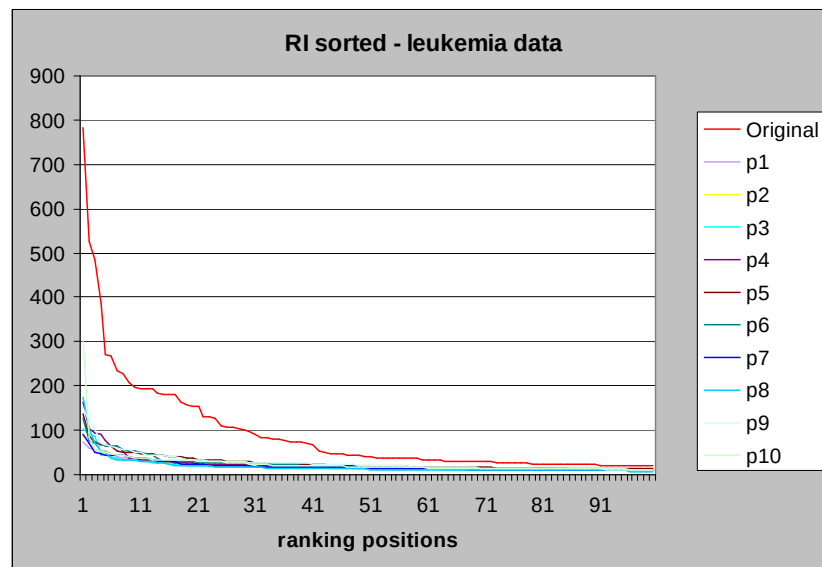
Rysunek 7.48. Porównanie wartości RI dla 100 najwyżżej ocenionych cech – dane chłoniak (Alizadeh)

eksperymentcie oznaczonym jako p10. W tym rankingu kolejny gen oceniono już znacznie niższą wartością $RI=82.7$, co wskazuje na przypadkowo wysoką miarę oceny pierwszego z listy. Jeśli uznamy rozkład wartości $maxRI$ za normalny to p-wartość=0.05 dla takiego rozkładu jest w punkcie 129.8, co z kolei odcina 22 najwyżżej ocenione cechy. Test Shapiro-Wilka po usunięciu odstającej wartości wskazuje na rozkład normalny, a p-wartość tego testu wynosi 0.26. Rozkłady wartości $maxRI$ przedstawione w tabelach 7.7 i 7.8 zaprezentowano na wykresie ramkowym 7.50, odrzucona wartość 305.8 w przypadku zbioru Goluba silnie odstaje od pozostałych.

Podczas gdy porównanie wartości miary oceny RI dla zbiorów o ory-

oryg. 781.4		
p1. 73.6	p11. 126.2	p21. 109.4
p2. 86.2	p12. 98.3	p22. 97.0
p3. 102.1	p13. 107.6	p23. 227.4
p4. 165.5	p14. 92.2	p24. 69.4
p5. 137.2	p15. 141.0	p25. 94.1
p6. 126.2	p16. 162.2	p26. 65.0
p7. 91.9	p17. 141.0	p27. 67.2
p8. 174.8	p18. 74.5	p28. 54.0
p9. 151.1	p19. 93.3	p29. 146.7
p10. 305.8	p20. 47.4	p30. 160.3

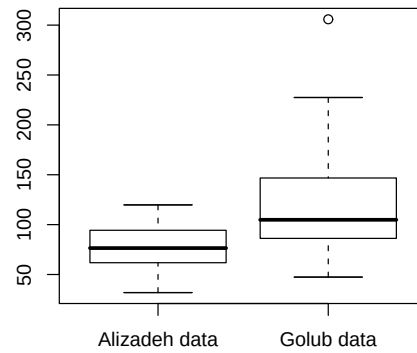
Tablica 7.8. Zestawienie $maxRI$ dla zbioru danych białaczka (Golub)



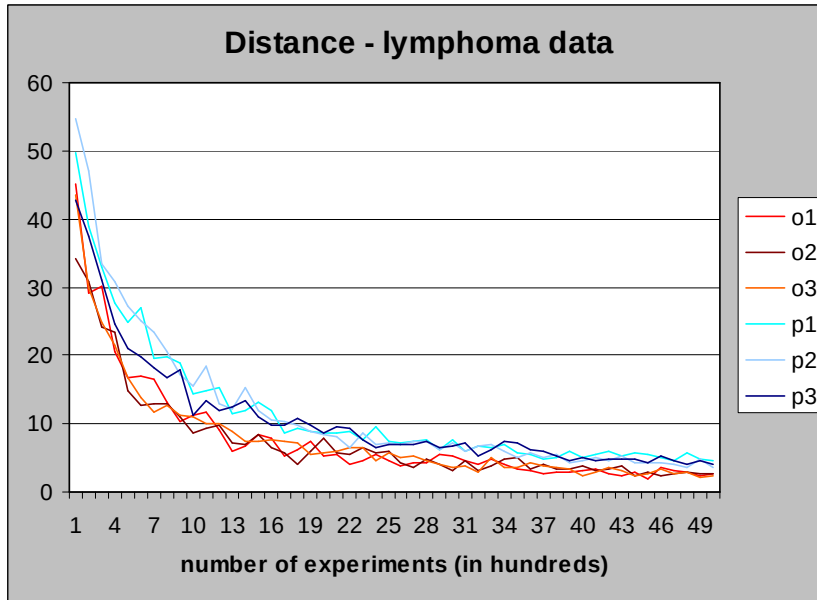
Rysunek 7.49. Porównanie wartości RI dla 100 najwyższych ocenionych cech – dane białaczka (Golub)

ginalnych oraz permutowanych wartościach atrybutu decyzyjnego wykazują niezwykle wyraźne różnice (patrz rysunki 7.48 i 7.49), to miara dystansu obserwowana wraz ze zwiększaniem parametru s nie jest już tak silnie zróżnicowana. Na rysunkach 7.51 i 7.52 wykreślono 6 różnych przebiegów miary dystansu rankingów (dla najwyższych ocenionych 5%) w zależności od s , 3 dla różnych permutacji i 3 dla różnych uruchomień MCFS dla zbioru o oryginalnej zmiennej decyzyjnej.

W obu grupach przypadków, można zaobserwować wyraźną stabilizację rankingów, lecz dla zbiorów o permutowanej decyzji miara dystansu jest na nieznacznie wyższym poziomie (odcienie koloru niebieskiego). Czołowe po-



Rysunek 7.50. Rozkłady wartości $maxRI$ otrzymane z 30 eksperymentów dla permutowanej zmiennej decyzyjnej

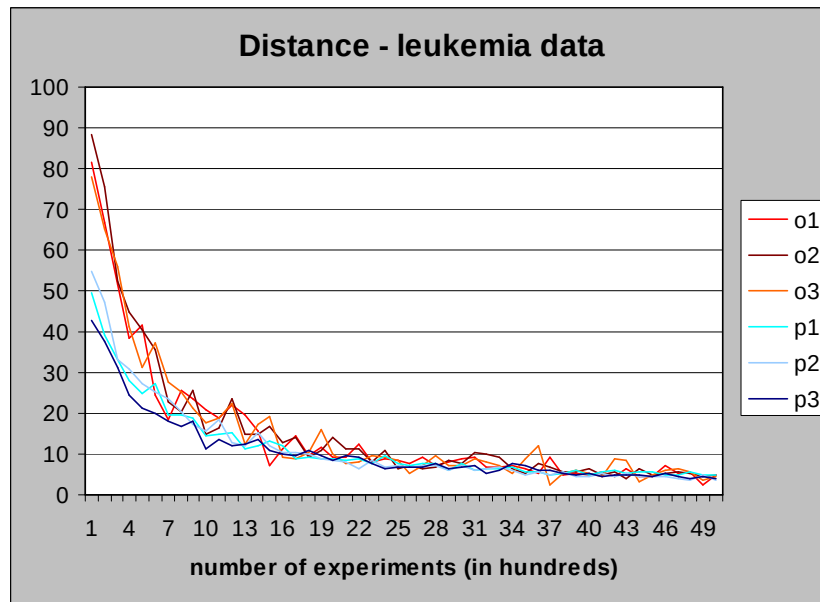


Rysunek 7.51. Porównanie początkowych fragmentów krzywych dystansu dla oryginalnej decyzji i permutowanej – dane chłoniak (Alizadeh)

zycje rankingów ustalają się zatem niezależnie od wprowadzonej losowości, natomiast czynnikiem krytycznym w RI jest miara $wAcc$, która to ocenia zdolności predykcyjne drzew opartych na poszczególnych cechach. Dla zbiorów o losowej zależności mimo stosunkowo wysokiej pozycji w ustalonych już rankingach cechy są tylko częściej niż inne wybierane do budowy drzew (tak jak ma to miejsce w przypadku $crudeRI$), natomiast nie przedstawiają one żadnej dodatkowej wartości. Porównanie odpowiednich wartości $wAcc$ znajduje się w tabeli 7.9.

7.2.6. Badanie wpływu klasyfikatora na ranking cech w MCFS

Jednym z ważniejszych pytań jakie można sobie zadać po zapoznaniu się z algorytmem MCFS jest to, czy wybór klasyfikatora ma krytyczny wpływ na finalny ranking cech? Wydaje się, że skoro każdy klasyfikator bazuje na atrybutach wybranych przy pomocy własnych kryteriów oceny, to ranking powinien być silnie zależny od zastosowanego algorytmu. Z drugiej jednak



Rysunek 7.52. Porównanie początkowych fragmentów krzywych dystansu dla oryginalnej decyzji i permutowanej – dane białaczka (Golub)

zbiór	o1	o2	o3	p1	p2	p3
Zbiór chłoniak (Alizadeh)	0.757	0.756	0.755	0.319	0.304	0.333
Zbiór białaczka (Golub)	0.726	0.730	0.730	0.476	0.488	0.504

Tablica 7.9. Zestawienie średniego $wAcc$ dla drzew budowanych podczas działania MCFS

strony informacja niesiona przez poszczególne cechy jest czymś bardziej uniwersalnym, a klasyfikatory różnią się jedynie sposobem ich selekcji i wykorzystania. Do porównania rankingów otrzymywanych przez odmienne algorytmy klasyfikacji użyto następujących zaimplementowanych w dmLab klasyfikatorów:

- implementacja drzewa decyzyjnego J48 z WEKA - algorytm C4.5,
- reimplementacja drzewa decyzyjnego - algorytm Sliq,
- autorski algorytm ADX oparty na indukcji reguł decyzyjnych.

Można się spodziewać, że rankingi J48 i Sliq będą bardziej do siebie podobne i oba odleglejsze od rankingu otrzymanego z ADX, z uwagi na podobieństwo w podejściu oraz jednakowo zdefiniowaną miarę RI_{g_k} . Miara RI_{g_k} dla ADX jest już nieco odmienna (patrz 6.10) z uwagi na odmienność modelu predykcyjnego, lecz zbudowano ją w oparciu o identyczny koncept.

Parametry dla metody MCFS w przypadku każdego klasyfikatora były identyczne:

- $s = 3000$,
- $t = 10$,
- $m = 300$,
- $u = v = 1$,

- $splitRatio = 0.66$.

