Uniwersytet Warszawski

Wydział Matematyki, Informatyki i Mechaniki

Arkadiusz Wojna

Nr albumu: 158975

Adaptacyjne definiowanie funkcji boolowskich z przykładów

Praca magisterska

na kierunku **MATEMATYKA**

Praca wykonana pod kierunkiem

prof. dr hab. Andrzej Skowron

Instytut Matematyki

Grudzień 2000

Pracę przedkładam do oceny

Data Podpis autora pracy

Praca jest gotowa do oceny przez recenzenta

Data Podpis kierującego pracą

Streszczenie

Tematem pracy są inkrementacyjne algorytmy definiowania funkcji boolowskich z przykładów. We wstepnej części opisane są metody przekształcania przestrzeni atrybutów wielowartościowych do przestrzeni wektorów wartości binarnych, a następnie przedstawiony jest podstawowy inkrementacyjny algorytm liczenia minimalnych reguł klasyfikacyjnych wraz z analizą jego złożoności. Następnie przedstawiona jest zmodyfikowana wersja tego algorytmu, która redukuje przeszukiwanie przestrzeni reguł do reguł spełniających dynamiczne ograniczenia monotoniczne. Dla klasy monotonicznych ograniczeń opartych na mierze pokrycia opisane i przedyskutowane zostały różne problemy wydajnościowe związane ze zmodyfikowaną wersją algorytmu. W końcowej części pracy zaprezentowane zostały zestawienia jakościowe i wydajnościowe opisanych algorytmów dla 5 przykładowych zbiorów danych.

Słowa kluczowe: uczenie maszynowe, uczenie inkrementacyjne, algorytmy decyzyjne, zbiory przybliżone, algorytmy dyskretyzacji

Klasyfikacja tematyczna: 03B05, 03D15, 68Q32, 68T05

Za poświęcony czas i cenne uwagi dziękuję
profesorowi Andrzejowi Skowronowi

Spis rzeczy

ROZDZIAŁ 1

WSTĘP

Temat pracy jest związany z teoretycznymi i praktycznymi zagadnieniami dotyczącymi definowania pojęć z dynamicznie rozszerzających się zbiorów danych.

Badania nad procesami uczenia się z danych sięgają początku lat 60-tych, kiedy wraz z rozwojem komputerów pojawiły się pierwsze próby zastosowania ich do automatycznego uczenia. Jeden z nurtów tych badań zmierzał do stworzenia matematycznego modelu uczenia: w 1967 roku Gold [**17**] zaproponował model indukcyjny, a w 1984 roku Valiant [**45**] podał bardziej praktyczny model statystyczny. Równolegle trwały prace zmierzające do opracowania metod uczenia, które miałyby dobre własności w terminach zdefinowanego modelu uczenia i jednocześnie charakteryzowały się małą złożonością czasową [**3**]. Ze względu na zazwyczaj dynamiczną naturę rzeczywistych zbiorów danych ogólny model uczenia już od samego początku uwzględniał dynamiczny charakter danych wejściowych.

Niniejsza praca dotyczy algorytmu liczenia reguł decyzyjnych będącego rozwinięciem inkrementacyjnej metody opisanej przez Shana i Ziarko [**40**], opartej na teorii zbiorów przybliżonych wprowadzonej przez Pawlaka [**35**]. Algorytm ten jest określony na dziedzinie funkcji boolowskich i bazuje na kluczowych w teorii zbiorów przybliżonych pojęciach reduktu i reguły generowanej z reduktu [**37**], [**38**]. Praca przedstawia teoretycze podstawy algorytmu i jego definicję, pokazuje, że spełnia on wymagane własności poprawności i ma średnią złożoność czasową podwykładniczą oraz opisuje metody stosowania wyliczonych reguł do klasyfikacji.

Struktura pracy jest nastepująca. Rozdział 2 przedstawia definicję przestrzeni funkcji boolowskich będącej dziedziną algorytmu oraz najważniejsze przykłady takich przestrzeni wraz z ich własnościami. Rozdział 3 opisuje metodę przekształcania wielowymiarowej dziedziny atrybutów wielowartościowych do wielowymiarowej dziedziny boolowskiej

umożliwiającą uniwersalne zastosowanie algorytmu. Rozdział 4 wprowadza teoretyczne podstawy algorytmu i wynikające stąd ograniczenia oraz przedstawia definicję i własności algorytmu inkrementacyjnego opisanego przez Shana i Ziarko [**40**], natomiast rozdział 5 zawiera analizę średniej złożoności czasowej tego algorytmu. Rozdział 6 przedstawia definicję oraz opis inkrementacyjnego algorytmu opartego na monotonicznych ograniczeniach będącego modyfikacją algorytmu Shana i Ziarko, jak również opis różnych technik poprawienia wydajności tego algorytmu. W rozdziale 7 opisane są metody klasyfikacji bazujące na policzonym zbiorze reguł decyzyjnych oraz rezultaty uzyskane przy zastosowaniu algorytmu do uczenia się pojęć z rzeczywistych zbiorów danych.

Praca była realizowana w ramach grantu KBN 8T11C 025 19.

ROZDZIAŁ 2

PROBLEM DEFINIOWANIA FUNKCJI BOOLOWSKICH Z PRZYKŁADÓW

Dziedziną poszukiwań przedstawionej w pracy metody definiowania pojęć jest ustalona przestrzeń funkcji boolowskich. W tym rozdziale przedstawimy definicje przestrzeni funkcji boolowskich i problemu definiowania takich funkcji oraz podamy ważne przykłady takich przestrzeni [**3**].

DEFINICJA 2.1. Przestrzeń n-argumentowych funkcji boolowskich

Dla ustalonego $n \in \mathbb{N}$ niech $BA_n = \{a_1, a_2, \ldots, a_n\}$ będzie uporządkowanym liniowo zbiorem zmiennych boolowskich takim, że $\|BA_n\| = n$. Niech $U_n = \{0,1\}^{BA_n}$ będzie zbiorem wszystkich n-bitowych wektorów, w których każda współrzędna odpowiada ustalonej zmiennej z BA_n i niech $V_d = \{d_1, \ldots, d_k\}$ będzie dowolnym skończonym zbiorem. Zbiór BA_n nazywamy *zbiorem atrybutów*, zbiór U_n *zbiorem obiektów*, a zbiór V_d *zbiorem decyzji*. Zbiór $F_n = V_d^{U_n}$ to zbiór wszystkich funkcji boolowskich nad zbiorem obiektów U_n. *Przestrzeń n-argumentowych funkcji boolowskich* to dowolny podzbiór $H_n \subseteq F_n$.

Dla ustalonego m naturalnego *m-próbką treningową* w zbiorze obiektów U_n nazwiemy dowolny ciąg $s \in (U_n \times V_d)^m$. Pojęcie próbki w uczeniu inkrementacyjnym odpowiada pojęciu tabeli decyzyjnej w teorii zbiorów przybliżonych [**37**].

DEFINICJA 2.2. Problem definiowania funkcji boolowskiej w ustalonej przestrzeni $H_n \subseteq F_n$

Problem definiowania funkcji boolowskiej w ustalonej przestrzeni H_n polega na skonstruowaniu efektywnego algorytmicznie odwzorowania $L : \bigcup_{m\in\mathbb{N}} (U_n \times V_d)^m \to H_n$.

Poniżej zdefinujemy podstawową własność wymaganą przy dobrych rozwiązaniach tego problemu.

DEFINICJA 2.3. Własność zgodności odwzorowania z ustaloną przestrzenią funkcji boolowskich $H_n \subseteq F_n$

Mówimy, że m-próbka treningowa $s \in (U_n \times V_d)^m$ jest *zgodna* z funkcją boolowską $f \in F_n$, jesli dla każdego przykładu $(x, b) \in U_n \times V_d$ występującego w tej próbce $f(x) = b$. Odwzorowanie $L : \bigcup_{m\in\mathbb{N}} (U_n \times V_d)^m \to H_n$ jest *zgodne* z przestrzenią H_n, jeśli dla każdej próbki $s \in (U_n \times V_d)^m$ zgodnej z pewną funkcją w H_n i dla każdego przykładu $(x, b) \in U_n \times V_d$ występującego w tej próbce $L(s)(x) = b$.

Do celu badania złożoności metod definowania funkcji boolowskich w zależności od liczby zmiennych wykorzystuje się pojęcie indeksowanej przestrzeni funkcji boolowskich.

DEFINICJA 2.4. Indeksowana przestrzeń funkcji boolowskich

Indeksowana przestrzeń funkcji boolowskich to zbiór $H = \bigcup_{n\geq 1} H_n$, gdzie każde H_n to dowolna przestrzeń n-argumentowych funkcji boolowskich.

Poniżej przedstawiamy najważniejsze przykłady przestrzeni funkcji boolowskich. We wszystkich przykładach zakładamy, że zbiór decyzji jest dwuwartościowy $V_d = \{0, 1\}$.

DEFINICJA 2.5. Przestrzeń jednomianów

Jednomian n-argumentowy wyznaczony przez podzbiór zmiennych boolowskich $B \subseteq BA_n$ to n-argumentowa funkcja boolowska f_B taka, że:

$$f_B(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{jeśli dla każdego } a_i \in B \text{ zachodzi } x_i = 1 \\ 0 & \text{wpp.} \end{cases}$$

Przez M_n oznaczamy przestrzeń wszystkich n-argumentowych jednomianów:

$$M_n = \{f_B \in F_n : B \subseteq BA_n\}$$

a przez $M_{n,k}$ przestrzeń wszystkich n-argumentowych jednomianów długości co najwyżej k:

$$M_{n,k} = \{f_B \in F_n : B \subseteq BA_n \wedge \|B\| \leq k\}$$

DEFINICJA 2.6. Przestrzeń alternatyw małych jednomianów

Alternatywa n-argumentowa k-małych jednomianów wyznaczona przez podzbiór jednomianów $P \subseteq M_{n,k}$ to n-argumentowa funkcja boolowska f_P taka, że:

$$f_P(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{jeśli istnieje } f \in P \text{ takie, że } f(x_1, x_2, \ldots, x_n) = 1 \\ 0 & \text{wpp.} \end{cases}$$

Przez $D_{n,k}$ oznaczamy zbiór wszystkich n-argumentowych alternatyw k-małych jednomianów:

$$D_{n,k} = \{f_P \in F_n : P \subseteq M_{n,k}\}$$

Definicja 2.7. Przestrzeń klauzul

Klauzula n-argumentowa wyznaczona przez podzbiór zmiennych boolowskich $C \subseteq BA_n$ to n-argumentowa funkcja boolowska f_C taka, że:

$$f_C(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{jeśli istnieje } a_i \in C \text{ takie, że } x_i = 1 \\ 0 & \text{wpp.} \end{cases}$$

Przez C_n oznaczamy przestrzeń wszystkich n-argumentowych klauzul:

$$C_n = \{f_C \in F_n : C \subseteq BA_n\}$$

Definicja 2.8. Przestrzeń małych konjunkcji klauzul

Konjunkcja klauzul n-argumentowa wyznaczona przez podzbiór klauzul $R \subseteq C_n$ to n-argumentowa funkcja boolowska f_R taka, że:

$$f_R(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{jeśli dla każdego } f \in R \text{ zachodzi } f(x_1, x_2, \ldots, x_n) = 1 \\ 0 & \text{wpp.} \end{cases}$$

Przez $C_{n,k}$ oznaczamy przestrzeń wszystkich n-argumentowych k-małych konjunkcji klauzul:

$$C_{n,k} = \{f_R \in F_n : R \subseteq C_n \wedge \|R\| \leq k\}$$

Definicja 2.9. Przestrzeń list decyzyjnych nad ustalonym podzbiorem funkcji boolowskich $K_n \subseteq F_n$

Lista decyzyjna n-argumentowa nad K_n wyznaczona przez ciąg $p = ((f_1, c_1), \ldots, (f_r, c_r))$, w którym $f_i \in K_n$ oraz $c_i \in \{0, 1\}$, to n-argumentowa funkcja boolowska f_p taka, że:

$$f_p(x_1, x_2, \ldots, x_n) = \begin{cases} c_j & \text{jeśli } j = \min\{i : f_i(x_1, x_2, \ldots, x_n) = 1\} \text{ istnieje} \\ 0 & \text{wpp.} \end{cases}$$

Przez $DL(K_n)$ oznaczamy przestrzeń wszystkich n-argumentowych list decyzyjnych nad K_n:

$$DL(K_n) = \left\{ f_p \in F_n : p \in \bigcup_{i \in \mathbb{N}} (K_n \times \{0, 1\})^i \right\}$$

DEFINICJA 2.10. Przestrzeń perceptronów

Perceptron n-argumentowy wyznaczony przez ciąg $q = (w_1, \ldots, w_n, \theta)$, w którym $w_i \in \mathbb{R}$ oraz $\theta \in \mathbb{R}$, to n-argumentowa funkcja boolowska f_q taka, że:

$$f_q(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{jeśli } \sum_{i=1}^{n} w_i x_i \geq \theta \\ 0 & \text{wpp} \end{cases}$$

Przez P_n oznaczamy przestrzeń wszystkich n-argumentowych perceptronów:

$$P_n = \left\{ f_q \in F_n : q \in \mathbb{R}^{n+1} \right\}$$

ROZDZIAŁ 3

BINARNA DYSKRETYZACJA PRZESTRZENI OBIEKTÓW Z WIELOWARTOŚCIOWYMI ATRYBUTAMI

W tym rozdziale przedstawimy zastosowaną w doświadczeniach metodę binarnej dyskretyzacji przestrzeni obiektów z wielowartościowymi atrybutami opartą na pojęciu miary rozróżnialności. W tym celu zdefiniujemy najpierw wspomnianą przestrzeń.