Ranking cech dla klasyfikatora J48

Dla klasyfikatora J48 użyto miary RI zdefiniowanej równaniem (6.9). Parametry algorytmu były domyślne – opisane we wstępie sekcji poświęconej eksperymentom selekcji cech (7.2).

1. GENE1622X	16. GENE669X	31. GENE2404X
2. GENE1602X	17. GENE588X	32. GENE1632X
3. GENE1613X	18. GENE537X	33. GENE584X
4. GENE1553X	19. GENE639X	34. GENE834X
5. GENE530X	20. GENE2553X	35. GENE1603X
6. GENE1610X	21. GENE542X	36. GENE1648X
7. GENE1647X	22. GENE1673X	37. GENE849X
8. GENE653X	23. GENE685X	38. GENE642X
9. GENE1606X	24. GENE454X	39. GENE1662X
10. GENE2426X	25. GENE844X	40. GENE524X
11. GENE622X	26. GENE640X	41. GENE694X
12. GENE2402X	27. GENE2368X	42. GENE617X
13. GENE1661X	28. GENE1607X	43. GENE771X
14. GENE598X	29. GENE1635X	44. GENE1672X
15. GENE2668X	30. GENE1605X	45. GENE1611X

Tablica 7.10. 45 najwyżej ocenionych cech przy użyciu J48 – dane chłoniak (Alizadeh)

1. X95735_at	16. M16038_at	31. J04615_at
2. M27891_at	17. M98399_s_at	32. X62320_at
3. M31166_at	18. M27783_s_at	33. D26308_at
4. M55150_at	19. D14874_at	34. X90858_at
5. D88422_at	20. Y12670_at	35. X74262_at
6. M54995_at	21. M84526_at	36. U37055_rna1_s_at
7. M23197_at	22. M92287_at	37. L47738_at
8. M77142_at	23. M31303_rna1_at	38. L08177_at
9. U46499_at	24. X62654_rna1_at	39. X87613_at
10. U02020_at	25. M31523_at	40. M91432_at
11. U50136_rna1_at	26. U46751_at	41. J04990_at
12. X70297_at	27. M83652_s_at	42. M29540_at
13. M21551_rna1_at	28. U22376_cds2_s_at	43. U85767_at
14. L09209_s_at	29. J05243_at	44. M12959_s_at
15. M81933_at	30. D49950_at	45. M11722_at

Tablica 7.11. 45 najwyżej ocenionych cech przy użyciu J48 – dane białaczka (Golub)

Ranking cech dla klasyfikatora Sliq

Podobnie jak dla klasyfikatora J48 miarą oceny jakości atrybutu była miara RI zdefiniowana w 6.9. Algorytm uruchamiano przy ustawieniach domyślnych t.j. $mdlPruning = false$.

1. GENE1602X	16. GENE2368X	31. GENE2345X
2. GENE1622X	17. GENE1141X	32. GENE1599X
3. GENE530X	18. GENE2346X	33. GENE3385X
4. GENE653X	19. GENE625X	34. GENE1635X
5. GENE622X	20. GENE3117X	35. GENE2190X
6. GENE833X	21. GENE2360X	36. GENE1140X
7. GENE834X	22. GENE1607X	37. GENE765X
8. GENE1553X	23. GENE2110X	38. GENE1606X
9. GENE598X	24. GENE646X	39. GENE2404X
10. GENE1610X	25. GENE3520X	40. GENE1603X
11. GENE1613X	26. GENE3519X	41. GENE1601X
12. GENE1647X	27. GENE626X	42. GENE3384X
13. GENE642X	28. GENE816X	43. GENE1648X
14. GENE2402X	29. GENE611X	44. GENE588X
15. GENE1661X	30. GENE1632X	45. GENE1637X

Tablica 7.12. 45 najwyżej ocenionych cech przy użyciu Sliq – dane chłoniak (Alizadeh)

1. X95735_at	16. M92287_at	31. M21551_rna1_at
2. M27891_at	17. U02020_at	32. X59417_at
3. M55150_at	18. M27783_s_at	33. M83652_s_at
4. M31166_at	19. M91432_at	34. M63138_at
5. X70297_at	20. U22376_cds2_s_at	35. X74262_at
6. U46499_at	21. M31303_rna1_at	36. L47738_at
7. D14874_at	22. Y12670_at	37. HG4321-HT4591_at
8. D88422_at	23. J04615_at	38. U12471_cds1_at
9. M77142_at	24. J05243_at	39. X90858_at
10. M23197_at	25. M31523_at	40. D49950_at
11. L09209_s_at	26. X62654_rna1_at	41. M12959_s_at
12. M16038_at	27. M84526_at	42. D10495_at
13. U50136_rna1_at	28. AFFX-HUMTFRR/M11507_3_at	43. M11722_at
14. M54995_at	29. X04085_rna1_at	44. U62136_at
15. M81933_at	30. U46751_at	45. D26308_at

Tablica 7.13. 45 najwyżej ocenionych cech przy użyciu Sliq – dane białaczka (Golub)

Ranking cech dla klasyfikatora ADX

Ranking dla algorytmu ADX jest budowany w oparciu o miarę RI zdefiniowaną równaniem 6.10. Klasyfikator ADX sparametryzowano zgodnie z opisem we wstępie sekcji 7.2, oraz użyto wbudowanej dyskretyzacji.

1. GENE1622X	16. GENE653X	31. GENE1645X
2. GENE1618X	17. GENE766X	32. GENE1607X
3. GENE1617X	18. GENE1610X	33. GENE1641X
4. GENE1602X	19. GENE622X	34. GENE1625X
5. GENE659X	20. GENE530X	35. GENE1649X
6. GENE1744X	21. GENE1553X	36. GENE1640X
7. GENE1619X	22. GENE1632X	37. GENE1627X
8. GENE1637X	23. GENE735X	38. GENE588X
9. GENE1636X	24. GENE1613X	39. GENE2368X
10. GENE1616X	25. GENE1634X	40. GENE1663X
11. GENE1644X	26. GENE1648X	41. GENE2402X
12. GENE1702X	27. GENE1635X	42. GENE675X
13. GENE1662X	28. GENE712X	43. GENE598X
14. GENE1647X	29. GENE1643X	44. GENE1603X
15. GENE1661X	30. GENE834X	45. GENE633X

Tablica 7.14. 45 najwyżej ocenionych cech przy użyciu ADX – dane chłoniak (Alizadeh)

1. X95735_at	16. D14874_at	31. X59417_at
2. M55150_at	17. U22376_cds2_s_at	32. X90858_at
3. M31166_at	18. M92287_at	33. U12471_cds1_at
4. M27891_at	19. X62654_rna1_at	34. X04085_rna1_at
5. M77142_at	20. U02020_at	35. U46751_at
6. D88422_at	21. X74262_at	36. U09087_s_at
7. X70297_at	22. M81933_at	37. J04615_at
8. U46499_at	23. M31523_at	38. M83652_s_at
9. U50136_rna1_at	24. U41813_at	39. U73737_at
10. M27783_s_at	25. M12959_s_at	40. L47738_at
11. M91432_at	26. M28130_rna1_s_at	41. X80907_at
12. L09209_s_at	27. U85767_at	42. X74801_at
13. M23197_at	28. J03930_at	43. X07743_at
14. M16038_at	29. M84526_at	44. M54995_at
15. Y12670_at	30. U62136_at	45. J05243_at

Tablica 7.15. 45 najwyżej ocenionych cech przy użyciu ADX – dane białaczka (Golub)

Z uwagi na budowę osobnych zestawów reguł (ruleset) w ADX dla każdej z wartości zmiennej decyzyjnej, dodatkowo w MCFS buduje się osobne rankingi cech dla poszczególnych klas. Każdy z takich rankingów jest tworzony w

oparciu o odpowiedni zestaw reguł zbudowany osobno dla klasy. Pozwala to w konsekwencji na osobną analizę atrybutów istotnych, takich które pozwalają na separację klasy względem pozostałych przykładów. W przypadku drzewa klasyfikacyjnego by otrzymać podobny efekt zastosowano odpowiednie modyfikacje w podejściu do MCFS (patrz sekcja 7.2.4) lecz otrzymane osobne rankingi są wynikiem modyfikacji MCSF a nie cechą charakterystyczną klasyfikatora toteż porównanie klasyfikatorów użytych w MCFS ograniczymy tylko do łącznego rankingu.

Porównanie wysoko ocenionych cech

Poszczególne pary zbiorów najwyżej ocenionych atrybutów zawierają około 50% wspólnych cech w przypadku danych Alizadeha i 70-80% dla danych Goluba (tabele 7.16 i 7.17). Rankingi otrzymane przy pomocy Sliq i J48 są nieznacznie bardziej podobne do siebie nawzajem, niż którykolwiek z nich do rankingu zbudowanego w oparciu o algorytm ADX. Co ciekawe, zróżnicowanie to jest stosunkowo niewielkie i wskazuje na grupę informatywnych cech wybieranych bez względu na zastosowany klasyfikator.

algorytm	J48	Sliq	ADX
J48	x	23	21
Sliq	44	x	21
ADX	41	39	x

Tablica 7.16. Porównanie liczby wspólnych cech w odpowiednich parach rankingów (prawa górna część 45 cech, lewa dolna 90 cech) – dane chłoniak (Alizadeh)

algorytm	J48	Sliq	ADX
J48	x	37	33
Sliq	73	x	36
ADX	65	69	x

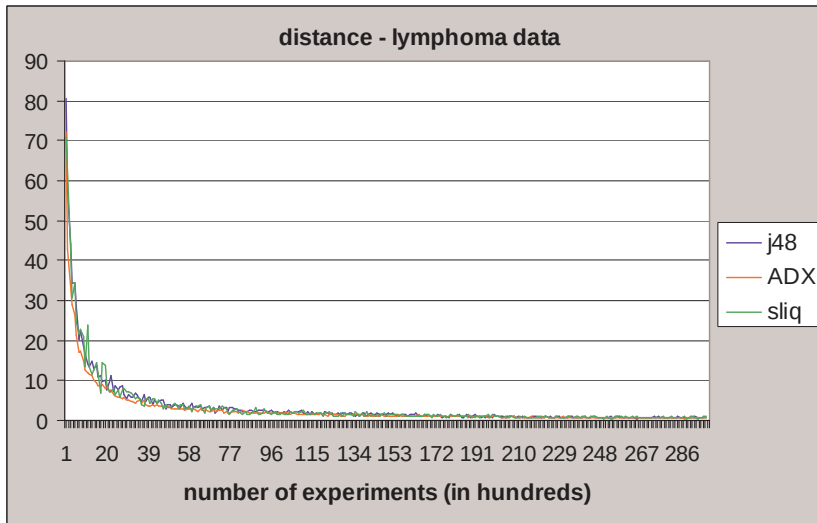
Tablica 7.17. Porównanie liczby wspólnych cech w odpowiednich parach rankingów (prawa górna część 45 cech, lewa dolna 90 cech) – dane białaczka (Golub)

Procentowy udział części wspólnych w poszczególnych rankingach właściwie nie zależy od wielkości analizowanych czołówek rankingów, co wskazuje na duże podobieństwo tych zbiorów mimo pewnego zróżnicowania w kolejności genów. Analizując czołówki rankingów np. dla J48 i ADX o wielkości 45 i 90 dla danych Goluba, można zauważyć że udział wspólnych genów wynosi odpowiednio 73% i 72%. Patrząc na problem z perspektywy tysięcy atrybutów poddawanych ocenie przez MCFS, finalny wybór 50 do 80%

jednakowych cech w czołówkach badanych rankingów pozwala stwierdzić, że algorytm MCFS jest w dużej mierze niezależny od zastosowanego typu klasyfikatora. Występujące naturalne różnice w poszczególnych rankingach są związane z odmiennym działaniem i sposobem oceny atrybutów, niemniej jednak w ogromnej masie ocenianych cech uwidacznia się silna tendencja stosunkowo wysokiej oceny tych samych genów w różnych rankingach (gen *X95735_at* znajduje się na pierwszym miejscu wszystkich rankingów dla zbioru danych Goluba, natomiast *GENE1622X* dla zbioru Alizadeha w dwóch rankingach zajmuje pozycję pierwszą a w przypadku Sliq drugą).