DEFINICJA 3.1. Przestrzeń obiektów z wielowartościowymi atrybutami

Niech $A = \{a_1, a_2, \ldots, a_n\}$ będzie liniowo uporządkowanym zbiorem zmiennych rozmiaru n i niech $V = \{V_a\}_{a \in A}$ będzie rodziną zbiorów indeksowaną zbiorem A. Przestrzenią obiektów nad zbiorem atrybutów A i zbiorem wartości V nazywamy produkt uogólniony $\prod_{a \in A} V_a$. Podobnie jak przy funkcjach boolowskich, dany jest również skończony zbiór decyzji $V_d = \{d_1, \ldots, d_k\}$.

Podstawowe pojęcie wykorzystywane w dyskretyzacji to wektor rozkładu decyzyjnego.

DEFINICJA 3.2. Wektor rozkładu decyzyjnego

Dla ustalonego zbioru przykładów $U \subseteq \left(\prod_{a \in A} V_a\right) \times V_d$ wektor rozkładu decyzyjnego tego zbioru to wektor wartości $distr(U)$, w którym i-ta współrzedna określa, ile w zbiorze U było obiektów z decyzją d_i:

$$distr(U) = (\|\{u \in U : u(d) = d_1\}\|, \ldots, \|\{u \in U : u(d) = d_k\}\|)$$

Liczbę obiektów w zbiorze U z decyzją d_i bedziemy oznaczać przez

$$d_i(U) = \|\{u \in U : u(d) = d_k\}\|$$

.

Wprowadzimy teraz definicje miar heurystycznych opisujących rozkłady decyzyjne.

DEFINICJA 3.3. Długość wektora rozkładu decyzyjnego

Długość wektora rozkładu decyzyjnego $(d_1(U), \ldots, d_k(U))$ zbioru przykładów $U \subseteq \left(\prod_{a \in A} V_a\right) \times V_d$ zdefiniowana jest nastepująco:

$$\|(d_1(U), \ldots, d_k(U))\| = \sqrt{\sum_{i=1}^{k} (d_i(U))^2}$$

DEFINICJA 3.4. Rozróżnialność rozkładów decyzyjnych dwóch zbiorów przykładów [**33**]

Rozróżnialność rozkładów decyzyjnych dwóch zbiorów przykładów $U', U'' \subseteq \left(\prod_{a \in A} V_a\right) \times V_d$ zdefiniowana jest następująco:

$$DISCERN\,(U', U'') = \sum_{i \neq j} d_i(U') d_j(U'')$$

Zakładamy, że zbiór przykładów $U \subseteq \left(\prod_{a \in A} V_a\right) \times V_d$ jest ustalony. Metoda binarnej dyskretyzacji składa się z dwóch etapów, każdy z nich opisany jest jednym parametrem. Etap pierwszy polega na skonstruowaniu dla każdego atrybutu $a \in A$ niezależnie odwzorowania $bin_a : V_a \to \{0,1\}^p$, gdzie p jest ustaloną liczbą atrybutów binarnych dla pojedynczego atrybutu wielowartościowego. Etap drugi polega na wyborze spośród wszystkich otrzymanych atrybutów binarnych podzbioru atrybutów ustalonej wielkości n maksymalnie dobrze aproksymującego zbiór wszystkich atrybutów.

Rozróżniamy dwa typy atrybutów wielowartościowych.

DEFINICJA 3.5. Atrybuty numeryczne i nominalne

Atrybut numeryczny to atrybut $a \in A$, którego zbiór wartości V_a jest liniowo uporządkowany. Atrybut nominalny to atrybut $a \in A$, którego zbiór wartości V_a nie jest liniowo uporządkowany.

Pierwszy etap dyskretyzacji binarnej różni się istotnie dla poszczególnych typów atrybutów.

Do zdefiniowania odwzorowania $bin_a : V_a \to \{0,1\}^p$ dla ustalonego atrybutu numerycznego $a \in A$ wykorzystamy pojęcie cięcia, które dla dowolnej wartości $c \in V_a$ oznacza podział ustalonego zbioru przykładów $U \subseteq \left(\prod_{a \in A} V_a\right) \times V_d$ na dwa podzbiory:

część lewą $L_a\left(c,U\right)=\{u\in U:u(a)<c\}$ i część prawą $R_a\left(c,U\right)=\{u\in U:u(a)\geq c\}$. Optymalne cięcie to cięcie o maksymalnej rozróżnialności [**33**]:

$$c_{opt}(a,U)=\max_{c\in V_a} DISCERN\left(L_a\left(c,U\right),R_a\left(c,U\right)\right)$$

Dla danego zbioru przykładów $U\subseteq\left(\prod_{a\in A}V_a\right)\times V_d$ odwzorowanie $bin_a:V_a\to\{0,1\}^p$ atrybutu numerycznego a w zbiór p atrybutów binarnych definiujemy rekurencyjnie w kolejności od wartości $bin_a(v)(p-1)$ do wartości $bin_a(v)(0)$. Wartość każdego atrybutu binarnego wyznaczana jest przez cięcie optymalne z tym, że dla $p-1$-ego atrybutu binarnego cięcie optymalne c_{opt} wyznaczane jest dla całego zbioru przykładów U:

$$bin_a(v)(p-1)=\begin{cases}0 & \text{jeśli } v<c_{opt}(a,U)\\ 1 & \text{jeśli } v\geq c_{opt}(a,U)\end{cases}$$

a dla każdego kolejnego atrybutu binarnego $0\leq i<p-1$ cięcie optymalne wyznaczane jest osobno dla każdego podzbioru przykładów $U_{a,v,i}\subseteq U$ o ustalonej kombinacji wartości wszystkich poprzednio wyznaczonych atrybutów binarnych $bin_a(v)(p-1),\ldots,bin_a(v)(i+1)$:

$$U_{a,v,i}=\{u\in U:\forall i<j\leq p-1:bin_a\left(u(a)\right)(j)=bin_a(v)(j)\}$$

$$bin_a(v)(i)=\begin{cases}0 & \text{jeśli } v<c_{opt}(a,U_{a,v,i})\\ 1 & \text{jeśli } v\geq c_{opt}(a,U_{a,v,i})\end{cases}$$

Opisane odwzorowanie ma tą własność, że ustalona kombinacja $bin_a(v)(p-1),\ldots,bin_a(v)(0)$ wartości wszystkich p atrybutów binarnych obejmuje zawsze jeden przedział zbioru wartości V_a.

W przypadku atrybutu nominalnego $a\in A$ postępujemy analogicznie z tym, że teraz nie możemy posłużyć się pojęciem cięcia, ponieważ zbiór wartości V_a nie jest liniowo uporządkowany. Zamiast tego wprowadzimy następującą defincję optymalnego podziału $part_{opt}(a,W):W\to\{0,1\}$ dowolnego podzbioru $W\subseteq V_a$ przy ustalonym zbiorze przykładów $U\subseteq\left(\prod_{a\in A}V_a\right)\times V_d$. Najpierw wartości $v_0\in W$, która ma największą długość wektora decyzyjnego:

$$v_0=\max_{v\in W}\|distr(\{u\in U:u(a)=v\})\|$$

przypisujemy wartość 0:

$$part_{opt}(a, W)(v_0) = 0$$

W każdym kolejnym kroku $0 < l \leq \|W\|$ szukamy wartości $v_l \in W$ o maksymalnej rozróżnialności $DISCERN$ względem jednej ze stron $b \in \{0, 1\}$ częściowo ustalonego podziału $part_{opt}(a, W) : \{v_0, \ldots, v_{l-1}\} \to \{0, 1\}$:

$$U_{a,v} = \{u \in U : u(a) = v\}$$

$$U_{a,v_0,\ldots,v_{l-1},b} = \{u \in U : u(a) \in \{v_0, \ldots, v_{l-1}\} \wedge part_{opt}(a, W)\,(u(a)) = b\}$$

$$v_l = \max_{v \in W \setminus \{v_0,\ldots,v_{l-1}\}} \max_{b \in \{0,1\}} DISCERN(U_{a,v}, U_{a,v_0,\ldots,v_{l-1},b})$$

i dodajemy ją do strony przeciwnej:

$$part_{opt}(a, W)(v_l) = \min_{b \in \{0,1\}} DISCERN\left(U_{a,v_l}, U_{a,v_0,\ldots,v_{l-1},b}\right)$$

Teraz postępujemy już podobnie jak przy definicji odwzorowania $bin_a : V_a \to \{0, 1\}^p$ dla atrybutu numerycznego, tzn. dla danego zbioru przykładów $U \subseteq \left(\prod_{a \in A} V_a\right) \times V_d$ wartość każdego atrybutu binarnego $bin_a(v)(i)$ wyznaczana jest przez podział optymalny $part_{opt}(a, *)$ z tym, że dla $p - 1$-ego atrybutu podział optymalny wyznaczany jest dla całego zbioru wartości V_a:

$$bin_a(v)(p - 1) = part_{opt}(a, V_a)(v)$$

a dla kolejnych atrybutów binarnych $0 \leq i < p - 1$ podział optymalny wyznaczany jest osobno dla każdego podzbioru wartości $W_{a,v,i} \subseteq V_a$ odpowiadającego ustalonej kombinacji wartości wszystkich poprzednio wyznaczonych atrybutów binarnych $bin_a(v)(p - 1), \ldots, bin_a(v)(i + 1)$:

$$W_{a,v,i} = \{v' \in V_a : \forall i < j \leq p - 1 : bin_a(v')(j) = bin_a(v)(j)\}$$

$$bin_a(v)(i) = part_{opt}\,(a, W_{a,v,i})\,(v)$$

W ten sposób zdefiniowaliśmy etap pierwszy dyskretyzacji binarnej i możemy przejść do opisu etapu drugiego, w którym wybieramy n najlepszych spośród wszystkich

atrybutów binarych. Podamy tu kilka metod, przy czym wszystkie bazują na pojęciu rozróżnialności względnej oraz macierzy atrybutów binarnych, których definicje podajemy poniżej.

DEFINICJA 3.6. Rozróżnialność rozkładów decyzyjnych dwóch zbiorów przykładów względem atrybutów binarnych

Rozróżnialność $DISCERN_{ba_1,\ldots,ba_{j-1}}(U', U'')$ rozkładów decyzyjnych dwóch zbiorów przykładów $U', U'' \subseteq \left(\prod_{a\in A} V_a\right) \times V_d$ względem ustalonych atrybutów binarnych $ba_1, \ldots, ba_{j-1} : U' \cup U'' \to \{0,1\}$ to suma rozróżnialności par podzbiorów:

$$U'_{(ba_1,b_1),\ldots,(ba_{j-1},b_{j-1})} = \{u \in U' : ba_1(u) = b_1 \wedge \ldots \wedge ba_{j-1}(u) = b_{j-1}\}$$
$$U''_{(ba_1,b_1),\ldots,(ba_{j-1},b_{j-1})} = \{u \in U'' : ba_1(u) = b_1 \wedge \ldots \wedge ba_{j-1}(u) = b_{j-1}\}$$

po wszystkich binarnych kombinacjach $b_1, \ldots, b_{j-1} \in \{0,1\}$ wartości tych atrybutów:

$$\sum_{b_1,\ldots,b_{j-1}\in\{0,1\}} DISCERN\left(U'_{(ba_1,b_1),\ldots,(ba_{j-1},b_{j-1})}, U''_{(ba_1,b_1),\ldots,(ba_{j-1},b_{j-1})}\right)$$

Pojęcie rozróżnialności względnej jest wykorzystywane przy definicji macierzy atrybutów binarnych.

DEFINICJA 3.7. Macierz atrybutów binarnych

Zakładamy, że dla każdego atrybutu wielowartościowego $a \in A$ mamy dane odwzorowanie w atrybuty binarne $bin_a : V_a \to \{0,1\}^p$, gdzie p jest parametrem pierwszego etapu dyskretyzacji. Macierz atrybutów binarnych to funkcja $BAM : \{0,\ldots,n-1\} \times \{0,\ldots,n-1\} \to (A \times \{0,\ldots,p-1\}) \cup \{\perp\}$, w której wartościami są pary (a,k): atrybut wielowartościowy $a \in A$ oraz numer atrybutu binarnego $k \in \{0,\ldots,p-1\}$, identyfikujące odpowiedni atrybut binarny $bin_a(*)(k) : V_a \to \{0,1\}$ z pierwszego etapu dyskretyzacji. Dopuszczamy nieokreśloną wartość macierzy oznaczaną symbolem $\perp$. Każdy wiersz $BAM(i,0),\ldots,BAM(i,n-1)$ macierzy atrybutów binarnych musi spełniać następujący warunek: dla każdego ustalonego atrybutu $a \in A$ podciąg numerów atrybutów binarnych $k \in \mathbb{N}$ występujących z atrybutem a jako druga składowa wartości elementów wiersza (a,k) tworzy ciąg kolejnych malejących liczb naturalnych rozpoczynający się od $p-1$. Aby uwzględnić ten warunek w definicji macierzy, dla

każdej pozycji macierzy $BAM(i,j)$ i każdego atrybutu $a \in A$ wprowadzimy funkcję właściwego kandydata $cnd_{a,i,j}$ na kolejny atrybut binarny dla a. Wartość $cnd_{a,i,j}$ jest zawsze największym numerem atrybutu binarnego dla a, który w i-tym wierszu macierzy BAM nie wystąpił na pozycjach poprzedzających pozycję (i,j):

$$cnd_{a,i,j} = \max\{k \in \{0,\ldots,p-1\} : \forall l < j : BAM(i,l) \neq (a,k)\}$$

Pierwszą kolumnę macierzy atrybutów binarnych definiujemy w ten sposób, że na kolejnych pozycjach w tej kolumnie dobieramy $p-1$-sze czyli najbardziej znaczące atrybuty binarne $(a, p-1)$ uporządkowane według wartości miary rozróżnialności $DISCERN$. Dokładniej, jeśli zbiór atrybutów występujących jako pierwsza składowa na pozycjach $BAM(0,0),\ldots,BAM(i-1,0)$ nie wyczerpuje całego zbioru A, to jako pierwszy atrybut binarny w i-tym wierszu wybierany jest $p-1$-szy atrybut binarny tego atrybutu wielowartościowego $a \in A$, który spośród wcześniej niewybranych ma najlepszą rozróżnialność:

$$BAM(i,0) = \left(\max_{a \in A \setminus \{BAM(0,0)\downarrow 1,\ldots,BAM(i-1,0)\downarrow 1\}} DISCERN \left(\begin{array}{c} U_{(bin_a(*)(p-1),0),} \\ U_{(bin_a(*)(p-1),1)} \end{array} \right), p-1 \right)$$

W przeciwnym przypadku wartość pierwszego elementu i-tego wiersza jest nieokreślona:

$$BAM(i,0) = \perp$$

Może się zdarzyć, że maksymalna rozróżnialność zostanie osiągnięta dla kilka atrybutów, wybieramy wtedy dowolny z nich, na przykład pierwszy według uporządkowania liniowego na zbiorze A.