Porównanie zbieżności MCFS

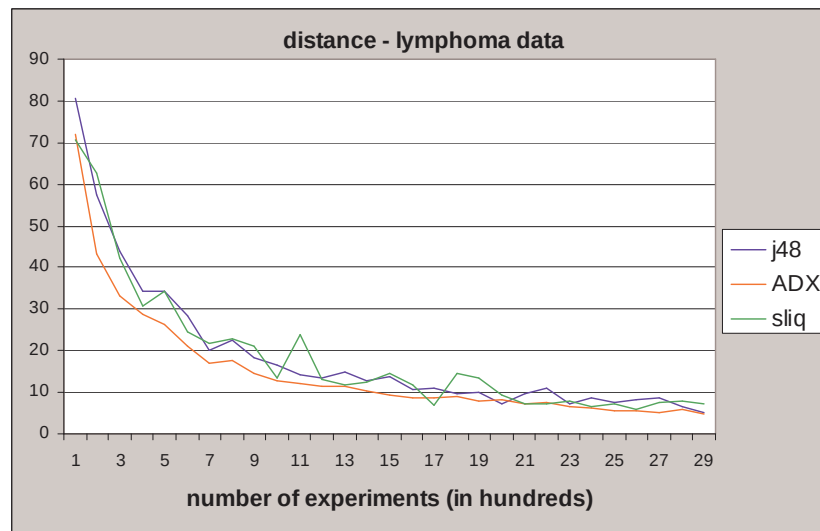
Co każde 10 kolejnych wartości s liczone dystans otrzymanych rankingów cech (6.12). Przy ustalonym $t = 10$ wartość rankingu zapisywano zatem co 100 pojedynczych eksperymentów (budowa i test klasyfikatora), a następnie porównano poszczególne wykresy dla wszystkich użytych algorytmów klasyfikacji. Rysunek 7.53 prezentuje wykresy wartości dystansów w pełnym zakresie s natomiast 7.54 prezentuje tylko początkowy jego fragment (do $s = 300$). Za każdym razem porównywano 10% najwyżej ocenionych cech.



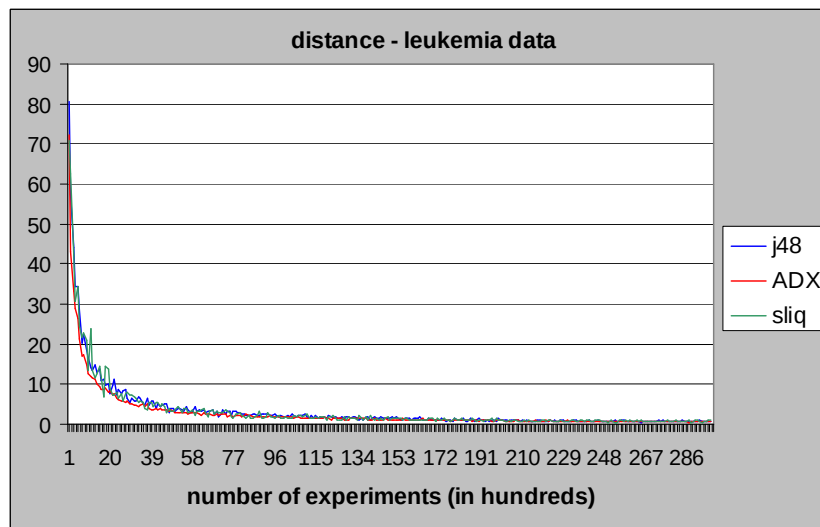
Rysunek 7.53. Porównanie dystansów rankingów $s = 3000$ $t = 10$ – dane chłoniak (Alizadeh)

Dla obu zbiorów danych mimo bardzo podobnych ogólnych wykresów dystansu (bez względu na klasyfikator) uwidacznia się jeden dość istotny szczegół. Dla klasyfikatora ADX przebieg krzywej dystansu jest wyraźnie gładniejszy. Dzięki temu w przeciągu mniejszej liczby s można stwierdzić ustalenie się finalnego rankingu, ponieważ dystans nie zmienia się w sposób tak gwałtowny i oscylacyjny jak to ma miejsce w przypadku pozostałych algorytmów.

Większa stabilność rankingów wraz ze zwiększaniem s wynika z charakteru generalizacji wiedzy dokonywanej przez poszczególne klasyfikatory. Drzewa klasyfikacyjne dokonują możliwie mało "cięć" wielowymiarowej przestrzeni i starają się stworzyć minimalny potrzebny opis. W przypadku rozważanych danych drzewa mają przeważnie od 2 do 4,5 węzłów, co wpływa na drobne lecz zauważalne wahania dystansów. Pojedyncze użycie atrybutu w



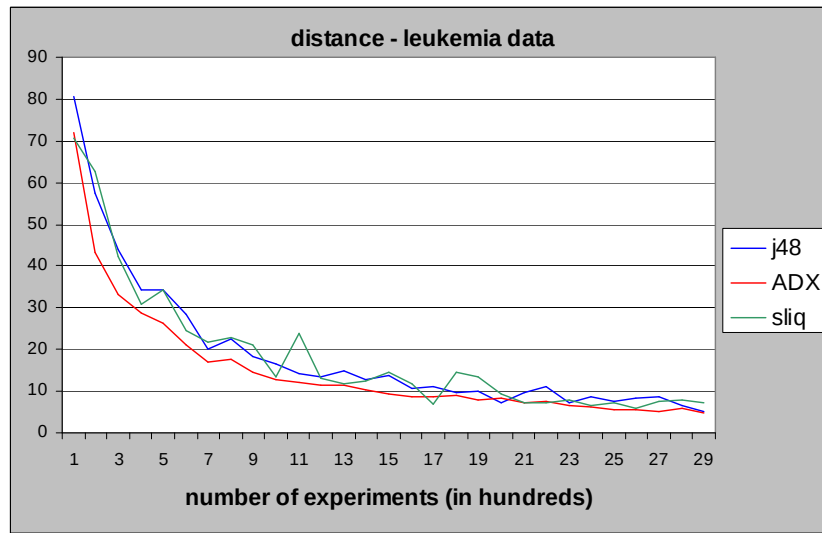
Rysunek 7.54. Porównanie dystansów rankingów $s = 300$ $t = 10$ – dane chłoniak (Alizadeh)



Rysunek 7.55. Porównanie dystansów rankingów $s = 3000$ $t = 10$ – dane białaczka (Golub)

węzle drzewa klasyfikacyjnego może w konsekwencji znacznie przesunąć w rankingu taki atrybut.

W ADX z uwagi na pewną dopuszczalną nadmiarowość opisu danych trenujących, w finalnym zestawie reguł ma szansę znaleźć się więcej pomocnych w klasyfikacji cech, co z kolei wpływa korzystnie na stabilizację rankingu. Minimalny zestaw reguł pokrywający zbiór trenujący może nie przechowywać wystarczającej informacji służącej do poprawnej klasyfikacji nieznanych przykładów. W przeciwieństwie do drzew decyzyjnych gdzie stawia się na minimalizację opisu, wspomniane właściwości klasyfikatora regułowego ADX pozwalają na osiągnięcie lepszych wyników.



Rysunek 7.56. Porównanie dystansów rankingów $s = 300$ $t = 10$ – dane białaczka (Golub)

7.2.7. Walidacja metody MCFS

Zgodnie z opisem w sekcji 6.4 dla obu badanych zbiorów przeprowadzono eksperyment walidacji wyników (rankingów) tzn. losowo wydzielono podzbiór przykładów do weryfikacji, a MCFS uruchomiono na pozostałych danych. Zbiory te odpowiednio nazwiemy *potwierdzającym* i *walidacyjnym* (patrz rys 6.4). W wyniku uruchomienia procedury MCFS na zbiorze walidacyjnym otrzymano odpowiednie rankingi cech według miary RI_{g_k} . Dla danych opisujących białaczkę do zbioru walidacyjnego losowo wydzielono 20 przykładów z klasy ALL oraz 8 z AML. Dla drugiego zbioru danych (chłoniak) wydzielono 26 przykładów z DLCL, 7 z FL i 9 z klasy CLL.

Dla obu zbiorów procedura MCFS została uruchomiona dla trzech klasyfikatorów z następującymi parametrami:

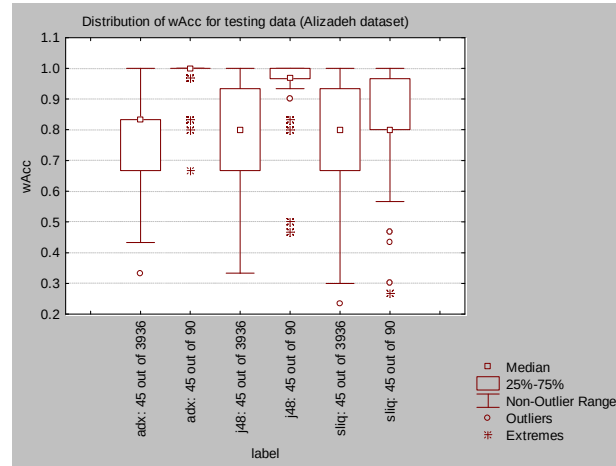
- $s = 3000$,
- $t = 10$,
- $m = 300$,
- $u = v = 1$,
- $splitRatio = 0.66$.

Parametry poszczególnych klasyfikatorów pozostawiono bez zmian do opisu podanego na początku sekcji 7.2, a klasyfikator ADX uruchamiano z włączoną dyskretyzacją wstępną.

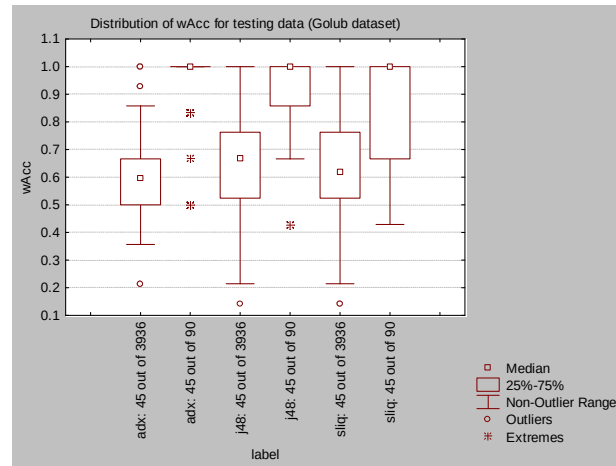
Należy zwrócić uwagę tutaj, iż w celu selekcji informatywnych genów w oparciu o zbiór walidacyjny budujemy znowu tysiące klasyfikatorów na zbiorach trenujących i oceniamy je na testujących. Oznacza to, że sam akt selekcji cech ma już wbudowaną walidację na nieznanych danych, a atrybut jest tym lepszy im wyższa jakość predykcji osiągnął klasyfikator oparty na nim. Przeprowadzony eksperyment ma jednak za zadanie udowodnić poprawne działanie metody dla kompletnie odseparowanych przykładów.

W kolejnej fazie eksperymentu nazwanej 2A – spośród najwyżżej ocenionych $2b$ cech losujemy b (t.j. połowę) i budujemy na nich klasyfikator (na zbiorze walidacyjnym), a wynik jego predykcji na zbiorze potwierdzającym zachowujemy. Dla obu badanych zbiorów wybierano 45 spośród 90. Dla każdego zbioru w fazie 2A przeprowadzono 1000 klasyfikacji, a otrzymane rozkłady porównano z wynikami otrzymanymi z odpowiednich eksperymentów 2B. Faza 2B jest analogiczna do 2A z tą różnicą, że losujemy b cech spośród $d-2b$, czyli tych znacznie mniej informatywnych według algorytmu MCFS.

Porównanie histogramów otrzymanych wartości $wAcc$ dla faz 2A i 2B - dla trzech klasyfikatorów - znajduje się na rysunkach 7.58 i 7.57.



Rysunek 7.57. Porównanie rozkładów $wAcc$ w kroku potwierdzającym: dane chłoniak (Alizadeh)



Rysunek 7.58. Porównanie rozkładów $wAcc$ w kroku potwierdzającym: dane białaczka (Golub)

Bez względu na zbiór danych oraz typ zastosowanego klasyfikatora dla losowo wybieranych 45 genów spośród 90 najwyżżej ocenionych przez MCFS pozwala na uzyskanie znacznie lepszego wyniku predykcji. Dla wszystkich rozkładów uwidacznia się wyraźne przesunięcie mediany oraz całego rozkładu

wartości wAcc w kierunku wyższych wartości, przy czym rozkłady ścieśniają się (mają niższą wariancję). Oznacza to, iż przy pomocy algorytmu MCFS udało się wyselekcjonować istotne cechy i na ich podstawie klasyfikować nieznane przykłady zdecydowanie lepiej niż na podstawie innego losowo wybranego zestawu atrybutów. Warto podkreślić, że niezwykle korzystną z selekcji atrybutów czerpie algorytm ADX, który właściwie dla obu zbiorów osiąga prawie 100% poprawnych klasyfikacji (mediana oraz kwartyle 1 i 3 są na poziomie 1). Mimo, iż oba pozostałe algorytmy również dowodzą prawidłowości wyboru istotnych cech w MCFS to w przypadku ADX pozwala to na budowę niezawodnego klasyfikatora.

7.2.8. Konkluzja wyników i weryfikacja biologiczna

Podsumowując dotychczasowe rozważania i sposoby walidacji (sekcja 7.2.7) można jednoznacznie stwierdzić że metoda MCFS pozwala na uzyskanie obiektywnego rankingu cech weryfikowalnego na nieznanymi danych. W takim wypadku można zatem przeprowadzić proces z udziałem wszystkich przykładów by otrzymać bardziej adekwatne rankingi wobec niewielkiej liczby dostępnych obiektów.

Dla danych opisujących przykłady białaczki (dane Goluba) 30 najbardziej informatywnych genów otrzymanych z MCFS na pełnej próbie to:

1. X95735_at	11. M54995_at	21. M16038_at
2. M31166_at	12. U02020_at	22. D14874_at
3. M27891_at	13. M77142_at	23. M84526_at
4. M55150_at	14. M81933_at	24. M92287_at
5. D88422_at	15. X70297_at	25. M31523_at
6. M23197_at	16. Y12670_at	26. X62654_rna1_at
7. M98399_s_at	17. U46499_at	27. M31303_rna1_at
8. U50136_rna1_at	18. M83652_s_at	28. D49950_at
9. M21551_rna1_at	19. U46751_at	29. U22376_cds2_s_at
10. M27783_s_at	20. L09209_s_at	30. J05243_at

Warto porównać wynik otrzymany z MCFS z genami wybranymi przez Dudoit et al. w pracach [Dudoit *et al.*, 2002] i [Dudoit *et al.*, 2003], gdzie poszukiwano genów nadekspresyjnych przy pomocy wielokrotnego testowania hipotez statystycznych. Co ciekawe, lista podana przez Dudoit et al. oraz lista z pracy [Golub *et al.*, 1999] w istotnej części są podobne. W pracy [Dudoit *et al.*, 2002] wybrano 32 nadekspresywne geny dla klasy AML i 60 dla klasy ALL. Należy tutaj zwrócić uwagę, iż nadekspresywność genu nie musi determinować jego istotności w procesie klasyfikacji i dlatego lista nadekspresyjnych genów nie musi się silnie pokrywać z rankingiem otrzymanym z MCFS. W szczególności gen, przy pomocy którego można dokonać doskonałej klasyfikacji nie musi charakteryzować nadekspresywnością dla żadnej z klas. Przykładowo, spośród 30 najwyżej ocenionych genów przez MCFS istnieje 12, które nie są nadekspresywne i nie znajdują się na liście Dudoit.

W tabeli 7.18 zaprezentowano możliwości predykcyjne (mierzone przy pomocy *wAcc*) 12 genów spośród najwyżej ocenionych 30 przez MCFS w dwóch sytuacjach – 12 najwyżej ocenionych przez MCFS wśród nie uznanych za na-

dekspresywne i 12 najwyżej ocenionych przez MCFS spośród genów uznanych za nadekspresywne takie przez Dudoit (dla dowolnej klasy). Dla obu zestawu 12 atrybutów przeprowadzono 10 krotnie eksperyment walidacji krzyżowej 10 zbiorowej, a w tabeli podano średni wynik $wAcc$. Następujące klasyfikatory użyte w eksperymencie pochodzą z implementacji WEKA i wszystkie uruchomiono przy użyciu domyślnych parametrów: C4.5 (J48), kNN (1NN), Naive Bayes, Random Forest i Support Vector Machine (SVM).

	J48	1NN	NB	RF	SVM
nie nad-ekspresywne	0,891	0,954	1	0.903	0.914
nad-ekspresywne	0.927	0.936	0.981	0.919	0.936

Tablica 7.18. Porównanie jakości klasyfikacji genów nad-ekspresywnych i pozostałych wybranych przez MCFS – dane białaczka (Golub)

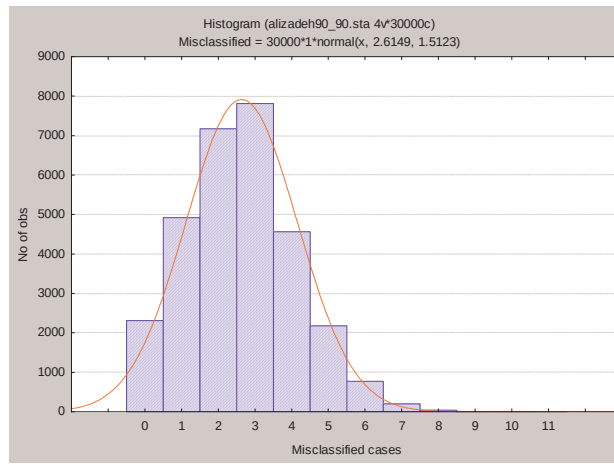
Bez wątpienia nadekspresywność genu nie jest potrzebna do poprawnej klasyfikacji. Ponadto, przy pomocy MCFS można dodatkowo wskazać pozostałe geny, te które nie są silnie nadekspresywne ale skutecznie rozróżniają klasy.

Finalny ranking 30 najwyżej ocenionych cech dla uzupełnionych pełnych danych z przykładu Alizadeh et al., jest następujący:

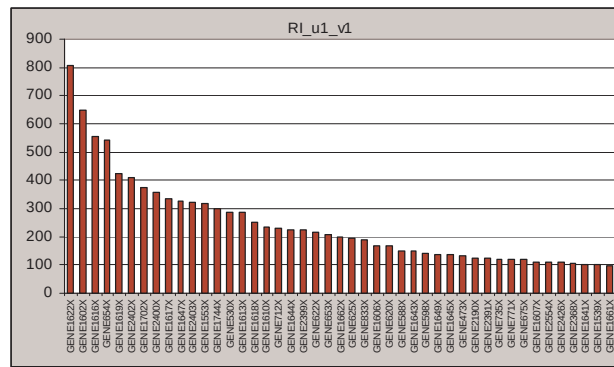
1. GENE1622X	11. GENE2403X	21. GENE622X
2. GENE1602X	12. GENE1553X	22. GENE653X
3. GENE1616X	13. GENE1744X	23. GENE1662X
4. GENE654X	14. GENE530X	24. GENE625X
5. GENE1619X	15. GENE1613X	25. GENE833X
6. GENE2402X	16. GENE1618X	26. GENE1606X
7. GENE1702X	17. GENE1610X	27. GENE620X
8. GENE2400X	18. GENE712X	28. GENE588X
9. GENE1617X	19. GENE1644X	29. GENE1643X
10. GENE1647X	20. GENE2399X	30. GENE598X

W przypadku zbioru danych Alizadeh et al. ranking pochodzący z MCFS można porównać ze zbiorem istotnych genów podanych przez Tibshiraniego i in. [Tibshirani *et al.*, 2003]. Autorzy w tej pracy przy pomocy metody *nearest shrunken centroids* wskazali finalnie 81 genów jako istotne w procesie klasyfikacji. Na liście 81 najwyżej ocenionych cech przy pomocy MCFS znajduje się jednak tylko 29 genów z listy Tibshiraniego, natomiast spośród 300 pierwszych w rankingu znajduje się 48 z listy. W [Tibshirani *et al.*, 2003] 59 przykładów (3 spośród 62 zostały odrzucone jako silnie odstające) podzielono na zbiór trenujący (39 przypadków) i testujący (20 przypadków) i w oparciu o 81 wybranych cech dokonano bezbłędnej predykcji.