Następne atrybuty binarne $BAM\,(i,j)$ w i-tym wierszu dobierane są w ten sposób, by miały najlepszą rozróżnialność względem atrybutów binarnych

$$ba_1 = bin_{BAM(i,0)\downarrow 1}(*)(BAM(i,0) \downarrow 2)$$

$$\vdots$$

$$ba_{j-1} = bin_{BAM(i,j-1)\downarrow 1}(*)(BAM(i,j-1) \downarrow 2)$$

z poprzednich pozycji danego wiersza:

$$a_{max}(i,j) = \max_{a \in A: cnd_{a,i,j} \geq 0} DISCERN_{ba_1,\ldots,ba_{j-1}} \left(U_{(bin_a(*)(cnd_{a,i,j}),0)}, U_{(bin_a(*)(cnd_{a,i,j}),1)}\right)$$

Jeśli $a_{max}(i,j)$ istnieje i wartość rozróżnialności względnej $DISCERN_{ba_1,\ldots,ba_{j-1}}$ dla atrybutu binarnego $bin_{a_{max}(i,j)}(*)(cnd_{a_{max}(i,j),i,j})$ jest większa od 0, to na pozycji (i,j) macierzy atrybutów binarnych BAM ustawiany jest atrybut wielowartościowy $a_{max}(i,j)$ i właściwy dla tej pozycji numer atrybutu binarnego $cnd_{a_{max}(i,j),i,j}$:

$$BAM(i,j) = \left(a_{max}(i,j), cnd_{a_{max}(i,j),i,j}\right)$$

W przeciwnym przypadku wartość elementu macierzy jest nieokreślona:

$$BAM(i,j) = \perp$$

Podobnie jak przy definiowaniu pierwszej kolumny macierzy BAM, maksymalna rozróżnialność może zostać osiągnięta dla kilka atrybutów, wybieramy wtedy dowolny z nich.

W oparciu o definicję macierzy atrybutów binarnych BAM podamy cztery metody doboru n końcowych atrybutów binarnych. W tym celu dla każdej metody zdefiniujemy odpowiedni ciąg wartości elementów macierzy BAM rozpoczynający się w lewym górnym wierzchołku $BAM(0,0)$:

1. metoda horyzontalna: $seq_{hor}(in+j) = BAM(i,j)$ dla $0 \leq i < n$, $0 \leq j < n$ (atrybuty binarne wybierane są w porządku horyzontalnym macierzy BAM);
2. metoda trójkatna: $seq_{tr}(\frac{(i+j)(i+j+1)}{2} + i) = BAM(i,j)$ dla $0 \leq i < n$, $0 \leq j < n$ (atrybuty binarne wybierane są w porządku trójkatnym macierzy BAM);
3. metoda kwadratowa: $seq_{sq}((\max(i,j))^2 + i) = BAM(i,j)$ dla $0 \leq i < j < n$ oraz $seq_{sq}(\max(i,j)\,(\max(i,j)+1) + j) = BAM(i,j)$ dla $0 \leq j \leq i < n$ (atrybuty binarne wybierane są w porządku kwadratowym macierzy BAM);
4. metoda wertykalna: $seq_{ver}(jn+i) = BAM(i,j)$ dla $0 \leq i < n$, $0 \leq j < n$ (atrybuty binarne wybierane są w porządku wertykalnym macierzy BAM);

Z odpowiedniego ciągu seq dla ustalonej metody wybieramy pierwsze n różnych elementów mających wartości określone i w ten sposób uzyskujemy ostateczny zbiór atrybutów binarnych. Tą operacją kończymy proces dyskretyzacji binarnej.

Przedstawiliśmy w tym rozdziale różne przekształcania aproksymacyjne z przestrzeni nad atrybutami wielowartościowymi w przestrzeń wektorów binarnych ustalonego rozmiaru. Opisane przekształcenia pozwalają na dużo szersze zastosowanie opisanych w tej pracy metody definiowania funkcji boolowskich, gdyż silnie rozszerzają klasę przestrzeni obiektów dających się interpretować jako dziedziny danych wejściowych dla opisanych w następnych rozdziałach algorytmów.

ROZDZIAŁ 4

PODSTAWOWY INKREMENTACYJNY ALGORYTM LICZENIA REGUŁ MINIMALNYCH

W tym rozdziale najpierw przedstawimy pojęcia wykorzystywane przez podstawowy inkrementacyjny algorytm definiowania funkcji boolowskich, a następnie opiszemy sam algorytm, opracowany przez Ziarko i Shana [**40**]. Ponieważ przedstawiliśmy już metody sprowadzania przestrzeni atrybutów wielowartościowych do dziedziny boolowskiej i algorytmy opisywane w dalszej części pracy bazują wyłącznie na atrybutach binarnych, wszystkie odtąd definiowane pojęcia opieramy na zbiorze atrybutów binarnych BA_n, zbiorze obiektów binarnych $U_n = \{0,1\}^{BA_n}$ oraz przestrzeni funkcji boolowskich $F_n = V_d^{U_n}$.

Najpierw zdefiniujemy pojęcie reduktu. Wprowadzenie tego pojęcia do teorii zbiorów przybliżonych [**37**], [**38**], [**42**] pozwalało na redukcję informacji o obiektach poprzez uwzględnienie faktu, że często już nieduży podzbiór atrybytów rozróżnia dany zbiór przykładów tak samo dobrze jak zbiór wszystkich atrybutów. Rozróżniamy dwa typy reduktów: globalne i lokalne.

DEFINICJA 4.1. Reduktem globalnym dla próbki $s \in (U_n, V_d)^m$ jest każdy podzbiór atrybutów $R \subseteq BA_n$ spełniający następujące dwa warunki:

- dla każdej pary przykładów (u_i, d), (u_j, d') o różnych decyzjach $d \neq d'$ występujących w próbce s istnieje atrybut $a \in R$ rozróżniający obiekty $u_i(a) \neq u_j(a)$;
- zbiór R jest zbiorem minimalnym w sensie inkluzji spełniającym poprzedni warunek.

DEFINICJA 4.2. Reduktem lokalnym dla przykładu (u, d) w skończonej próbce $s \in (U_n, V_d)^m$ jest każdy podzbiór atrybutów $R \subseteq BA_n$ spełniający następujące dwa warunki:

- dla każdego przykładu (u_i, d') z inną decyzją $d' \neq d$ występującego w próbce s istnieje atrybut $a \in R$ rozróżniający obiekty $u(a) \neq u_i(a)$;
- zbiór R jest zbiorem minimalnym w sensie inkluzji spełniającym poprzedni warunek.

Zbiory spełniające pierwszy warunek w definicji reduktu globalnego (lokalnego) nazywane są odpowiednio nadreduktami globalnymi (lokalnymi). Prosta zależność między tymi pojęciami jest taka, że każdy redukt globalny jest nadreduktem lokalnym dla każdego przykładu występującego w próbce.

Pojęciem wykorzystywanym bezpośrednio przez algorytmy opisane w tej pracy jest pojęcie reguły klasyfikacyjnej.

DEFINICJA 4.3. Reguła klasyfikacyjna to dowolna formuła $\alpha \Rightarrow d$, w której α jest jednomianem nad zbiorem atrybutów BA_n, a $d \in V_d$ wartością decyzji.

W rozwiązaniach algorytmicznych redukty reprezentowane są poprzez zbiór reguł klasyfikacyjnych generowanych z reduktów nazywanych dalej regułami minimalnymi.

DEFINICJA 4.4. Regułą minimalną dla próbki s jest każda reguła $\bigwedge_{a \in R} (a = u(a)) \Rightarrow d$, w której (u, d) jest przykładem występującym w s, a R reduktem lokalnym dla (u, d).

Reguła jednoznaczna dla próbki s jest definiowana analogicznie w oparciu o pojęcie nadreduktu lokalnego, tzn. jest to taka reguła klasyfikacyjna, która rozróżnia dany obiekt od wszystkich obiektów z inną decyzją, natomiast nie musi spełniać warunku minimalności.

Zdefiniowaliśmy reguły minimalne pochodzące od reduktów lokalnych, bo tylko z takich będziemy korzystać, ale podobnie możnaby definiować reguły pochodzące od reduktów globalnych.

W ogólności reguły klasyfikacyjne generowane przez algorytmy nie muszą być rozłączne, w związku z tym istnieje potrzeba porównywania ich jakości. Dwie podstawowe miary wykorzystywane w wielu zastosowaniach to trafność (ang. confidence) i pokrycie (ang. coverage) [**35**], [**39**], [**44**].

Aby je zdefniować, wprowadzimy dodatkowe defincje dwóch zbiorów określanych dla ustalonej próbki s. Pierwszy zbiór to nośnik jednomianu α nad zbiorem zmiennych BA_n:

$$[\alpha]_s = \{(u_i, d_i) \in s : u_i \text{ spełnia } \alpha\}$$

Drugi to klasa decyzyjna z próbki s:

$$Class_s(d) = \{(u_i, d_i) \in s : d_i = d\}$$

Teraz możemy juz podac definicje wspomnianych miar. Trafność reguły $\alpha \Rightarrow d$ dla próbki s definiowana jest jako:

$$confidence_s(\alpha \Rightarrow d) = \begin{cases} 0 & \text{jeśli } [\alpha]_s = \emptyset \\ \frac{||[\alpha]_s \cap Class_s(d)||}{||[\alpha]_s||} & \text{jeśli } [\alpha]_s \neq \emptyset \end{cases}$$

a pokrycie reguły $\alpha \Rightarrow d$ dla próbki s to:

$$coverage_s(\alpha \Rightarrow d) = \frac{||[\alpha]_s \cap Class_s(d)||}{||Class_s(d)||}$$

Następujący fakt podaje charakterystykę jednoznacznych reguł klasyfikacyjnych przy pomocy miary trafności:

FAKT 4.5. *Reguła klasyfikacyjna $\alpha \Rightarrow d$ jest jednoznaczna wtedy i tylko wtedy gdy $confidence_s(\alpha \Rightarrow d) = 1$.*

Dowód. Niech $\alpha \Rightarrow d$ będzie jednoznaczną regułą klasyfikacyjną wygenerowaną z nadreduktu lokalnego R dla przykładu (u, d) należącego do próbki s. Oznacza to, że dla każdego przykładu (u_i, d_i) takiego, że $d_i \neq d$, istnieje atrybut $a \in R$ rozróżniający obiekty u i u_i, tzn. taki, że $u(a) \neq u_i(a)$. Zatem wszystkie przykłady próbki (u_i, d_i) spełniające formułę $\bigwedge_{a \in R}(a = u(a))$ muszą mieć taką samą decyzję $d_i = d$, a stąd dostajemy, że $[\alpha]_s \cap Class_s(d) = [\alpha]_s$ czyli $confidence_s(\alpha \Rightarrow d) = 1$. Podobnie w przeciwną stronę. Jeśli dla reguły $\alpha \Rightarrow d$ mamy, że $confidence_s(\alpha \Rightarrow d) = 1$, oznacza to, że $[\alpha]_s \cap Class_s(d) = [\alpha]_s$ czyli każdy obiekt spełniający α ma decyzję d, zatem każdy obiekt u_i mający decyzję $d_i \neq d$ musi być odróżniany od obiektu u poprzez jeden z atrybutów $a \in BA_n$ występujących w α. Stąd reguła $\alpha \Rightarrow$d jest jednoznaczna. ■

Rozwiązanie, które przedstawimy w tym rozdziale, wylicza wszystkie bieżące reguły minimalne. Niestety, dwa poniżej przedstawione fakty dowodzą, że problem liczenia reguł minimalnych jest algorytmicznie trudny i że takie uniwersalne rozwiązanie nie zawsze może być wydajne.