W celu porównania obu list zbudowano 30000 drzew na losowo wybranych 42 przykładach, a następnie dokonano predykcji klasy dla pozostałych 20 przypadków. Za każdym razem klasyfikator budowano na losowym podzbiorze 45 genów z 90 najwyżej rozmieszczonych w rankingu. W rezultacie



Rysunek 7.59. Histogram liczby błędnie zaklasyfikowanych przykładów w oparciu o losowo wybrane 45 genów z 90 najwyżżej ocenionych: dane chłoniak - Alizadeh

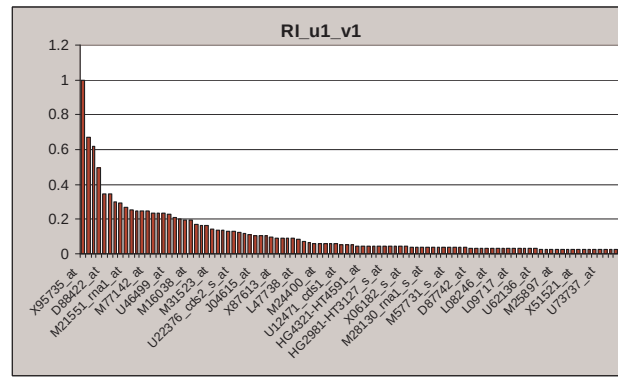


Rysunek 7.60. Relatywna istotność dla 45 najwyżżej ocenionych genów: dane chłoniak

otrzymano w 8% drzew bezbłędną klasyfikację, w około 75% maksymalnie 3 błędy (patrz histogram 7.59).

Wartości relatywnej istotności zostały przedstawione na wykresach słupkowych 7.60 dla uzupełnionych danych Alizadeha i 7.61 dla danych Goluba. Na obu wykresach widoczna jest typowa charakterystyka RI_{g_k} - poza nieliczną grupą najbardziej istotnych cech reszta wartości jest raczej płaska (wykres istotności również dla zupełnie innych zbiorów danych wygląda podobnie jak $f(x) = \frac{1}{x}$).

W ramach badań niezależności metody od techniki klasyfikacji dokonano eksperymentu bazującego na porównaniu jakości klasyfikacji innych niż drzewo decyzyjne algorytmów w oparciu o cechy wybrane przez MCFS. Do badań użyto implementacji klasyfikatorów 1NN, NB, Random Forest i SVM (WEKA 3.4.10). W eksperymencie porównano mediany $wAcc$ osiągniętego w 100 kolejnych losowaniach: 45 genów z 90 najlepszych i 45 z pozostałych (Alizadeh); 100 genów z 200 najlepszych i 100 z pozostałych (Golub). W tabeli 7.19 znajduje się stosowne porównanie jakości klasyfikacji różnych technik. Dla każdego z klasyfikatorów średnie $wAcc$ predykcji uzyskane na genach wyselekcjonowanych przez MCFS przyjmuje wartości istotnie większe.



Rysunek 7.61. Relatywna istotność dla 100 najwyżej ocenionych genów: dane białaczka

	Alizadeh et al.		Golub et al.	
	45 out of 90	45 out of 3936	100 out of 200	100 out of 6930
J48	0.93	0.78	0.86	0.71
1NN	1	0.93	0.93	0.78
NB	1	0.71	0.93	0.86
RF	1	0.86	0.93	0.78
SVM	1	0.93	0.93	0.86

Tablica 7.19. Porównanie jakości klasyfikacji różnych technik klasyfikacji w oparciu o geny wybrane przez MCFS

Zainteresowani wnikliwą dyskusją o charakterze biologicznym na temat wyników powinni sięgnąć do pracy [Dramiński *et al.*, 2008], w rozprawie jedynie w bardzo ogólnym ujęciu zostanie scharakteryzowany rezultat badań. Walidacja genetyczna została przeprowadzona przy pomocy badań literaturowych oraz bazy Ontologii Genów (Gene Ontology (GO) [Gene Ontology]) dla wyników otrzymanych dla zbioru danych Goluba (dane białaczka) oraz opublikowanych przez Dudoit *et al.* Grupy genów będą odpowiednio oznaczone MCFS oraz Dudoit.

Najwyżej ocenione geny zostały podzielone na trzy grupy: geny występujące jedynie w MCFS, jedynie w Dudoit, oraz geny występujące w obu zbiorach. Po takim podziale na grupy przeprowadzono osobną analizę GO dla każdej z grup. Wynik dla grupy Dudoit wydaje się bardziej związany z rakiem – geny z tej grupy należą do kategorii związanych z np. rekombinacją DNA czy reperacją DNA, przy czym w różny sposób dotyczą różnych typów raka. Niekiedy kilka z tych genów wiązano jedynie z przypadkiem ALL ale już nie z AML i na odwrót.

Patrząc na wynik analizy GO dla MCFS można odnieść wrażenie, że geny te są mniej związane z rakiem. Jest w tej grupie jednak wyraźna nadreprezentacja genów z kategorii związanej z odpowiedzią immunologiczną i obroną przed bodźcami biotycznymi (*biotic stimuli*) np. odpowiedź na działanie obecnej bakterii czy zranienie. Co najbardziej interesujące, to geny wybrane przy pomocy MCFS związane są z dwoma różnymi typami komórek systemu im-

munologicznego (makrofagi, granulocyty związane z wrodzonym systemem obronnym i limfocyty związane bardziej z adaptywną częścią systemu obronnego organizmu), które to atakowane są oddzielnie przez dwa typy badanej białaczki. Typ białaczki AML atakuje granulocyty natomiast w przypadku ALL komórki rakowe powstają na bazie limfocytów. Można zatem wyróżnić osobne grupy genów, które mogą pomóc w odróżnianiu obu typów białaczki od siebie.

Konkludując, metoda Dudoit et al., która wybiera geny w oparciu o silne zróżnicowanie pomiędzy poziomami ekspresji w klasach AML i ALL, pozwala na selekcję typowych genów związanych ogólnie z różnymi typami raka i odpowiedzialnych za cykl komórkowy i reperację DNA. Z drugiej strony MCFS pozwala na wybór genów bardziej związanych z genezą choroby i odpowiedzią komórek układu odpornościowego atakowanych przez różne odmiany białaczki, a tym samym umożliwia odróżnienie obu typów od siebie. Można mieć nadzieję, że metoda oparta na predykcji (MCFS) w większym stopniu pozwala na wyodrębnienie genów silniej powiązanych ze źródłem choroby niż jej ogólnym przebiegiem. Tak przynajmniej jest we właśnie przeanalizowanym przykładzie.

8. Implementacja ADX i MCFS w bibliotece dmLab

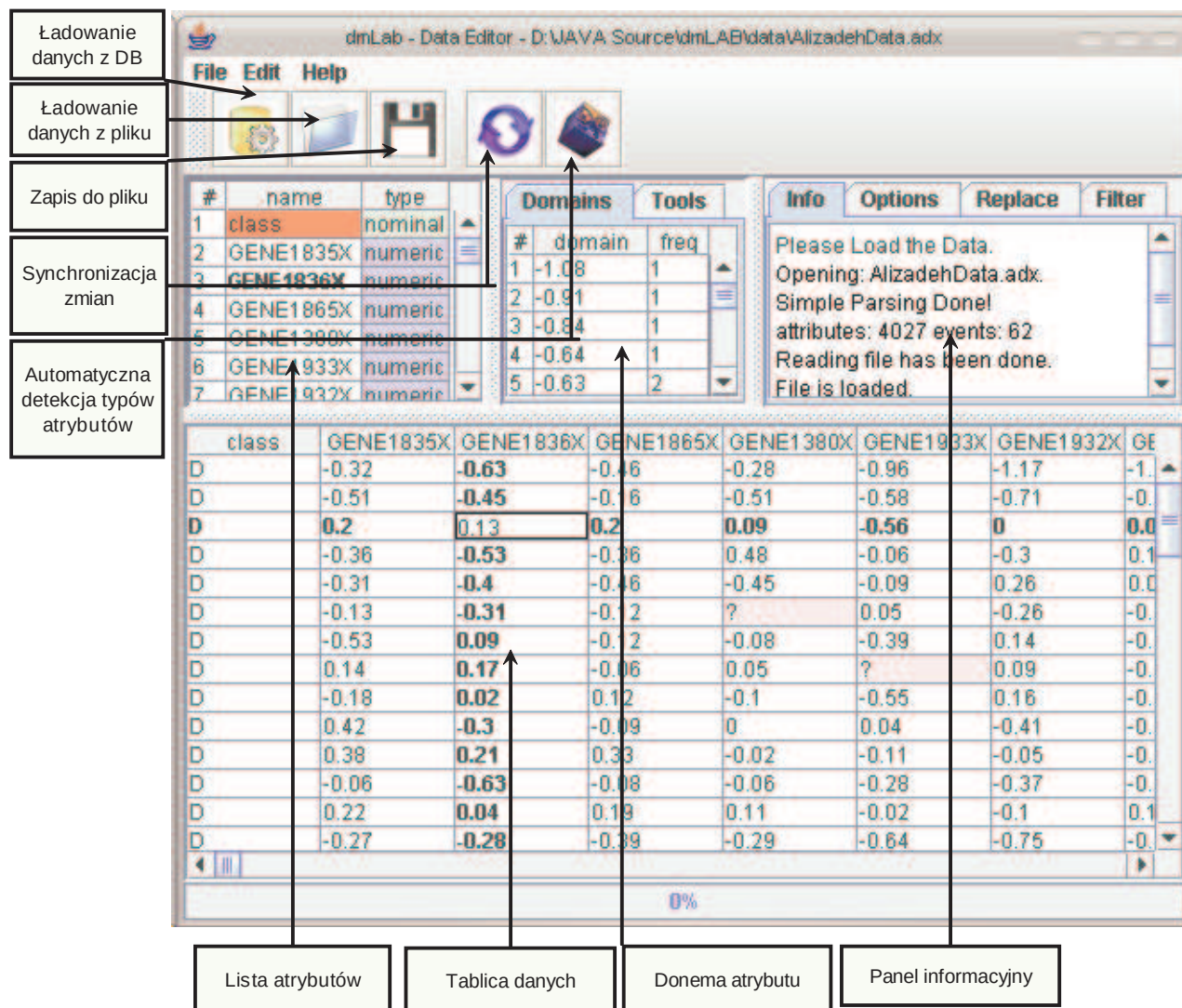
Implementacja biblioteki dmLab powstała w języku JAVA w wersji 5.0. Poszczególne składowe systemy uruchamia się poprzez wystartowanie odpowiednich klas. Biblioteka dmLab w wersji produkcyjnej dostarczona jest jako jeden plik typu *jar* z tekstowymi plikami konfiguracyjnymi i plikami służącymi do bezpośredniego uruchomienia poszczególnych modułów (pliki z rozszerzeniem *bat*). Eksperyment klasyfikacji uruchamia się poprzez plik *run_Classification.bat* a MCFS przez *run_MCFS.bat*. Pliki konfiguracyjne eksperymentów mają rozszerzenie **.run*, a parametry algorytmów w plikach **.cfg*. We wszystkich plikach konfiguracyjnych znak '#' oznacza komentarz i wszystkie znaki występujące po nim są ignorowane.

8.1. Edytor danych

Edytor danych uruchamia się plikiem *editor.bat* i za jego pomocą można w dość prosty sposób przygotować i/lub konwertować dane wejściowe dla modułów klasyfikacji i MCFS. Program jest w stanie edytować dane o dowolnej liczbie kolumn i wierszy, a jedynym ograniczeniem jest pamięć komputera na którym uruchomiono edytor. Czołowy panel aplikacji przedstawiono na rysunku 8.1.

Edytor danych w dmLab pozwala na:

- Wczytanie danych z plików: csv, arff (format WEKA), adx (format dmLab).
- Definicję wejściowego formatu danych w postaci csv np. czy pierwsza linia zawiera nazwy atrybutów, określenie separatora pól etc..
- Wczytanie danych z bazy danych typu: MS SQL Server, MySQL, Postgres, Oracle.
- Zapis do plików: csv, arff (format WEKA), adx (format dmLab), dta/var (format INLEN 3.1).
- Automatyczne rozpoznanie typu atrybutu: nominal, integer, numeric.
- Edytowanie atrybutów: zmianę nazwy, określenie ręczne typu, usunięcie, dodanie, przesunięcie w dowolną stronę.
- Edytowanie automatyczne wybranych atrybutów: zamiana ciągu znaków lub zamiana zdefiniowanej wartości.
- Edytowanie ręczne danych: zmianę wartości, zamianę ciągu znaków dla określonej listy atrybutów.
- Filtrowanie wierszy przy spełnieniu zdefiniowanego warunku logicznego nałożonego na atrybut.
- Podgląd dziedziny atrybutu wraz z informacją o licznosciach poszczególnych wartości.



Rysunek 8.1. Panel aplikacji edytora danych

8.2. Parametry wejściowe ADX

Plik który definiuje odpowiednie parametry potrzebne do uruchomienia eksperymentu klasyfikacji to *classification.run*. Pozwala on wybrać algorytm klasyfikatora (adx, j48, sliq) oraz zdefiniować warunki eksperymentu takie jak: typ walidacji (krzyżowa, podział losowy, zdefiniowany zbiór testowy), ilość powtórzeń, włączenie dyskretyzacji, katalogi i nazwy plików danych.

Parametry algorytmu klasyfikacji znajdują się w pliku o odpowiadającej mu nazwie (np. *adx.cfg*). Poniższa lista opisuje wszystkie parametry sterujące implementacją algorytmu ADX. Podane wartości są domyślne.

Parametry klasyfikatora ADX:

adx.verbose=false – wartość *true* powoduje wypisywanie dodatkowych informacji o działaniu metody

adx.debug=false – wartość *true* powoduje wypisywanie na ekran dodatkowych informacji pomocnych w procesie usuwania błędów.

- adx.searchBeam=50** – definiuje liczbę najlepszych kompleksów wybieranych jako rodziców
- adx.finalBeam=[50]** – definiuje maksymalną liczbę najlepszych kompleksów wybieranych do finalnego zbioru, liczby oddzielone przecinkiem [50,20] definiują osobno wartości dla każdej z klas (kolejność definicji jest identyczna z definicją klas w pliku wejściowym z danymi), pojedyncza liczba w nawiasie odnosi się do wszystkich klas
- adx.complexGenerality=0** – wartość 0 determinuje budowę nieco dłuższych kompleksów
- adx.qMethod=1** – definiuje metodę oceny jakości kompleksu dla etapu budowy kompleksów: 0 – miara $Q = p - n$; 1 – miara $Q = (p - n)(1 - n)$; 2 – miara $Q = (p - n)(1 - n)^2$; 3 – miara $Q = p$; 4 – miara $Q = n$; 5 – miara $Q = p(1 - n)$ (patrz sekcja 5.3)
- adx.qMethodFinal=2** – definiuje metodę oceny jakości kompleksu dla etapu wyboru finalnego zestawu, parametr przyjmuje identyczną listę możliwych wartości co parametr *qMethod*
- adx.selSignificantMethod=4** – metoda wyboru finalnego zbioru kompleksów: 0 – wybiera kompleksy gdzie $n > 0$; 1 – wybiera kompleksy gdzie $p > n$; 2 – wybiera kompleksy gdzie $q \geq qMin$; 3 – wybiera *finalBeam* kompleksów najwyżej ocenionych przez *qMethodFinal*; 4 – wybiera zestaw kompleksów maksymalizujący poprawną klasyfikację próbki danych trenujących (patrz sekcja 5.2.4)
- adx.qMin=0.1** – określa wartość progową jakości kompleksów wybieranych przy *selSignificantMethod* = 2
- adx.maxEventsForSelection=3000** – określa wielkość losowej próby przykładów trenujących przy *selSignificantMethod* = 4
- adx.scoreMethod=6** – określa miarę przynależności do klasy (patrz sekcja 5.4)
- adx.useSensitivity=0** – określa użycie czułości względem klasy: 0 – nie wymusza czułości; 1 – odpowiednie wartości czułości podaje użytkownik; 2 – odpowiednie wartości czułości są definiowane na podstawie zbioru trenującego i są równe prawdopodobieństwu wystąpienia klasy
- adx.sensitivityArray=[1]** – określa wektor czułości, kolejność klas taka jak w definicji pliku wejściowego z danymi (wartości z zakresu 0;1)
- adx.cleanCandidates=2** – określa usuwanie kompleksów kandydatów: 0 – brak usuwania; 1 – usuwa jeśli kompleks nie ma wyższego Q niż przynajmniej jeden z selektorów składowych; 2 – usuwa jeśli kompleks nie ma wyższego Q niż przynajmniej jeden z rodziców
- adx.mergeCondition=0** – określa sklejanie kompleksów: 0 – brak sklejania; 1 – sklejanie bezwarunkowe; 2 – sklejanie jeśli wyższe Q niż kompleksów bazowych

8.3. Parametry wejściowe MCFS

Plik który definiuje odpowiednie parametry potrzebne do uruchomienia metody to *MCFS.run*. Znak '#' oznacza komentarz i wszystkie znaki występujące po nim są ignorowane. Poniższa lista opisuje wszystkie parametry

sterujące MCFS. Podane wartości są domyślne i pozwalają na uniwersalne uruchomienie.

Parametry eksperymentu:

verbose=false – wartość *true* powoduje wypisywanie dodatkowych informacji o działaniu metody

debug=false – wartość *true* powoduje wypisywanie na ekran dodatkowych informacji pomocnych w procesie usuwania błędów

inputFilesPATH = ./data/ – definiuje ścieżkę do katalogu danych

tempFilesPATH = ./resources/ – definiuje ścieżkę do katalogu wyników działania

inputFileName=AlizadehData.adx – definiuje nazwę wejściowego pliku danych

testFileName=AlizadehData_test.adx – definiuje opcjonalną nazwę pliku danych testowych (działa dla trybu *allClassesAtOnce* i *removeBestAttributes=false*), jeśli parametr został zdefiniowany to nie wykonuje się wewnętrznej pętli (losowego podziału na zbiór trenujący i testujący)

Parametry MCFS:

mcfs.showDistanceProgress=true – wartość *true* powoduje wyświetlenie okna z bieżącym przebiegiem wartości dystansu dla eksperymentu

mcfs.histogramFileName=ranking.csv – definiuje nazwę pliku z rankingiem

mcfs.permFileName=permSummary.csv – definiuje nazwę pliku z podsumowaniem dla pojedynczej permutacji

mcfs.accFile=acc_wAcc.csv – definiuje nazwę pliku z kolejnymi wartościami *Acc* i *wAcc* otrzymywanymi podczas działania algorytmu

mcfs.distFileName=dist.csv – definiuje plik z kolejnymi wartościami dystansu

mcfs.deleteTMPFiles=true – wartość *true* powoduje usuwanie poszczególnych plików z danymi do uczenia i testowania klasyfikatorów

mcfs.discretizeData=false – wartość *true* powoduje włączenie dyskretyzacji poszczególnych zbiorów danych, przedziały dyskretyzacji są osobno znajdowane dla zbiorów trenujących i następnie aplikowane na zbiorach testowych

mcfs.classifier=j48 – definiuje typ klasyfikatora (możliwe wartości: j48, adx, sliq)

mcfs.discretizerConfigFile=discretizer.cfg – definiuje plik z parametrami dyskretyzacji

mcfs.saveResultInterval=10 – definiuje krok co ile kolejnych iteracji *s* ma być liczony dystans rankingów

mcfs.bestSetFractionSize=0.05 – definiuje 5% zbiór najwyższej ocenionych cech potrzebnych do obserwacji wartości rankingu (wartość większa od jednego definiuje dokładną liczbę cech)

mcfs.project=true – wartość *true* włącza losowanie cech

mcfs.projections=2000 – definiuje liczbę losowań cech (parametr

mcfs.projectionSize=300 – definiuje liczbę losowanych cech (parametr *m*), wartość mniejsza od 1 definiuje ułamek (0.1 to 10% cech)

mcfs.split=true – wartość *true* włącza losowy podział na zbiory trenujące i testujące

- mcfs.splits=10** – definiuje liczbę podziałów (parametr t)
- mcfs.splitRatio=0.66** – definiuje stosunek podziału na zbiór trenujący i testujący
- mcfs.balanceClasses=true** – wartość *true* włącza balansowanie liczby przykładów w klasach
- mcfs.balanceRatio=2** – definiuje krotność dla najmniej licznej klasy w zbiorze, pozostałe bardziej liczne klasy będą losowo próbkowane tak by ich liczba nie przekroczyła liczności klasy minimalnej mnożonej przez *balanceRatio*, działa dla *mcfs.balanceClasses=true*
- mcfs.permuteClass=false** – wartość *true* powoduje permutowanie klasy decyzyjnej, proces ten jest nadrzędny w stosunku do MCFS (wykonywany jednorazowo przed pętlą projekcji i selekcji)
- mcfs.permutations=1** – definiuje liczbę eksperymentów permutacji
- mcfs.measureName=RI** – definiuje preferowaną miarę oceny atrybutu np. do wyboru najlepszych 5% do określenia wartości dystansu (możliwe wartości: 'freq' 'crudeRI' 'RI_wAcc' 'RI_wAcc2' 'RI')
- mcfs.u=1** – definiuje parametr u w mierze RI
- mcfs.v=1** – definiuje parametr v w mierze RI
- mcfs.filterAttributes=false** – wartość *true* pozwala na aplikowanie zewnętrznego filtra atrybutów opartego na wybranych cechach
- mcfs.histogramFilterFileName=histogram.csv** – definiuje nazwę pliku rankingów atrybutów do filtrowania
- mcfs.filterSize=20** – definiuje liczbę najwyżej ocenionych cech w pliku ranking *mcfs.histogramFilterFileName*, które mają być wybrane do bieżącego eksperymentu
- mcfs.inverseFiltering=true** – wartość *true* powoduje wybór *mcfs.filterSize* najwyżej ocenionych cech z *mcfs.histogramFilterFileName* jako filtr w bieżącym eksperymencie, wartość *false* powoduje wybranie pozostałych
- mcfs.mode=1** – wartość 1 definiuje tryb działania MCFS dla wszystkich klas, wartość 2 jedna klasa przeciwstawiona pozostałym połączonym, wartość 3 uruchamia przetwarzanie dla pary klas (uruchamia trzykrotnie MCFS dla problemu o trzech klasach)
- mcfs.removeBestAttributes=false** – wartość *true* uruchamia iteracyjne działanie MCFS, kolejne uruchomienia odbywają się na zbiorach w których usunięto najwyżej ocenione atrybuty z poprzedniej iteracji
- mcfs.bestSetRemoveSize=5** – definiuje liczbę kolejnych cech usuwanych w trybie iteracyjnym
- mcfs.runs=10** – definiuje liczbę iteracji w trybie iteracyjnym
- j48.useInfoGain=true** – wartość *false* powoduje użycie w RI wartości miary *gainRatio* dla węzła w klasyfikatorze J4.8
- adx.useComplexQuality=true** – wartość *false* powoduje użycie tylko jakości poszczególnych selektorów w RI dla klasyfikatora ADX
- adx.qMethod=2** – definiuje miarę jakości kompleksu używanej w RI dla klasyfikatora ADX
- sliq.useDiversityMeasure=true** – wartość *false* powoduje używanie w RI miary *goodnessOfSplit* dla węzła w klasyfikatorze Sliq

9. Podsumowanie

Seria eksperymentów (rozdział 7) badających działanie algorytmów ADX i MCFS wykazała działanie zgodne z założeniami potwierdzając ich interesujące i przydatne właściwości.