FAKT 4.6. *Istnieje próbka, dla której liczba wszystkich reguł minimalnych, a więc i liczba wszystkich reduktów lokalnych jest wykładnicza zarówno względem liczby atrybutów jak i liczby przykładów w próbce.*

Dowód. Zakładamy, że mamy dany zbiór n atrybutów binarnych BA_n=$\{a_1, \ldots, a_n\}$ parzystej wielkości oraz próbkę $s = (u_i, d_i)_{1 \leq i \leq \frac{n}{2}+1}$, gdzie pierwszy element ma wartości wszystkich atrybutów oraz decyzję równą 0: $u_1(a_j) = 0$ i $d_1 = 0$ dla wszystkich $1 \leq j \leq n$, a każdy następny przykład $u_2, \ldots, u_{\frac{n}{2}+1}$ ma wartości dwóch sąsiednich atrybutów i decyzji równe 1: $u_i(a_{2i-1}) = 1$, $u_i(a_{2i}) = 1$, $d_i = 1$, a dla pozostałych atrybutów $a_1, \ldots, a_{2i-2}, a_{2i+1}, \ldots, a_n$ ma wartość 0. W ten sposób każdy podzbiór atrybutów $R \subseteq BA_n$ taki, że z każdej pary a_{2i-1}, a_{2i} dokładnie jeden atrybut należy do R, jest reduktem lokalnym. Oznaczając przez r liczbę wszystkich reduktów lokalnych i przez $m = \frac{n}{2}+1$ liczbę wszystkich obiektów w próbce s otrzymujemy następujące wykładnicze zależności między liczbą reduktów, rozmiarem próbki oraz liczbą atrybutów:

$$r = 2^{m-1} = 2^{\frac{n}{2}} = \left(\sqrt{2}\right)^n$$

Ponieważ każdy redukt lokalny generuje inną regułę minimalną, więc zależność między liczbą reguł minimalnych oraz rozmiarem próbki i liczbą atrybutów jest również wykładnicza. ■

FAKT 4.7. *Problem znalezienia najkrótszej reguły minimalnej jest NP-trudny* [**36**].

Dowód. Ponieważ reguły minimalne oraz redukty lokalne odpowiadają sobie wzajemnie jednoznacznie, dowód możemy przeprowadzić dla problemu szukania najkrótszego reduktu lokalnego. Rozważmy inny problem (ang. HITTING SET PROBLEM): dana jest rodzina podzbiorów C z ustalonego skończonego zbioru S oraz naturalna liczba $K \leq \|S\|$, pytamy, czy istnieje podzbiór $S' \subseteq S$ taki, że $\|S'\| \leq K$ oraz dla każdego zbioru $S_i \in C$ przecięcie $S_i \cap S'$ jest niepuste. Wiadomo, że problem ten jest NP-zupełny [**16**] . Pokażemy, że problem ten ma wielomianowe sprowadzenie do następującego problemu (ang. HITTING REDUCT PROBLEM): dla danej próbki m przykładów z jednym wybranym przykładem z tej próbki (u_0, d_0) oraz naturalnego K pytamy, czy

istnieje redukt lokalny dla (u_0, d_0) rozmiaru mniejszego lub równego K. Sprowadzenie robimy w następujący sposób: dla danej instancji problemu HSP jako zbiór atrybutów w problemie HRP przyjmujemy S i przyjmując dowolne liniowe uporządkowanie zbioru $C = \{S_1, \ldots, S_{\|C\|}\}$ próbkę $\left((u_0, d_0), (u_1, d_1), \ldots, \left(u_{\|C\|}, d_{\|C\|}\right)\right)$ definiujemy w następujący sposób. Pierwszy element ma wszystkie współrzędne i decyzję równą 0:

$$\begin{aligned} u_0(s) &= 0 \text{ dla wszystkich } s \in S \\ d_0 &= 0 \end{aligned}$$

a każdy kolejny element (u_i, d_i) ma decyzję równą 1 i te współrzędne, które należą do zbioru S_i, mają wartość 1, a te współrzędne, które nie należą do S_i, mają wartość 0:

$$\begin{aligned} u_i(s) &= \begin{cases} 1 & \text{dla } s \in S_i \\ 0 & \text{dla } s \notin S_i \end{cases} \\ d_i &= 1 \end{aligned}$$

Przy takim sprowadzeniu zbiór S' jest rozwiązaniem problemu HSP wtedy i tylko wtedy gdy jest reduktem lub nadreduktem lokalnym w HRP. Ponieważ każdy nadredukt zawiera pewien redukt, więc odpowiedź dla problemu HSP jest zawsze taka sama jak dla odpowiadającego mu problemu HRP. Stąd problem HRP jest NP-trudny, więc i problem znalezienia najkrótszego reduktu lokalnego jest NP-trudny. ■

Inkrementacyjny algorytm liczący wszystkie reguły minimalne został podany przez Ziarko i Shan'a w 1994 roku [**40**]. Podczas uczenia utrzymywana jest struktura następujących zbiorów:

- s — aktualna próbka przykładów uczących
- $Rules(d)$ — zbiór reguł minimalnych z decyzją d
- $Cand(d)$ — zbiór reguł-kandydatów z decyzją d

Przed rozpoczęciem uczenia algorytm wykonuje następującą procedurę inicjującą:

ALGORYTM 4.8. initialize

$s := \emptyset$;
dla każdego $d \in V_d$
  $Rules(d) := \{\bigwedge \emptyset \Rightarrow d\}$;
  $Cand(d) := \emptyset$;

W ciągu całego procesu uczenia zbiory utrzymywane przez algorytm zmieniane są dla każdej decyzji niezależnie. Dla każdego nowego przykładu wykonywana jest następująca procedura:

ALGORYTM 4.9. learn(u,d)

$s := s + (u, d)$;
dla każdego $d' \in V_d \setminus \{d\}$
  przenieś każdą $\alpha \Rightarrow d' \in Rules(d')$ taką, że u spełnia α, do $Cand(d')$;
  dopóki w $Cand(d')$ jest reguła $\alpha \Rightarrow d'$ taka, że $[\alpha]_s \cap Class_s(d') \neq \emptyset$
    usuń ją z $Cand(d')$;
    dla każdego atrybutu $a \in A \setminus Attributes(\alpha)$
      $l :=$literał a wykluczający u;
      jeśli $\alpha \wedge l \Rightarrow d'$ nie jest uogólniana przez inną regułę z $Rules(d') \cup Cand(d')$
        $Rules(d') := Rules(d') \cup \{\alpha \wedge l \Rightarrow d'\}$;

Przy każdym nowym przykładzie procedura zmienia najpierw status każdej reguły ze zbioru $Rules(d')$ sprzecznej z nowym przykładem z minimalnej na kandydata. Następnie każdy kandydat jest poprawiany poprzez dodawanie pojedynczego literału usuwającego sprzeczność na nowym obiekcie. Rozszerzona reguła wraca do zbioru $Rules(d')$ pod warunkiem, że nie ma już żadnej innej bardziej ogólnej reguły z tą samą decyzją.

Pokażemy, że opisany algorytm generuje dokładnie wszystkie reguły minimalne:

TWIERDZENIE 4.10. *Przy wyjściu z każdego wywołania procedury learn zbiór* $\bigcup_{d \in V_d} Rules(d)$ *zawiera wszystkie reguły minimalne i tylko takie dla aktualnej próbki* s.

Dowód. Najpierw pokażemy, że reguła minimalna dla każdej bieżącej próbki s jest wygenerowana przez procedurę *learn*. Rozważmy pojedyncze wykonanie procedury dla nowego przykładu (u, d) oraz dowolną regułę minimalną $\alpha \Rightarrow d'$ dla bieżącej próbki

s. Niech s' będzie próbką przed dodaniem nowego przykładu: $s = s' + (u, d)$. Dwie sytuacje są możliwe. Albo reguła $\alpha \Rightarrow d'$ była dobrą regułą minimalną dla poprzedniej próbki s', wtedy była już w zbiorze $Rules(d')$ przed rozpoczęciem procedury $learn$ i pozostała tam przez całe jej wykonanie. Druga możliwość to taka, że reguła $\alpha \Rightarrow d'$ była zbyt specyficzna dla próbki s'. Ale wtedy musi być przynajmniej regułą jednoznaczną dla s', ponieważ zbiór przykładów z s' jest podzbiorem przykładów z s. To oznacza, że istnieje reguła minimalna $\beta \Rightarrow d'$ taka, że $Literals(\beta) \subset Literals(\alpha)$ i reguła ta musi być niespójna z nowym przykładem (u, d), bo inaczej byłaby dobrą reguła minimalną dla s, co zaprzeczałoby minimalności reguły $\alpha \Rightarrow d'$. Zatem na początku procedury reguła $\beta \Rightarrow d'$ jest przenoszona z $Rules(d')$ do $Cand(d')$. Rozważmy moment, kiedy reguła ta jest pobierana do rozszerzenia przez literały wykluczające obiekt u. Decyzje d i d' muszą być różne i jednomian α musi wykluczać u. Niech l będzie takim literałem z α wykluczającym u. Reguła $\beta \wedge l \Rightarrow d'$ jest jednoznaczna dla całej próbki s, a z drugiej strony jest nadal zawarta w minimalnej dla s regule $\alpha \Rightarrow d'$, co oznacza, że obie te reguły muszą być takie same. Jeśli okaże się, że $\beta \wedge l \Rightarrow d'$ jest uogólniana przez innego kandydata $\beta' \Rightarrow d'$, to możemy powtórzyć to rozumowanie dla tego nowego kandydata. Dla ostatniego z takich kandydatów algorytm nie znajdzie już bardziej ogólnego kandydata, więc zostanie on przeniesiony do $Rules(d')$ jako dobra reguła minimalna. Pozostaje jeszcze pokazać, że procedura $learn$ generuje tylko reguły minimalne. Zróbmy najpierw dwie obserwacje. Pierwsza to taka, że każda reguła, która jest generowana przez dodanie do reguły minimalnej dla poprzedniej próbki s' literału wykluczającego nowy obiekt u, jest regułą jednoznaczną dla nowej próbki s. Druga mówi tyle, że każda reguła $\alpha \Rightarrow d'$ minimalna dla nowej próbki s jest albo już zawarta w $Rules(d')$ albo uogólniana przez co namniej jednego kandydata w $Cand(d')$. Stąd, kiedy generowane jest nowe rozszerzenie kandydata z decyzją d', musi to być reguła jednoznaczna, i jeśli nie jest uogólniane przez żadną inną regułę z $Rules(d') \cup Cand(d')$, to jest minimalną regułą jednoznaczną ■

Podstawowy algorytm inkrementacyjny zawiera dwie operacje istotne dla szybkości działania: poszukiwanie reguł pokrywających nowy obiekt u oraz poszukiwanie bardziej ogólnych reguł dla rozwiniętej reguły $\alpha \wedge l \Rightarrow d'$. Operacje te mogą zostać istotnie przyspieszone poprzez zastosowanie drzew TRIE [**11**], [**24**] do przechowywania reguł.

ROZDZIAŁ 5

OCZEKIWANA ZŁOŻONOŚĆ PODSTAWOWEGO ALGORYTMU INKREMENTACYJNEGO

W poprzednim rozdziale pokazaliśmy, że w pesymistycznym przypadku liczba reguł minimalnych jest wykładnicza i w takich sytuacjach podstawowy inkrementacyjny algorytm liczenia reguł minimalnych nie ma zastosowania ze względu na nieakceptowalny czas działania. Jednak lepszym wykładnikiem mozliwości praktycznego zastosowania algorytmu jest jego złożoność oczekiwana i dlatego w tym rozdziale przedstawimy szacowanie oczekiwanej liczby operacji sprawdzenia pokrywania obiektu przez regułę w modelu obliczeń, w którym rozmiar danych wejściowych jest scharakteryzowany przez liczbę atrybutów binarnych n oraz długość próbki m.

Przypomnijmy, że zbiór n atrybutów binarnych oznaczamy przez BA_n. Jako dziedzinę problemu liczenia reguł minimalnych przyjmujemy przestrzeń probabilistyczną będącą produktem dwóch podprzestrzeni probablistycznych. Pierwsza to przestrzeń wszystkich funkcji boolowskich $F_n = V_d^{U_n}$ z jednorodnym rozkładem czyli takim, w którym prawdopodobieństwo każdej funkcji $f \in F_n$ jest takie samo i wynosi $p(f) = \frac{1}{\|V_d\|^{\|U_n\|}} = \frac{1}{\|V_d\|^{2^n}}$. Druga to przestrzeń $S_m = U_n^m$ wszystkich próbek $s : m \to U_n$ długości m, również z rozkładem jednorodnym, w którym prawdopodbieństwo każdej próbki $s \in S_m$ wynosi $p(s) = \frac{1}{\|U_n\|^m} = \frac{1}{(2^n)^m}$.

Zauważmy, że najczęstszą operacją wykonywaną przez algorytm jest sprawdzenie, czy reguła pokrywa nowy obiekt. Niech $cost : F_n \times S_m \to \mathbb{N}$ będzie taką zmienną losową, że dla danej funkcji boolowskiej $f \in F_n$ i danej próbki $s \in S_m$ wartość $cost(f, s)$ oznacza liczbę wszystkich operacji sprawdzenia pokrywania obiektu przez regułę wykonywanych w kolejnych wywołaniach procedury $learn\,(s(i), f(s(i)))$.