Algorytm ADX w porównaniu z najbardziej popularnymi technikami klasyfikacji (tabela 7.5) wykazuje wysoką efektywność działania przy skuteczności klasyfikacji na poziomie najlepszych metod. Dla badanych rzeczywistych dużych zbiorów danych (gdzie liczba przykładów przekracza 15 tysięcy) proces walidacji krzyżowej dla ADX wynosi zaledwie kilka sekund, uzyskując najszybszy czas wobec innych technik, z których najwolniejsza implementacja potrzebowała kilku godzin (rys. 7.25). Skalowalność algorytmu można w zasadzie przybliżyć wykresem liniowym (rys. 7.18, 7.19), choć wobec tak heurystycznego podejścia zdarza się (w zależności od charakteru danych), że wraz ze zwiększaniem liczby atrybutów lub przykładów czas obliczeń rośnie nawet wolniej niż liniowo. Taka skalowalność algorytmu uczenia jest niewątpliwie dużym sukcesem w szczególności w zastosowaniach praktycznych, gdzie wymaga się przetwarzania bardzo dużych zbiorów danych.

Finalna baza wiedzy tworzona przez ADX jest czytelna, a reguły znajdujące się w niej są wybierane pod kątem skutecznej klasyfikacji przykładów, co również jest korzystne w rozwiązaniach biznesowych. Wybrany na drodze eksperymentalnej zestaw domyślnych parametrów ADX pozwala na uzyskiwanie wyników klasyfikacji bardzo zbliżonych do wyników przy optymalnym ustawieniu parametrów. Dla zbiorów danych, które nie brały udziału w procesie doboru parametrów nie uzyskano wyniku odstającego na niekorzyść w stosunku do innych technik. Pozwala to przypuszczać, że domyślne ustawienia ADX ograniczają wrażliwość klasyfikatora na charakter danych, a tym samym skracają proces ich ewentualnej optymalizacji.

Poprzez zastosowanie w ADX podejścia heurystycznego (w zasadzie na każdym etapie budowy modelu) uzyskano bardzo skuteczny i zarazem szybki klasyfikator regułowy, który niewątpliwie ma szansę być chętnie stosowany w rozwiązaniach praktycznych.

Jak wykazała druga część eksperymentów algorytm MCFS służący do budowy rankingu cech dla wysokowymiarowych danych jest również skutecznym narzędziem. Każdy gen/cecha zostaje w wyniku działania oceniony wartością miary *RI* relatywnej istotności. Metoda pozwala na obiektywną ocenę cech, przy czym kieruje się ich możliwościami dyskryminacyjnymi na zbiorze uczącym, jak również ich możliwościami przeprowadzenia poprawnej klasyfikacji nieznanych przykładów. Heurystyczna metoda budowy rankingu cech pozwala w krótkim czasie (rzędu kilkunastu do kilkudziesięciu minut dla zbiorów o tysiącach atrybutów) na uzyskanie stabilnego rankingu najistotniejszych cech i mimo badania w pracy jedynie zbiorów genetycznych doskonale nadaje

się do oceny dowolnych wysokowymiarowych danych. Autor z powodzeniem wykorzystywał MCFS również w zupełnie innych problemach, np. budowie rankingu cech odróżniających sklepy efektywne od nie efektywnych dla jednej z dużych sieci handlowych, gdzie każdy sklep był opisany setkami atrybutów odpowiadającym cechom demograficznym położenia sklepu.

W przypadku większości wysokowymiarowych zbiorów danych jedynie pewna niewielka grupa najwyższej ocenionych cech jest faktycznie istotna i dlatego krytycznym zdaniem była propozycja wskazania punktu odcięcia na liście rankingowej. Zadanie to zostało zrealizowane poprzez wykorzystanie testu statystycznego porównującego rozkład wartości najwyższej ocenionej oceny dla kolejnych zbiorów o permutowanej decyzji (sekcja 7.2.5) z wartościami z oryginalnego rankingu. Dzięki temu zabiegowi udało się w sposób bardzo staranny dla każdego badanego zbioru danych wskazać istotny zbiór genów. Tak wyselekcjonowany zbiór genów użyty do budowy klasyfikatorów pozwolił na znaczne poprawienie jakości klasyfikacji zbioru testowego w stosunku do losowej grupy genów (sekcja 7.2.7).

W sekcji 7.2.6 zauważono iż rankingi otrzymane przy pomocy różnych technik klasyfikacji są podobne do siebie. Biorąc pod uwagę ogromną przestrzeń możliwych do wyboru cech i analizując pozycje szczytowe rankingów (dla różnych bazowych klasyfikatorów, np. C4.5, ADX i Sliq) należy stwierdzić, że wspólna część około połowy cech dla każdej z list to duży sukces. Istnienie zbioru genów znajdujących się w każdym rankingu pozwala stwierdzić że metoda MCFS jest w dużej mierze niezależna od klasyfikatora. Przy tym dla algorytmu ADX klasyfikacja przeprowadzona na podstawie tylko wyselekcjonowanych cech pozwoliła na uzyskanie właściwie 100% klasyfikacji próbki danych, wyłączonej wcześniej z procesu selekcji cech (rys. 7.57, 7.58).

Drugi sposób weryfikacji wyników MCFS, bazujący na wiedzy eksperckiej genetyków dotyczącej problemu białaczki, pozwolił na równie ciekawe odkrycia. Algorytm pozwolił na wyodrębnienie genów związanych z genezą choroby i odpowiedzią różnych typów komórek układu odpornościowego atakowanych odrębnie przez różne badane odmiany białaczki. Co bardzo istotne, przy pomocy MCFS wyselekcjonowano geny uaktywniające się we wczesnej fazie rozwoju choroby, w odróżnieniu od innych metod, które raczej wskazują na geny związane już ze skutkami przebiegu schorzenia. Spostrzeżenie daje duże nadzieje na skuteczny i obiektywny wybór istotnych genów/cech które mogą przyczyniać się do rozwoju raka, bądź pozwalają na bardzo wczesną diagnozę choroby. Artykuł [Dramiński *et al.*, 2008] opisujący metodę MCFS i wnioski analizy biologicznej, został zatwierdzony do publikacji w prestiżowym czasopiśmie Bioinformatics, co wskazuje na zainteresowanie światowego środowiska bioinformatyki uzyskanymi rezultatami.

Bibliografia

- [Agrawal *et al.*, 1994] Agrawal R., Srikant R. (1994) Fast Algorithms for Mining Association Rules. In Proc. of the 20th Int'l Conference on Very Large Databases, Santiago, Chile.
- [Alizadeh *et al.*, 2000] Alizadeh A.A., Eisen M.B, Davis R.E., Ma C., Lossos I.S., Rosenwald A., Boldrick J.C., Sabet H., Tran T., Yu X., Powell J.I., Yang L., Marti G.E., Moore T., Hudso J. Jr., Lu L., Lewis D.B., Tibshirani R., Sherlock G., Chan W.C., Greiner T.C., Weisenburger D.D., Armitage J.O., Warnke R., Levy R., Wilson W., Grever M.R., Byrd J.C., Botstein D., Brown P.O., Staudt L.M. (2000) Distinct Types of Diffuse Large B-cell Lymphoma Identified by Expression Profiling. *Nature* **403**, 503-511.
- [Alsberg] B.K Alsberg, Norwegian University of Science and Technology, Division of Physical Chemistry, <http://phys.chem.ntnu.no/bka/research.html>
- [ASIMO] Humanoid Robot ASIMO by HONDA, <http://www.honda-robots.com/english/html/asimo/frameset2.html>
- [Bazan *et al.*, 2002] Bazan J.G. , Szczuka M.S. , Jakub Wróblewski J. (2002) A new version of rough set exploration system. In: James J. Alpigini, James F. Peters, Andrzej Skowron, Ning Zhong, Third International Conference on Rough Sets and Current Trends in Computing RSCTC, volume 2475, Lecture Notes in Artificial Intelligence, pp. 397-404, Malvern, PA, October 14-16 2002. Springer-Verlag.
- [BDGS] Biology Departament Genomic Service, <http://transcriptome.ens.fr/sgdb/presentation/principle.php>
- [BioInfo Bank] BioInfo Bank, <http://gibk26.bse.kyutech.ac.jp/jouhou/jouhoubank.html>
- [Breiman *et al.*, 1984] Breiman L., Friedman J.H., Olshen R.A., Stone C.J (1984) *Classification and Regression Trees*. Wadsworth International Group, Monterey, Ca.
- [Breiman, 1996] Breiman L. (1996) Bagging predictors. *Machine Learning*, 24(2):123-140.
- [Breiman, 2001] Breiman L. (2001) Random Forests. *Machine Learning* 45 (1), pp. 5-32.
- [Bryan, 1951] Bryan, J. G. (1951). The generalized discriminant function: mathematical foundation and computational routine. *Harvard Educ. Rev.*, 21: 90-95.
- [Cendrowska, 1987] Cendrowska J. (1987) PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machines Studies*, 27:349-370.
- [Chow *et al.*, 1968] Chow C., Liu C. (1968) Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3), pp. 462-467.
- [Cichosz, 2000] Cichosz P. (2000) Systemy uczące się. *Wydawnictwa Naukowo Techniczne*, Warszawa.
- [Clarac *et al.*, 1997] Clarac P., Niblett T. (1989) The CN2 induction algorithm. *Machine Learning*, 3, p. 261-283

- [Cohen, 1960] Cohen J. (1960) A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 1960:37-46.
- [Cohen, 1995] Cohen W. (1995) Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Tahoe, California.
- [Cover *et al.*, 1967] Cover T.M. Hart, P.E. Nearest neighbor pattern classification. *IEEE Trans. Inform. Theory*, IT-13(1):21–27, 1967.
- [Dramiński *et al.*, 2004] Dramiński, M., Koronacki, J., Ćwik, J. and Komorowski, J. (2004) Monte Carlo Gene Screening for Supervised Classification. In: *Current Issues in Data and Knowledge Engineering*, B. De Baets, R. de Caluwe, G. de Tr, J. Fodor, J. Kacprzyk, S. Zadrozny (eds.), Exit, Warsaw.
- [Dramiński *et al.*, 2008] Dramiński M., Rada-Iglesias A., Enroth S., Wadelius C., Koronacki J., Komorowski J. (2008) Monte Carlo feature selection for supervised classification. *Bioinformatics*, vol. 24(1), pp. 110–117.
- [Dudoit *et al.*, 2002] Dudoit S., Shaffer J.P., Boldrick J.C. (2002) Multiple Hypothesis Testing in Microarray Experiments. Technical Report 110, Division of Biostatistics, Univ. of California, Berkeley.
- [Dudoit *et al.*, 2003] Dudoit S., Fridlyand J. (2003) Classification in Microarray Experiments. In: *Statistical Analysis of Gene Expression Microarray Data*, T. Speed (ed.), Chapman & Hall/CRC, 93-158.
- [Dudoit *et al.*, 2003] Dudoit S., Shaffer J.P., Boldrick, J.C. (2003) Multiple Hypothesis Testing in Microarray Experiments. *Statistical Science*, **18**, 71-103.
- [EBI] Brazma A., Parkinson H., Schlitt T., Shojatalab M. (2001) A quick introduction to elements of biology - cells, molecules, genes, functional genomics, microarrays. European Bioinformatics Institute, http://www.ebi.ac.uk/microarray/biology_intro.html
- [Efron, 1979] Efron B. (1979) Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics* 7 (1): 1–26.
- [Fayyad *et al.*, 1992] Fayyad U.M., Irani K.B. (1992) On the Handling of Continuous-Valued Attributes in Decision Tree Generation. *Machine Learning*, vol. 8, pp. 87–102
- [Fayyad *et al.*, 1996] Fayyad U., Piatetsky-Shapiro G., Smyth P. (1996) From Data Mining to Knowledge Discovery: An Overview, in Fayyad, Piatetsky-Shapiro, Smyth, Uthurusamy, *Advances in Knowledge Discovery and Data Mining*, AAAI Press / The MIT Press, Menlo Park, CA, pp.1-34
- [Fisher, 1936] Fisher R.A. (1936), "The use of multiple measurements in taxonomic problems", *Annals of Eugenics* 7, 179-188.
- [Fix *et al.*, 1951] Fix E., Hodges J. L. (1951) Discriminatory analysis – nonparametric discrimination: consistency properties, Project 21-49-004, Report no. 4, USAF School of Aviation Medicine, pp. 261–279, Randolph Field.
- [Frawley, 1992] W. Frawley, G. Piatetsky-Shapiro, C. Matheus (1992), Knowledge Discovery in Databases: An Overview. *AI Magazine*, pp 213–228.
- [Freund *et al.*, 1996] Y. Freund, R. Schapire (1996) Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference*, strongy 148–156.
- [Freund *et al.*, 1997] Freund Y., Schapire R. (1997) A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.
- [Gaines, 1989] Gaines B.R. (1989) An ounce of knowledge is worth a ton of data: quantitative studies of the trade-off between expertise and data based

- on statistically well-founded empirical induction. Proceedings of the Sixth International Workshop on Machine Learning. pp.156-159
- [Gene Ontology] The Gene Ontology Consortium. "Gene Ontology: tool for the unification of biology." *Nature Genet.* (2000) 25: 25-29.
- [Golub *et al.*, 1999] Golub T.R., Slonim D.K., Tamayo P., Huard C., Gaasenbeek M., Mesirov J.P., Coller H., Loh M., Downing J.R., Caligiuri M.A., Bloomfield C.D., Lander E.S. (1999) Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. *Science* **286**, 531-537.
- [Grzymala-Busse *et al.*, 1997] Grzymala-Busse J.W., Wang A.Y. (1997) Modified algorithms LEM1 and LEM2 for rule induction from data with missing attribute values, w: Proc. of 5th Int. Workshop on Rough Sets and Soft Computing (RSSC'97) at JCIS'97, pp. 69-72.
- [Grzymala-Busse, 2003] Grzymala-Busse J. W. (2003) MLEM2-Discretization During Rule Induction, Intelligent Information Processing and Web Mining, Proceedings of the International IIS:IIPWM'03 Conference held in Zakopane Poland.
- [Grzymala-Busse, 1992] Grzymala-Busse J. W. (1992) LERS - a system for learning from examples based on rough sets. R. Słowiński (red.), Intelligent Decision Support. Handbook of Applications and Advances of the Rough Sets Theory, pp. 3-18. Kluwer Academic Publishers, Norwell (MA), 1992.
- [Grzymala-Busse, 1999] Grzymala-Busse J. W. (1999) LERS - a knowledge discovery system. R. Słowiński (red.), Rough Sets in Knowledge Discovery 2. Applications, Case Studies and Software Systems, pp. 562-565. Physica-Verlag, Heidelberg.
- [Hand, 2001] D.J. Hand , H. Mannila , P. Smyth (2001) Principles of Data Mining, MIT Press Computers / Data Base Management, 546 pages, ISBN 026208290X.
- [Human Genome Project] Human Genome Project,
http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml
- [Jóźwik *et al.*, 1998] Jóźwik A., Serpico S., Roli F.: A parallel network of modified 1- NN and k-NN classifiers – application to remote-sensing image classification, *Pattern Recognition Letters* 19, pp. 57-62, 1998.
- [Kaufman *et al.*, 1999] Kaufman K.A., Michalski R.S.: Learning From Inconsistent and Noisy Data: The AQ18 Approach. Proceedings of the Eleventh International Symposium on Methodologies for Intelligent Systems (ISMIS'99) Warsaw, pp. 411-419
- [Kerber, 1992] Kerber R. (1992) Chimerge: Discretization of numeric attributes. In: *Proceedings of the Tenth National Conference on Artificial Intelligence*, MIT Press, pp. 123-128.
- [Kłopotek, 2002] Kłopotek M.A. (2002) A New Bayesian Tree Learning Method with Reduced Time and Space Complexity. *Fundamenta Informaticae*, 49(no 4), IOS Press, pp. 349-367.
- [Koronacki *et al.*, 2005] Koronacki J., Ćwik J. (2005) Statystyczne systemy uczące się, Wydawnictwa Naukowo-Techniczne, ISBN: 83-204-3157-3.
- [Loh *et al.*, 1988] Loh W.-Y., Vanichsetakul N. (1988) Tree-structured classification via generalized discriminant analysis (with discussion), *Journal of the American Statistical Association*, vol. 83, 715-728.
- [Loh *et al.*, 1997] Loh W.-Y., Shih Y.-S. (1997) Split selection methods for classification trees, *Statistica Sinica*, vol. 7, 815-840.
- [Mehta *et al.*, 1996] Mehta M., Agrawal R., Rissanen J. (1996) SLIQ: A fast sca-

- lable classifier for data mining. In Proceedings of the Fifth International Conference on Extending Database Technology. 18–32.
- [Michalski *et al.*, 2001] Michalski R.S., Kaufman K.A. (2001) The AQ19 System for Machine Learning and Pattern Discovery: A General Description and User's Guide, Reports of the Machine Learning and Inference Laboratory, MLI 01–2, George Mason University, Fairfax, VA
- [Minsky, 2001] Minsky M. (2001) Thoughts About Artificial Intelligence, the chapter in *The Age of Intelligent Machines*, Ray Kurzweil.
- [Mitchell, 1997] Mitchell T. (1997) Machine Learning, McGraw Hill, ISBN 0070428077.
- [Mitchell, 1999] Mitchell T. (1999) Machine Learning and Data Mining *Communications of the ACM*, Vol. 42, No. 11.
- [Ohrn, 1997] Ohrn A., Komorowski J. (1997) ROSETTA: A Rough Set Toolkit for Analysis of Data. Joint Conference of Information Sciences: semiotics, fuzzy logic, soft computing, computer vision, neural computing, genetic algorithm, pattern recognition, evolutionary computing, Durham, NC, Duke University Press: 403-407.
- [Pawlak, 1982] Pawlak Z. (1982) Rough sets. International Journal of Computer and Information Sciences, 11(5):341–356.
- [Pawlak, 1991] Pawlak Z. (1991). Rough Sets: Theoretical Aspects of Reasoning About Data. Dordrecht: Kluwer Academic Publishing. ISBN 0-7923-1472-7.
- [Quinlan, 1986] Quinlan J. R. (1986) Induction of decision trees. Machine Learning, 1, p. 81–106
- [Quinlan, 1993] Quinlan J. R. (1993) C4.5: Programs for Machine Learning. Morgan Kaufmann
- [Quinlan, 1996] Quinlan J. R. (1996) Improved use of continuous attributes in C4.5. Journal of Artificial Intelligence Research, 4, 77–90
- [RoboCup] RoboCup, <http://www.robocup.org/>
- [R-project] The R project for statistical computing, <http://www.r-project.org/>
- [RSES] Rough Set Exploration System, <http://alfa.mimuw.edu.pl/rses/>
- [SAS] SAS, <http://www.sas.com/>
- [Shafer *et al.*, 1996] Shafer, J. C.; Agrawal, R.; and Mehta, M. (1996) SPRINT: A scalable parallel classifier for data mining. In Proceedings of the 22nd International Conference on Very Large Databases. Morgan Kaufmann. 544–555.
- [Shakhnarovich *et al.*, 2005] Shakhnarovich G., Darrell T., Indyk P. (2005) Nearest-Neighbor Methods in Learning and Vision, The MIT Press, 2005, ISBN 0-262-19547-X
- [Simon *et al.*, 2003] Simon R., Radmacher M.D., Dobbin K., McShane L.M. (2003) Pitfalls in the Use of DNA Microarray Data for Diagnostic and Prognostic Classification. *J. National Cancer Inst.* **95**, 14-18.
- [Smyth *et al.*, 2003] Smyth G.K., Yang J.H., Speed T.P. (2003) Statistical Issues in cDNA Microarray Data Analysis. In: *Functional Genomics: Methods and Protocols*, M.J. Brownstein and A.B. Khodursky (eds.), Methods in Molecular Biology Volume 224, Humana Press, Totowa, NJ, 111-136.
- [Speed, 2003] Speed T. [ed.]: *Statistical Analysis of Gene Expression Microarray Data*. Chapman & Hall/CRC, 2003.
- [Statistica] StatSoft, <http://www.statsoft.com/>
- [Stefanowski, 1998] Stefanowski J. (1998) On rough set based approaches to induction of decision rules. In Polkowski L., Skowron A. (eds.) Rough Sets in Data Mining and Knowledge Discovery, Physica—Verlag, 500-529

-
- [Stefanowski, 2001] Stefanowski J. (2001) Algorytmy indukcji reguł decyzyjnych w odkrywaniu wiedzy, wolumen 261. Wydawnictwo Politechniki Poznańskiej.
- [Tibshirani *et al.*, 2003] Tibshirani R., Hastie T., Narasimhan B., Chu G. (2003) Class Prediction by Nearest Shrunken Centroids, with Applications to DNA Microarrays. *Statistical Science*, **18**, 104-117.
- [UCI Repository] Newman D.J., Hettich S., Blake C.L. , Merz C.J. (1998). UCI Repository of machine learning databases [<http://www.ics.uci.edu/mllearn/MLRepository.html>]. Irvine, CA: University of California, Department of Information and Computer Science.
- [Vapnik, 1995] Vapnik V. (1995) The Nature of Statistical Learning Theory. Springer-Verlag, New York.
- [Weka 3] Wittenn I. H., Eibe F. (2005), Weka 3: Data Mining Software in Java ,*Data Mining: Practical Machine Learning Tools and Techniques. 2nd Edition*, Morgan Kaufmann, San Francisco.