TWIERDZENIE 5.1. *Załóżmy, że* $2 \leq n \leq m \leq \sqrt{\|U\|}$. *Wartość oczekiwana* EX *liczby* $cost$ *wszystkich operacji sprawdzenia pokrywania obiektu przez regułę w podstawowym*

algorytmie inkrmentacyjnym ma następujące oszacowanie asymptotyczne

$$EX(cost) = O\left(\frac{\|V_d\|\, n}{\sqrt{\lceil \lg m \rceil}} m^{2+\lg \frac{n}{\sqrt{\lceil \lg m \rceil}}}\right)$$

Dowód. Na początku ustalamy decyzję $v_d \in V_d$ i liczymy wartość oczekiwaną dla ustalonej decyzji. Niech r będzie dowolną regułą $a_{i_1} = b_1 \wedge \ldots \wedge a_{i_k} = b_k \Rightarrow d = v_d$ i niech $cost_r : F_n \times S_m \to \mathbb{N}$ będzie zmienną losową, która dla danej funkcji boolowskiej $f \in F_n$ i danej próbki $s \in S_m$ oznacza liczbę wszystkich operacji sprawdzenia pokrycia obiektów przez regułę r przy wykonywaniu procedury *learn* dla kolejnych pzykładów $(s(i), f(s(i)))$. Obliczanie wartości oczekiwanej $EX(cost)$ możemy sprowadzić do liczenia wartości oczekiwanej $EX(cost_r)$ dla pojedynczych reguł:

$$\begin{aligned} EX(cost) &= \sum_{f \in F_n} p(f) \sum_{s \in S_m} p(s) \sum_{r \in R} cost_r(f,s) \\ &= \sum_{r \in R} \sum_{f \in F_n} p(f) \sum_{s \in S_m} p(s) \cdot cost_r(f,s) = \sum_{r \in R} EX(cost_r) \end{aligned}$$

Zauważmy najpierw, że jeśli reguła r długości k pojawia się w czasie uczenia próbki s jako reguła redukcyjna, musi być minimalna czyli dla każdego jej deskryptora $a_{i_j} = b_j$ w próbce s występuje obiekt, który nie spełnia j-tego deskryptora, spełnia wszystkie pozostałe deskryptory reguły r i ma decyzję inną niż r. Przez U_n^j oznaczmy zbiór obiektów spełniających wszystkie deskryptory po lewej stronie reguły r z wyjatkiem j-tego. Na $n-k$ atrybutach wartości tych obiektów są dowolne, zatem rozmiar każdego zbioru U_n^j wynosi $\|U_n^j\| = 2^{n-k}$. Możemy teraz powiedzieć, że reguła r pojawia się w momencie, kiedy z każdego ze zbiorów $U_n^1, \ldots, U_n^k$ wystąpi przynajmniej jeden obiekt z decyzją inną niż decyzja reguły r. Niech B_j będzie zmienną losową oznaczającą liczbę obiektów z U_j z decyzją inną niż decyzja reguły r i niech A_j będzie zmienną losową oznaczającą liczbę wszystkich obiektów z U_j występujących w próbce długości m. Prawdopodobieństwo pojawienia się reguły redukcyjnej r można teraz oszacować przez

$$P(r \text{ occured}) \leq P(B_1 \neq 0, \ldots, B_k \neq 0)$$

Ponieważ zdarzenie $B_j \neq 0$ implikuje zdarzenie $A_j \neq 0$, więc możemy to z kolei oszacować przez

$$P(r \text{ occured}) \leq P(A_1 \neq 0, \ldots, A_k \neq 0)$$

Aby oszacować $P(A_1 \neq 0, \ldots, A_k \neq 0)$, policzymy najpierw wartość $P(A_j \neq 0 | A_1 + \ldots + A_{j-1} = p)$ która określa prawdobodobieństwo, że w próbce pojawi się co najmniej jeden obiekt z U_j pod warunkiem, że wystąpiło dokładnie $p > 0$ obiektów ze zbiorów $U_1, \ldots, U_{j-1}$. Prawdopodobieństwo, że w próbce długości m mającej już i miejsc zajętych obiektami spoza U_j, w następnym wybranym miejscu nie pojawi się obiekt z U_j wynosi:

$$\frac{2^n - 2^{n-k}}{2^n} = 1 - \frac{1}{2^k}$$

Stąd:

$$\begin{aligned} P(A_j \neq 0 | A_1 = p_1, \ldots, A_{j-1} = p_{j-1}) &= 1 - P(A_j = 0 | A_1 = p_1, \ldots, A_{j-1} = p_{j-1}) \\ &= 1 - \prod_{i=p_1+\cdots+p_{j-1}}^{m-1} \left(\frac{2^n - 2^{n-k}}{2^n} \right) = 1 - \prod_{i=p_1+\cdots+p_{j-1}}^{m-1} \left(1 - \frac{1}{2^k} \right) \\ &\leq 1 - \left(1 - \frac{1}{2^k} \right)^m \leq 1 - \left(1 - \frac{m}{2^k} \right) = \frac{m}{2^k} \end{aligned}$$

Do policzenia $P(A_j \neq 0 | A_1 \neq 0, \ldots, A_k \neq 0)$ wystarczy teraz posumować $P(A_j \neq 0 | A_1 = p_1, \ldots, A_{j-1} = p_{j-1})$ po wszystkich mozliwych kombinacjach wartości $p_1, \ldots, p_{j-1}$:
$P(A_j \neq 0 | A_1 \neq 0, \ldots, A_{j-1} \neq 0) =$
$= \sum_{p_1 \geq 1, \ldots, p_{j-1} \geq 1} P(A_1 = p_1, \ldots, A_{j-1} = p_{j-1} | A_1 \neq 0, \ldots, A_{j-1} \neq 0) \cdot \cdot P(A_j \neq 0 | A_1 = p_1, \ldots, A_{j-1} = p_{j-1}) \leq$
$\leq \frac{m}{2^k} \sum_{p_1 \geq 1, \ldots, p_{j-1} \geq 1} P(A_1 = p_1, \ldots, A_{j-1} = p_{j-1} | A_1 \neq 0, \ldots, A_{j-1} \neq 0) = \frac{m}{2^k}$ Teraz możemy już oszacować prawdopodobieństwo, że z każdego U_j pojawi się przynajmniej jeden obiekt:

$$\begin{aligned} P(A_1 \neq 0, \ldots, A_k \neq 0) = \\ = P(A_1 \neq 0) P(A_2 \neq 0 | A_1 \neq 0) \cdots P(A_k \neq 0 | A_1 \neq 0, \ldots, A_{k-1} \neq 0) \leq \left(\frac{m}{2^k} \right)^k \end{aligned}$$

Wartość oczekiwana zmiennej losowej $cost_r$ dla reguł tej samej długości jest taka sama, a reguł o długości k jest $\|V_d\| \binom{n}{k} \cdot 2^k$, wystarczy więc przesumować teraz po wszystkich

długościach, żeby otrzymać wartość oczekiwaną dla wszystkich reguł. Niech $l = \lceil \lg m \rceil$. Prawdopodobieństwo pojawienia się reguły długości $k > l$ szacujemy przez $P(A_1 \neq 0, \ldots, A_k \neq 0)$, a dla pozostałych szacujemy przez 1. Ponadto szacujemy rozrzutnie, że jeśli regule uda sie pojawić, to jest porównywana ze wszystkimi m obiektami w próbce:

$$EX(cost) = \sum_{r \in R} EX(cost_r) \leq \|V_d\| \sum_{k=0}^{l} \binom{n}{k} 2^k m + \|V_d\| \sum_{k=l+1}^{n} \binom{n}{k} 2^k m \left(\frac{m}{2^k}\right)^k$$

Korzystając z założenia $l \leq \frac{n}{2}$ pokażemy teraz zależności pomiędzy sasiednimi składnikami w każdej części sumy. Dla $k \leq l$:

$$\frac{\binom{n}{k-1} 2^{k-1} m}{\binom{n}{k} 2^k m} = \frac{k}{n-k+1} \frac{1}{2} \leq \frac{1}{2}$$

i dla $k \geq l$:

$$\frac{\binom{n}{k+1} 2^{k+1} m \left(\frac{m}{2^{k+1}}\right)^{k+1}}{\binom{n}{k} 2^k m \left(\frac{m}{2^k}\right)^k} = \frac{n-k}{k+1} 2 \frac{m}{2^{2k+1}} \leq \frac{n}{2} 2 \frac{m}{2^{2k+1}} = \frac{n}{2^k} \frac{m}{2^k} \frac{1}{2} \leq \frac{1}{2}$$

Zatem obie cześci sumy można oszacować przez ciąg geometryczny:

$$\begin{aligned} EX(cost) &\leq \|V_d\| \left(\sum_{k=0}^{l} \binom{n}{l} 2^l m \left(\frac{1}{2}\right)^{l-k} + \sum_{k=l+1}^{n} \binom{n}{l} 2^l m \left(\frac{m}{2^l}\right)^l \left(\frac{1}{2}\right)^{k-l} \right) \\ &\leq \|V_d\| \left(2\binom{n}{l} 2^l m + \binom{n}{l} 2^l m \left(\frac{m}{2^l}\right)^l \right) \\ &\leq \|V_d\| \left(4m^2 \binom{n}{l} + m^2 \binom{n}{l} \right) \leq 3 \|V_d\| m^2 \binom{n}{\lceil \lg m \rceil} \end{aligned}$$

W otrzymanym oszacowaniu uprościmy jeszcze czynnik $\binom{n}{\lceil \lg m \rceil}$:

$$\binom{n}{\lceil \lg m \rceil} = \frac{n(n-1)\cdots(n-\lceil \lg m \rceil + 1)}{\lceil \lg m \rceil (\lceil \lg m \rceil - 1)\cdots 1} \leq \frac{n^{\lceil \lg m \rceil}}{\lceil \lg m \rceil (\lceil \lg m \rceil - 1)\cdots 1} = 2^{\lg \frac{n^{\lceil \lg m \rceil}}{\lceil \lg m \rceil (\lceil \lg m \rceil - 1)\cdots 1}}$$

Oszacujemy najpierw wykładnik wykorzystując wklęsłość funkcji lg, a dokładniej zależność $\lg i \geq \frac{i-1}{x-1} \lg x$ dla $1 \leq i \leq x$:

$$\begin{aligned}
\lg \frac{n^{\lceil \lg m \rceil}}{\lceil \lg m \rceil (\lceil \lg m \rceil - 1) \cdots 1} &= \lg n^{\lceil \lg m \rceil} - \lg (\lceil \lg m \rceil (\lceil \lg m \rceil - 1) \cdots 1) \\
&= \lceil \lg m \rceil \lg n - \sum_{i=1}^{\lceil \lg m \rceil} \lg i \leq \lceil \lg m \rceil \lg n - \sum_{i=1}^{\lceil \lg m \rceil} \frac{i-1}{\lceil \lg m \rceil - 1} \lg \lceil \lg m \rceil \\
&= \lceil \lg m \rceil \lg n - \frac{\lceil \lg m \rceil (\lceil \lg m \rceil - 1)}{2 (\lceil \lg m \rceil - 1)} \lg \lceil \lg m \rceil \\
&\leq \lceil \lg m \rceil \lg n - \lceil \lg m \rceil \lg \sqrt{\lceil \lg m \rceil} = \lceil \lg m \rceil \lg \frac{n}{\sqrt{\lceil \lg m \rceil}}
\end{aligned}$$

Wracamy teraz do szacowania dwumianu:

$$\binom{n}{\lceil \lg m \rceil} \leq 2^{\lceil \lg m \rceil \lg \frac{n}{\sqrt{\lceil \lg m \rceil}}} \leq (2m)^{\lg \frac{n}{\sqrt{\lceil \lg m \rceil}}} = 2^{\lg \frac{n}{\sqrt{\lceil \lg m \rceil}}} m^{\lg \frac{n}{\sqrt{\lceil \lg m \rceil}}} = \frac{n}{\sqrt{\lceil \lg m \rceil}} m^{\lg \frac{n}{\sqrt{\lceil \lg m \rceil}}}$$

Podstawiając ten wynik możemy ostatecznie oszacować oczekiwaną liczbę operacji:

$$EX(cost) \leq 6 \, \|V_d\| \, \frac{n}{\sqrt{\lceil \lg m \rceil}} m^{2+\lg \frac{n}{\sqrt{\lceil \lg m \rceil}}} = O\left(\frac{\|V_d\| \, n}{\sqrt{\lceil \lg m \rceil}} m^{2+\lg \frac{n}{\sqrt{\lceil \lg m \rceil}}} \right)$$

■

W pokazanym twierdzeniu skorzystalismy z założenia, że $2 \leq n \leq m \leq \sqrt{\|U\|}$, jednakże nie jest to istotne ograniczenie, gdyż na ogół rzeczywiste zbiory danych spełniają ten warunek.

Pokazaliśmy, że w opisanym w poprzednim rozdziale algorytmie średnia liczba operacji sprawdzenia pokrywania obiektu przez regułę jest podwykładnicza, tzn. wykładnik tej liczby jest co najwyżej polilogarytmiczny. Fakt ten ma istotne znaczenie praktyczne, gdyż dla wielu rzeczywistych danych wartość wykładnika $2+\lg \frac{n}{\sqrt{\lceil \lg m \rceil}}$ jest na tyle mała, że algorytm wykonywany na odpowiednio szybkich maszynach może mieć akceptowalny czas wykonania.

Należy jednak zwrócić uwagę na fakt, że przedstawiony algorytm zawiera jeszcze inną kosztowną operację: sprawdzenie dla nowej reguły, czy istnieje reguła bardziej ogólna. Operacja ta jest wykonywana znacznie rzadziej niż operacja sprawdzenia pokrycia obiektu przez regułę, niemniej koszt jej wykonania jest bardziej złożony, zależny od

wykorzystanych struktur danych i przeprowadzenie kompletnej analizy złożoności algorymu wymaga oszacowania złożoności także tej operacji.

ROZDZIAŁ 6

INKREMENTACYJNY ALGORYTM LICZENIA REGUŁ MINIMALNYCH OPARTY NA MONOTONICZNYCH OGRANICZENIACH

Własności podstawowego algorytmu inkrementacyjnego pokazane w dwóch poprzednich rozdziałach wskazują, że przy wielu zbiorach danych czas działania tego algorytmu może okazać się zbyt duży. W tym rozdziale przedstawimy zmodyfikowaną wersję tego algorytmu, która pozwala na istotne przyspieszenie działania algorytmu, i jednocześnie, jak pokazują eksperymenty, zachowuje jakość oryginalnego algorytmu.

Zmodyfikowana wersja algorytmu bazuje na redukcji przestrzeni poszukiwań reguł minimalnych do reguł spełniających monotoniczne ograniczenie zdefiniowane w następujący sposób:

DEFINICJA 6.1. Monotoniczne ograniczenie C to dowolny podzbiór przestrzeni wszystkich reguł klasyfikacyjnych nad zbiorem atrybutów binarnych BA_n spełniający następujący warunek: jeśli $\alpha \Rightarrow d \in C$, to dla każdego podzbioru B literałów występujących w α reguła $\bigwedge B \Rightarrow d$ również należy do C.

Istotnym elementem algorytmu jest postać przyjętego ograniczenia monotonicznego, przy czym zakładamy zawsze, że monotoniczne ograniczenie jest efektywnie weryfikowalnym zbiorem dynamicznie zależnym od danej próbki s, tzn. istnieje algorytm który otrzymawszy na wejściu dowolną próbkę s i dowolną regułę r daje odpowiedź, czy reguła r należy do ograniczenia definiowanego przez próbkę s.

Poniżej przedstawiamy dwa podstawowe przykłady monotonicznych ograniczeń oparte na mierze pokrycia *coverage* dla reguł klasyfikacyjnych. Pierwszy typ obejmuje te reguły klasyfikacyjne, których pokrycie przekracza ustaloną wartośc progową:

$$C_s^{coverage>\theta} = \{r : coverage_s(r) > \theta\}$$

drugi natomiast zawiera zawsze ustaloną liczbę k najlepszych reguł minimalnych dla każdej klasy decyzyjnej, czyli wymagany próg dla danej klasy jest równy wartości pokrycia k-tej co do wielkości tego parametru reguły minimalnej:

$$C_s^{best-k} = \{r : \|\{r' : r' \text{ minimalna oraz } coverage_s(r') > coverage_s(r)\}\| < k\}$$

W inkrementacyjnym algorytmie opartym na monotonicznych ograniczenich zbiór $Rules(d)$ przechowuje tylko reguły minimalne spełniające ograniczenie C, a zbiór kandydatów jest tym razem podzielony na dwa zbiory: $CCand(d)$, który zawiera wszystkich kandydatów spełniających ograniczenie C oraz $nonCCand(d)$, który zawiera wszystkich kandydatów nie spełniających C.

ALGORYTM 6.2. learn(u,d)

$s := s + (u, d)$;
uaktualnij ograniczenie C;
dla każdego $d' \in V_d$
krok 1:
  przenieś każdą regułę $r \in Rules(d')$ taką, że $r \notin C$ do $nonCCand(d')$;
  przenieś każdą regułę $r \in Rules(d')$ sprzeczną z (u, d) do $CCand(d')$;
  przenieś każdą jednoznaczną regułę $r \in nonCCand(d')$ taką, że $r \in C$ do $Rules(d')$;
  przenieś każdą regułę $r \in nonCCand(d')$ taką, że $r \in C$ do $CCand(d')$;
krok 2:
  dopóki $CCand(d') \neq \emptyset$
    usuń dowolną regułę $\alpha \Rightarrow d'$ z $CCand(d')$;
    znajdź przykład (u'', d'') sprzeczny z $\alpha \Rightarrow d'$;
    dla każdego atrybutu $a \in A \setminus Attributes(\alpha)$
      l :=literał a wykluczający u'';
      jeśli $\alpha \wedge l \Rightarrow d'$ nie ma ogólniejszej reguły
        w $Rules(d') \cup CCand(d') \cup nonCCand(d')$ to
        jeśli $\alpha \wedge l \Rightarrow d' \notin C$ to $nonCCand(d') := nonCCand(d') \cup \{\alpha \wedge l \Rightarrow d'\}$
        lub jeśli $\alpha \wedge l \Rightarrow d'$ jest jednoznaczna
          to $Rules(d') := Rules(d') \cup \{\alpha \wedge l \Rightarrow d'\}$
        lub w przeciwnym przypadku $CCand(d') := CCand(d') \cup \{\alpha \wedge l \Rightarrow d'\}$;

Podobnie jak w podstawowym algorytmie, startujemy ze zbiorami $Rules(d)$ zawierającymi najbardziej ogólne reguły — jedną dla każdej decyzji. Teraz jednak procedura *learn* rozszerza tylko kandydatów ze zbioru $CCand(d)$ pozostawiając kandydatów z $nonCCand(d)$ bez zmian.

Na poczatku procedury *learn* zakładamy, że zbiory $Rules(d')$ zawierają wszystkie reguły minimalne spłniające C, a $nonCCand(d')$ zawierają wszystkie wygenerowane dotąd reguły nie spełniające C, oba przypadki w odniesieniu do stanu próbki s przed dodaniem nowego przykładu (u, d). Zbiory $CCand(d')$ powinny być puste. Następnie wykonywane są dwa kroki.

W kroku 1 algorytm poprawia aktulaną zawartość zbiorów $Rules(d')$, $CCand(d')$ i $nonCCand(d')$ zgodnie ze zmianami w próbce s i ograniczeniu C: reguła minimalna dla próbki przed dodaniem przykładu (u, d) może stracić tą własność po jego dodaniu, a zmodyfikowane ograniczenie C może objąć istniejących kandydatów ze zbioru $nonCCand(d)$ i wykluczyć poprzednio pokrywane reguły minimalne.

Czas wykonania kroku 1 będzie istotnie zależeć od przyjętego ograniczenia. Rozważmy dwa przykłady: pierwsze skrajnie słabe ograniczenie $C_1 = C^{coverage>0}$ oraz drugie $C_2 = C^{coverage>\theta}$ z dowolną wartością θ z przedziału $(0, 1)$. W przypadku C_1 krok 1 redukuje się do dwóch akcji: pierwsza to przeniesienie wszystkich reguł ze zbiorów $Rules(d')$ sprzecznych z nowym przykładem do zbiorów $CCand(d')$, druga to znalezienie wszystkich reguł w $nonCCand(d')$ pokrywających nowy obiekt i przeniesienie ich do $CCand(d')$. W przypadku C_2 migracja reguł we wszystkich wyliczonych w procedurze *learn* kierunkach jest możliwa, zatem proces adaptacji do zmian jest dużo bardziej złożony. Innym problemem jest sprawdzanie spełnialności ograniczenia przez regułę. W praktyce wymaga to utrzymywania aktualnej wartości pokrycia dla wszystkich reguł, co prowadzi do postawienia nowego zadania znalezienia struktury obiektów dobrze przyspieszającej liczenie tego parametru. Jednym z mozliwych rozwiązań jest zastosowanie bardzo wydajnych dla pewnej klasy zbiorów obiektów diagramów OBDD [**10**], [**30**], [**31**], [**41**]. Diagramy OBDD są strukturą danych interpetowaną jako funkcje boolowskie, w których węzły odpowiadają zmiennym boolowskim, a rozgałęzienia wychodzące z węzłów wartościowaniom tych zmiennych. W opisanym algorytmie diagramy OBDD mogłyby definiować funkcję boolowską $f \in F_n$, w której

wartość $f(u)$ na wektorze wartości boolowskich $u \in \{0,1\}^{BA_n}$ byłaby równa decyzji d przykładu z próbki, jeśli w próbce występuje przykład (u,d) lub pewnej ustalonej wartości nie należącej do V_d oznaczącej brak przykładu, jeśli taki przykład nie występuje w próbce.

Przy użyciu ograniczenia typu $best-k$ problem spełnialności ograniczenia jest jeszcze trudniejszy. W tym wypadku rozwiązaniem może być przyjęcie rankingu reguł minimalnych na bazie ich aktualnego zbioru i wykonywanie kroku 2 zawsze dla reguły z najwyższą wartością pokrycia *coverage* z jednoczesnym poprawianiem rankingu za każdym razem, kiedy nowa reguła minimalna jest znajdowana. To gwarantuje, że przy wyjściu z procedury ranking dla k najlepszych reguł minimalnych będzie poprawny.

W kroku 2 algorytm rozszerza wszystkich kandydatów, którzy spełniają aktualne ograniczenie C. Krok 2 różni się istotnie w dwóch miejscach w stosunku do podstawowego algorytmu inkrementacyjnego. Po pierwsze, kandydat może być sprzeczny z dowolnym przykładem w próbce, niekoniecznie ostatnim, w związku z czym trzeba dodać szukanie takiego przykładu. Po drugie, reguła, która powstaje po rozszerzeniu, może ponownie trafić do zbioru kandydatów, gdyż może pozostawać sprzeczna z innymi przykładami w próbce. Jak wspomnieliśmy, w kroku 2 pojawiła się jedna nowa operacja: szukanie dla reguły-kandydata sprzecznego przykładu (u'', d''). Możliwe usprawnienia to jak poprzednio korzystanie z diagramów OBDD lub prostsze polegające na tym, że z każdą rozszerzoną regułą $\alpha \wedge l \Rightarrow d'$ zapamiętywana jest pozycja w próbce s, na której sprzeczny przykład został znaleziony dla poprzedniej reguły $\alpha \Rightarrow d'$, i szukanie sprzecznego przykładu, jeśli rozszerzona reguła $\alpha \wedge l \Rightarrow d'$ stanie sie nowym kandydatem, od tego miejsca. Ponieważ każdy przykład zgodny z regułą $\alpha \Rightarrow d'$ jest również zgodny z każdym rozszerzeniem tej reguły, opisane usprawnienie zachowuje poprawność.

Następujący fakt implikuący poprawność opisanego algorytmu jest prawdziwy:

TWIERDZENIE 6.3. *Przy wyjściu z każdego wywołania procedury learn zbiór* $\bigcup_{d \in V_d} Rules(d)$ *zawiera wszystkie reguły minimalne spełniające* C *i tylko takie dla aktualnej próbki* s. *Ponadto, dla każdej reguły minimalnej* r *z decyzją* d *nie spełniającej* C *istnieje przynajmniej jedna bardziej ogólna niż* r *reguła klasyfikacyjna w zbiorze* $nonCCand(d)$.

Dowód. Dowód poprawności inkrementacyjnego algorytmu bazującego na monotonicznych ograniczeniach jest analogiczny do prostego algorytmu inkrementacyjnego. Następujący niezmienniek jest zawszy prawdziwy: każda reguła minimalna z decyzją d' dla bieżącej zawartości próbki s albo znajduje się w zbiorze $Rules(d')$ albo istnieje bardziej ogólna od niej reguła zawarta w sumie $CCand(d') \cup nonCCand(d')$. Na początku procedury *learn* niezmiennik jest spełniony, ponieważ każda reguła minimalna dla bieżącej próbki s jest równa lub uogólniana przez pewną regułę minimalną dla poprzedniej próbki s', a własność niezmiennicza przenosi się na bardziej specyficzne reguły. Następnie każda reguła-kandydat $\alpha \Rightarrow d'$ jest rozszerzana przez wszystkie literały wykluczające pewien obiekt u'' niezgodny z tą regułą. Zauważmy, że każda reguła minimalna $\beta \Rightarrow d'$ pokrywana przez regułę-kandydata $\alpha \Rightarrow d'$ musi zawierać co najmniej jeden literał l wykluczający u'' czyli musi być pokrywana także przez pewne rozszerzenie $\alpha \wedge l \Rightarrow d'$, które to spełnia wymagany niezmiennik dla $\beta \Rightarrow d'$. Z drugiej strony każda wygenerowana reguła zostaje zachowana tylko wtedy, kiedy nie jest uogólniana przez inną regułę. To zapewnia, że jeśli algorytm zapamiętuje regułę, to ona musi być minimalna. Ponieważ wszyscy kandydaci spełniający ograniczenie C są rozszerzani, zatem wszystkie reguły minimalne spełniające C muszą być wygenerowane. ■

Istotną wadą algorytmu dla bardzo dużej klasy monotonicznych ograniczeń jest fakt, że nie wykonuje on żadnej redukcji reguł. Generowanie coraz większej liczby nowych reguł prowadzi do coraz dłuższego czasu przeszukiwania zbioru reguł, a ostatecznie do wyczerpania pamięci. W celu uniknięcia tego można zastosować następujące rozwiązanie. Za każdym razem, kiedy reguła $\alpha \Rightarrow d'$ jest wstawiana do zbioru $nonCCand(d')$, jest jednocześnie maksymalnie redukowana:

ALGORYTM 6.4. reduce($\alpha \Rightarrow d'$)

zredukuj regułę $\alpha \Rightarrow d'$ do $\beta \Rightarrow d'$
gdzie β jest dowolnym minimalnym jednomianem ogólniejszym od α
takim, że $\beta \Rightarrow d' \notin C$;

Przedstawione rozwiązanie stosuje się do wszystkich ograniczeń mających własność "ściągania", to znaczy, że nowy przykład w próbce może spowodować wykluczenie z ograniczenia reguły, która wcześniej spełniała to ograniczenie. Przykładem ograniczenia

"ściągającego" jest $C^{coverage>\theta}$ dla dowolnego dodatniego θ, natomiast ograniczenie $C^{coverage>0}$ nie ma tej własności.

Taka modyfikacja jest jednak źródłem nowego problemu, mianowicie "migotania" reguł, tzn. pojedyncza reguła może być generowana i redukowana wielokrotnie podczas uczenia się nowych przykładów i dynamicznych zmian ograniczenia C. W przypadku, gdy przy każdej generacji reguły algorytm wykonuje złożone obliczenia jak np. liczenie pokrycia, powtarzanie obliczeń znacznie zwalnia działanie. W związku z tym przedstawimy dwie metody radzenia sobie z tym problemem.

Pierwsza polega na utrzymywaniu buforów. Bufor $BufExt$ przechowuje te reguły, na których operacja rozszerzenia została już wykonana, a bufor $BufRed$ przechowuje reguły, które zostały zredukowane. Bufory będą zazwyczaj zbyt ograniczone, by przechowywać wszystkie reguły wygenerowane w czasie uczenia, w związku z tym muszą stosować pewną miarę do oceny, dla których reguł ponowne użycie w niedalekiej przyszłości jest najbardziej prawdopodobne. Dla ograniczeń opartych na pokryciu można użyć tej właśnie miary. Następujące procedury są zawsze wykonywane, gdy reguła jest odpowiednio rozwijana lub redukowana:

ALGORYTM 6.5. saveExtended(r)

jeśli bufor $BufExt$ nie jest pełny to dodaj r do $BufExt$;
wpp. jeśli $coverage_s(r) < \max_{r' \in BufExt} coverage_s(r')$
    to zastąp regułę z maksymalnym $coverage$ w $BufExt$ przez r ;

ALGORYTM 6.6. saveReduced(r)

jeśli bufor $BufRed$ nie jest pełny to dodaj r do $BufRed$;
wpp. jeśli $coverage_s(r) > \min_{r' \in BufRed} coverage_s(r')$
    to zastąp regułę z minimalnym $coverage$ w $BufRed$ przez r ;

Kiedy procedura *learn* potrzebuje policzyć parametry dla nowo wygenerowanej lub zredukowanej reguły, najpierw sprawdza, czy nie ma jej w odpowiednim buforze i dopiero, jeśli jej tam nie znajdzie, wykonuje obliczenia. Czynnikiem decydującym o jakości przyspieszenia jest wielkość buforów.

Inne rozwiązanie redukujące "migotanie" reguł to grupowanie przykładów. Zamiast uczenia się każdego przykładu osobno, algorytm otrzymuje za każdym razem dużą

grupę przykładów. Proces uczenia dla dużej grupy przykładów może trwać bardzo długo i blokować niedopuszczalnie długo proces klasyfikacji. Można to rozwiązać w następujący sposób. Procedura *learn* ma dwa kroki. Pierwszy trwa zawsze bardzo krótko w porównaniu z drugim, więc wystarczającym rozwiązaniem jest dodanie możliwości zatrzymania procedury w drugim kroku za każdym razem, kiedy procedura klasyfikacyjna jest wołana. Taki algorytm wymaga od procesu klasyfikacji umiejętności użycia reguł o trafności mniejszej niż 1. Dobra strategia wyboru kolejnych reguł do rozwinięcia może mieć tu decydujące znaczenie. Ponieważ reguły zachowujące wyższą trafność po dodaniu nowej grupy przykładów są potencjalnie bardziej wiarygodne, dobrą strategią wydaje się rozpocząć rozszerzanie od reguł z najwyższą trafnością i zmierzać w kierunku reguł o coraz niższej wartości tego parametru. W ten sposób bardziej wiarygodne reguły są szybciej adaptowane do nowego zbioru przykładów.

Ostatnie rozwiązanie jest również dobrą podstawą do obliczeń współbieżnych. Rozważmy trzy procesy: proces 1 uaktualnia zawartość zbiorów $Rules(d')$, $CCand(d')$ i $nonCCand(d')$ oraz oblicza parametry reguł za każdym razem, kiedy grupa nowych przykładów jest wyuczana, proces 2 generuje nowe reguły, a proces 3 jest procesem klasyfikacyjnym. Proces 1 i 3 wykluczają się wzajemnie, natomiast proces 2 może być wykonywany asynchronicznie w następujący sposób. Po pierwsze, proces ten nie musi czekać do momentu, kiedy proces 1 się zakończy, lecz może działać, jeśli tylko jest jakikolwiek kandydat do rozszerzania w $CCand(d')$. Po drugie, nawet, jeśli proces klasyfikujący w dane chwili działa, proces 2 może ciągle rozszerzać reguły nie pokrywające klasyfikowanych obiektów.

W sytuacji, kiedy zbiory danych są zbyt duże, aby opisany algorytm zastosować bezpośrednio do wyuczenia wszystkich przykładów, można zastosować pewną jego modyfikację.

Załóżmy, że dany jest duży zbiór przykładów s. Algorytm działa podobnie, tzn. wykonuje procedurę *learn* dla kolejnych przykładów w s. Jednak w przypadku dużych zbiorów czas uczenia dla całej próbki byłby bardzo długi i blokowałby klasyfikator, w związku z tym można zastosować następujące rozwiązanie. Za każdym razem, kiedy przychodzi żądanie klasyfikacji obiektów, procedura *learn* jest wstrzymywana i uruchamiany jest proces klasyfikacji, który korzysta zawsze z bieżącego zbioru reguł.

Przed rozpoczęciem klasyfikacji algorytm oblicza parametry jakościowe reguł: trafność i pokrycie, w oparcu o całą próbkę s.

ALGORYTM 6.7. classify(u)

dla każdego $d \in V_d$
  dla każdej reguły $r \in Rules(d) \cup CCand(d) \cup nonCCand(d)$ pokrywającej u
    oblicz $confidence_s(r)$ i $coverage_s(r)$;
wylicz decyzję z $\bigcup_{d \in V_d} \{\alpha \Rightarrow d \in Rules(d) \cup CCand(d) \cup nonCCand(d) : u \in [\alpha]_s\}$;

Liczenie parametrów dla zbioru reguł zajmuje znacznie mniej czasu niż generowanie tego zbioru, niemniej liczenie ich za każdym razem, kiedy procedura klasyfikacyjna jest wołana, jest zwykle zbyt kosztowne dla dużych zbiorów przykładów. Aby uniknąć tego problemu, algorym może wykonywać następujące operacje:

- dla każdego obiektu do klasyfikacji oblicza parametry tylko dla reguł pokrywających obiekt;
- raz policzone parametry dla reguły mogą być zachowane tak długo jak długo reguła jest trzymana w jednym ze zbiorów $Rules(d)$, $nonCCand(d)$, $CCand(d)$;
- niezależnie od wykonań procedury klasyfikacyjnej procedura wyuczająca może w regularnych odstępach wyliczać parametry reguł wygenerowanych od momentu poprzedniego liczenia parametrów;
- różne struktury danych przyspieszające liczenie parametrów mogą być użyte dla reguł i obiektów.

Przedstawione rozwiązanie wymaga od zastosowanego klasyfikatora możliwości stosowania reguł z trafnością mniejszą niż 1.

W tym rozdziale opisaliśmy inkrementacyjny algorytm klasyfikacyjny oraz różne metody redukcji złożoności czasowej i pamięciowej tego algorytmu. W przypadku, gdy takie szczególne rozwiązania zawodzą lub nie dają się zastosować, pełne przeszukiwanie przestrzeni reguł może być zawsze zastąpione dowolnym przeszukiwaniem heurystycznym. Inne rozwiązanie dla dużych zbiorów — oparte na próbkowaniu i reduktach dynamicznych — można znaleźć w pracy [**5**].

ROZDZIAŁ 7

ADAPTACYJNE DEFINIOWANIE FUNKCJI BOOLOWSKICH W OPARCIU O REGUŁY KLASYFIKACYJNE

W poprzednich rozdziałach przedstawiliśmy różne metody liczenia reguł klasyfikacyjnych. Przedmiotem tego rozdziału jest opis metody definiowania funkcji bolowskich na bazie ustalonego zbioru reguł oraz jakościowe i wydajnościowe wyniki doświadczeń z zastosowania tej metody oraz opisanych algorytmów liczenia reguł do rzeczywistych zbiorów danych. W dalszej części rozdziału algorytm defininiowania funkcji boolowskiej będziemy nazywać algorytmem klasyfikacyjnym.

Dla ustalonej próbki treningowej $s = ((u_1, d_1), \ldots, (u_m, d_m))$ algorytm klasyfikacyjny działa w następujący sposób. Najpierw generowany jest zbiór reguł klasyfikacyjnych $ClRules$ poprzez wykonywanie procedury $learn(u_i, d_i)$ dla kolejnych przykładów z próbki s. W przypadku podstawowego algorytmu inkrementacyjnego jest to zbiór $ClRules = \bigcup_{d \in V_d} Rules(d)$, natomiast w przypadku algorymu opartego na ograniczeniach zbiór $ClRules = \bigcup_{d \in V_d} Rules(d) \cup nonCCand(d)$. Następnie dla każdego obiektu $u \in U_n$ do klasyfikacji wywoływana jest procedura klasyfikacyjna *classify*.

Procedura działa w następujący sposób. Spośród reguł pokrywających obiekt u wyszukuje reguły $Best$, które mają maksymalną trafność $confidence$, a spośród tych reguły o maksymalnym pokryciu $coverage$. Następnie wylicza, na ktorą decyjzę głosuje więcej reguł. Jeśli jest tylko jedna decyzja z maksymalną liczbą reguł, to procedura zwraca ją jako wynik, w przypadku większej liczby takich decyzji procedura powtarza wyszukiwanie kolejnych najlepszych reguł, ale juz tylko dla najlepszych decyzji z poprzedniego kroku, które zapamiętuje w zbiorze $CandDec$.

ALGORYTM 7.1. classify(u)

$CandDec := V_d$;

powtarzaj, dopóki $\|CandDec\| > 1$ i $ClRules \neq \emptyset$

$Best$:=zbiór tych reguł $\alpha \Rightarrow d$, dla których zachodzą 4 poniższe warunki:

a) $d \in CandDec$

b) α jest spełniane przez u

c) $confidence(\alpha \Rightarrow d) = \max_{r \in ClRules} confidence(r)$

d) $coverage(\alpha \Rightarrow d) = \max_{r \in ClRules: confidence(r) = \max_{r' \in ClRules} confidence(r')} coverage(r)$

$CandDec := \{d' \in CandDec : \|\{\alpha \Rightarrow d' \in Best\}\| = \max_{d \in V_d} \|\{\alpha \Rightarrow d \in Best\}\|\}$

$ClRules := ClRules \setminus Best$

zwróć dowolną decyzję ze zbioru $CandDec$

Jak wskazuje opis algorytm klasyfikacyjny rozstrzyga konflikty między najlepszymi regułami poprzez przeglądanie kolejnych według jakości reguł klasyfikacyjnych. W ten sposób przy dużym zbiorze reguł klasyfikacyjnych sytuacja, w której decyzja podejmowana jest niedeterministycznie, jest bardzo mało proawdopodobna.

Opisana procedura uczenia i klasyfikacji została zastosowana do 5 zbiorów danych pochodzących z repozytorium UCI (cztery pierwsze zostały wykorzystane również w projekcie Esprit Project 5170 StatoLog) dostępnych w Internecie pod adresem http://kdd.ics.uci.edu/. Wartości atrybutów dla wszystkich przykładów we wszystkich tych zbiorach są zawsze określone. Poniżej podany jest opis tych zbiorów, przy czym w statystykach dotyczących liczby i jakości innych metod zostały wzięte pod uwage wszystkie metody, które wykonały się poprawnie i osiągnęły wyniki lepsze niż prosta metoda klasyfikująca zawsze z tą decyzją, która wystąpiła najczęściej w zbiorze treningowym.

1. **DNA**
   - Liczba atrybutów: 60 (60 nominalnych)
   - Liczba obiektów treningowych: 2000
   - Liczba obiektów testowych: 1186
   - Liczba decyzji: 3
   - Rozkład decyzji w zbiorze testowym:
     (50.84%, 25.55%, 23.62%)

- Inne metody:
  - liczba: 21 (StatLog)
  - najmniejszy błąd: 4.1%
  - średni błąd: 9.73%

2. **Shuttle**
   - Liczba atrybutów: 9 (9 numerycznych)
   - Liczba obiektów treningowych: 43500
   - Liczba obiektów testowych: 14500
   - Liczba decyzji: 7
   - Rozkład decyzji w zbiorze testowym:
     $(79.16\%, 14.86\%, 5.58\%, 0.27\%, 0.09\%, 0.03\%, 0.01\%)$
   - Inne metody
     - liczba: 21 (StatLog)
     - najmniejeszy błąd: 0.01%
     - średni błąd: 1.4%
3. **SAT**
   - Liczba atrybutów: 36 (36 numerycznych)
   - Liczba obiektów treningowych: 4435
   - Liczba obiektów testowych: 2000
   - Liczba decyzji: 6
   - Rozkład decyzji w zbiorze testowym:
     $(23.50\%, 23.05\%, 19.85\%, 11.85\%, 11.20\%, 10.55\%)$
   - Inne metody
     - liczba: 22 (StatLog)
     - najmniejszy błąd: 9.4%
     - średni błąd: 15.25%
4. **Letter**
   - Liczba atrybutów: 16 (16 numerycznych)
   - Liczba obiektów treningowych: 15000
   - Liczba obiektów testowych: 5000
   - Liczba decyzji: 26

- Rozkład decyzji w zbiorze testowym:
  brak danych
- Inne metody
  - liczba: 21 (StatLog)
  - najmniejszy błąd: 6.4%
  - średni błąd: 22.09%

5. **Income**
   - Liczba atrybutów: 13 (6 numerycznych, 7 nominalnych)
   - Liczba obiektów treningowych: 30162
   - Liczba obiektów testowych: 15060
   - Liczba decyzji: 2
   - Rozkład decyzji w zbiorze testowym:
     $(75.22\%, 24.78\%)$
   - Inne metody
     - liczba: 16 (MLC++)
     - najmniejszy błąd: 14.05%
     - średni błąd: 16.24%

Eksperymenty dla poszczególnych zbiorów danych wykonane zostały w następujący sposób. Na wstępie dla każdego zbioru danych na podstawie zbioru treningowego wykonywana była dyskretyzacja poszczególnych atrybutów do maksymalnie 8 atrybutów binarnych, a następnie spośród nich wybierane były najlepsze 32 atrybuty. Realizacja obu tych etapów przebiegała według metody opisanej w rozdziale 3.

Pojedynczy test wykonywany by w ten sposób, że w pierwszym kroku algorytm uczy się na podstawie ustalonej część zbioru treningowego i w każdym następnym kroku algorytm dobiera 2 razy większą niż poprzednio liczbę obiektów, na podstawie których się douczał, przy czym krok ten był powtarzany do momentu wyczerpania się zbioru treningowego.

Testy miały dwa parametry: wielkość początkowej części zbioru treningowego oraz wartość minimalnego pokrycia reguły *coverage* jako monotonicznego ograniczenia na przestrzeń reguł. Dla każdego z 5 opisanych zbiorów danych testy mogły być wykonywane dla dowolnej kombinacji wielkości początkowego zbioru: 1/16 (5 faz uczenia), 1/8

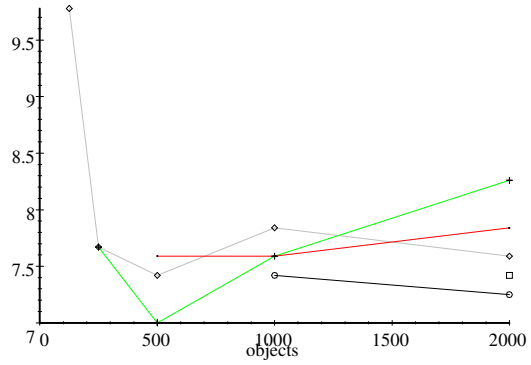
(4 fazy), 1/4 (3 fazy), 1/2 (2 fazy), 1/1 (1 faza) i wartości minimalnego pokrycia: 16%, 8%, 4%, 2%, 1%, 0.5%, 0.25%, 0%.

Zestawienia na wszystkich wykresach odpowiadają liczbie faz uczenia w następujący sposób (kolejne ułamki odpowiadają rozmiarom fragmentów zbioru treningowego wyuczanych w kolejnych fazach):

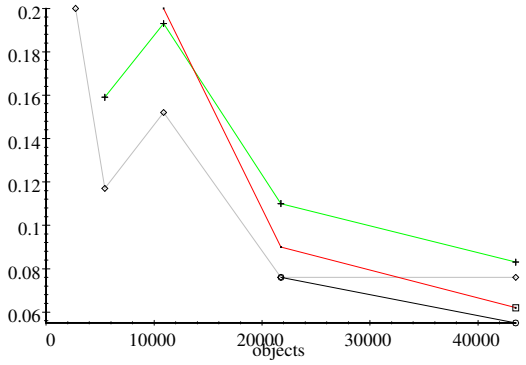
- romb (najjaśniejsza linia) - 5 faz uczenia: 1/16, 1/16, 1/8, 1/4, 1/2
- krzyżyk (jasna linia) - 4 fazy uczenia: 1/8, 1/8, 1/4, 1/2
- punkt (ciemna linia) - 3 fazy uczenia: 1/4, 1/4, 1/2
- kółeczko (najciemniejsza linia) - 2 fazy uczenia: 1/2, 1/2
- kwadracik - 1 faza uczenia: 1/1

W pierwszej części rezultatów przedstawimy porównanie skuteczności i wydajności uczenia inkrementacyjnego z uczeniem nieinkrementacyjnym. W tym celu wykonane zostało 25 testów — po 5 dla każdego z 5 zbiorów danych. Każdy test polegał na wykonaniu podstawowego algorytmu inkrementacyjnego na zbiorze treningowym, przy czym w kolejnych testach dla pojedynczego zbioru liczba faz zmieniała się od 5 do 1, czyli ostatni test był nieinkrementacyjny. W każdym teście mierzone były błąd klasyfikacji, czas liczenia reguł oraz ilość reguł.

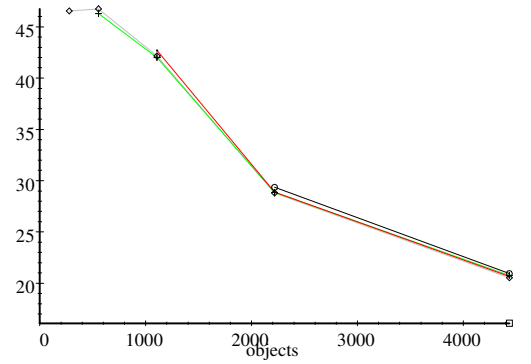
Pierwsze 5 wykresów przedstawia błąd klasyfikacji dla poszczególnych zbiorów danych. Analizując tą grupę wykresów można dokonać następującej obserwacji: dla danych SAT najlepszy jest algorytm nieinkrementacyjny, ale już dla DNA i Shuttle lepszy jest algorytm 2-fazowy, dla Letter 2- i 3-fazowy, a dla Income wszystkie wielofazowe algorytmy są lepsze, przy czym w przypadku zbioru Income jakość klasyfikacji poprawia się wraz ze wzrostem liczby faz. Wniosek, który się nasuwa z tej obserwacji, jest taki, że wraz ze wzrostem rozmiaru danych rośnie liczba faz, która jest optymalna do uzyskania najlepszego wyniku. Druga ważna obserwacja jest następująca: dla niektórych danych (DNA, Shuttle, Income) poziom błędu porównywalny z końcowym został osiągnięty po wyuczeniu małego fragmentu danych. Z reguły nie jesteśmy w stanie określić, jaka ilość danych dla danego zadania jest potrzebna do uzyskania ustalonego poziomu jakości, a uczenie inkrementacyjne pozwala osiągnąć ten poziom możliwie najwcześniej bez posiadania tej informacji.



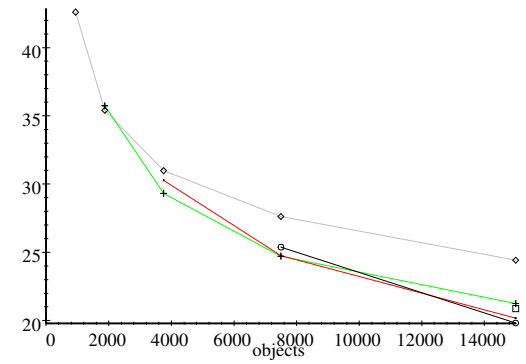
DNA, błąd klasyfikacji (%)



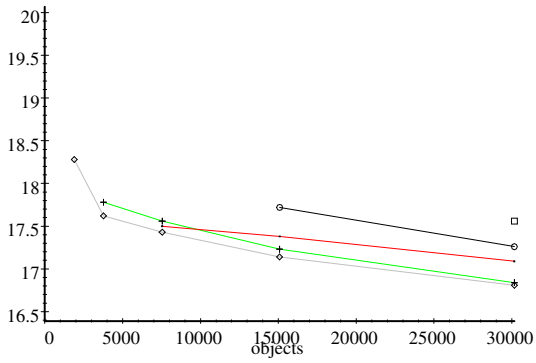
Shuttle, błąd klasyfikacji (%)



SAT, błąd klasyfikacji (%)



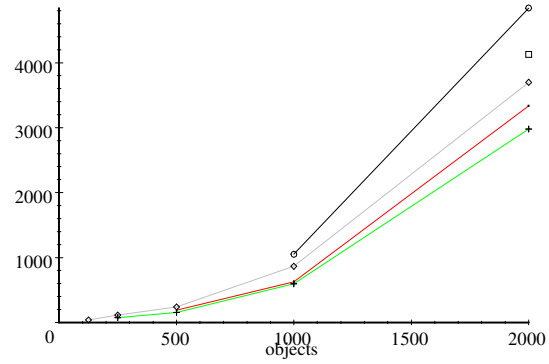
Letter, błąd klasyfikacji (%)



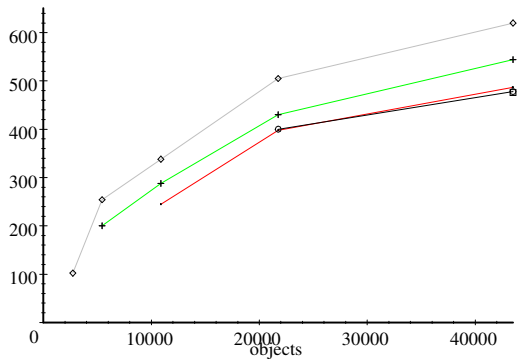
Income, błąd klasyfikacji (%)

Następne 5 wykresów przedstawia ilość reguł wygenerowanych po kolejnych fazach działania algorytmu inkrementacyjnego. Relacja, jaką można zaobserowować pomiędzy liczbą reguł i liczbą obiektów treningowych, jest przeważnie zbliżona do liniowej, choć zdarzają się widoczne odchylenia w kierunku wklęsłości (DNA) lub wypukłości (Shuttle). Natomiast różnica pomiędzy liczbą reguł w wykonaniu inkrementacyjnym i w

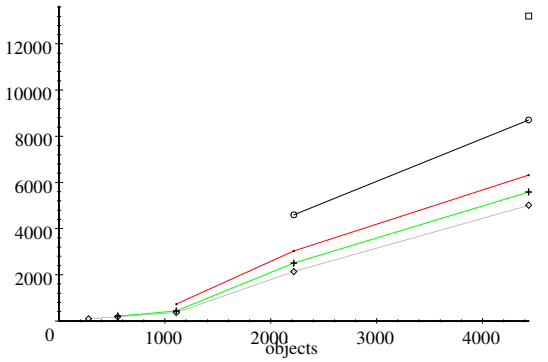
wykonaniu nieinkrementacyjnym jest zależna od typu danych i waha sie od podobnej dla obu algorytmów do 6 razy mniejszej w algorytmie inkrementacyjnym.



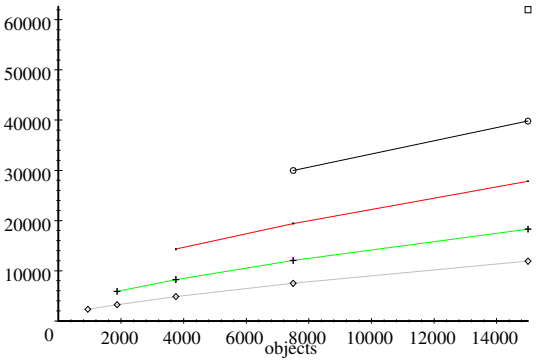
DNA, liczba reguł



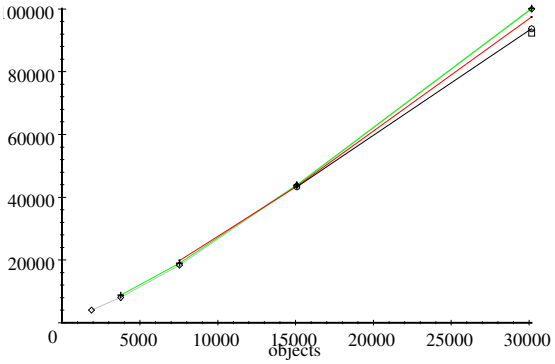
Shuttle, liczba reguł



SAT, liczba reguł



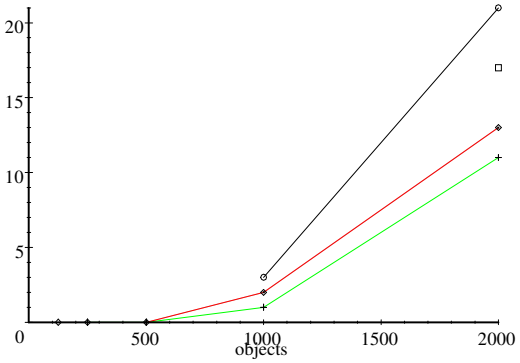
Letter, liczba reguł



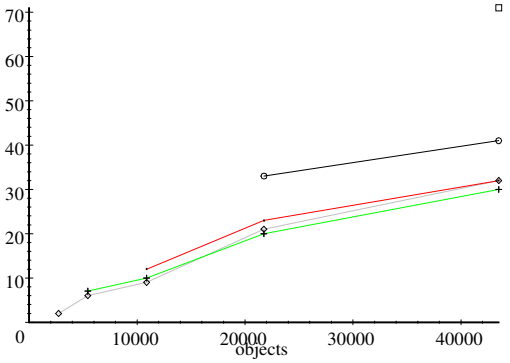
Income, liczba reguł

Ostatnie 5 wykresów w tej części przedstawia czas działania algorytmu na poszczególnych zbiorach danych. Jak pokazują wykresy, czas działania algorytmu jest silnie skolerowany z generowaną przez niego liczbą reguł. Można jednak zauważyć jedną

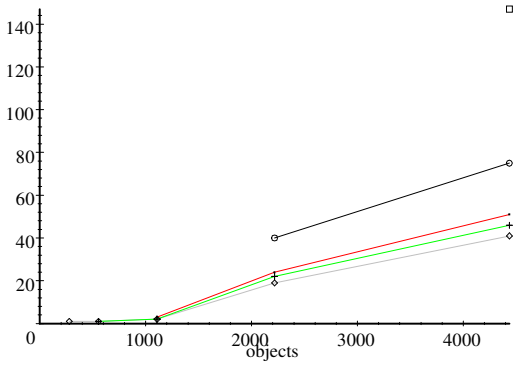
różnicę pomiędzy wykresami liczby reguł i czasu działania. Czas działania algorytmu wielofazowego jewst prawie zawsze (jeden wyjątek: 2-fazowe wykonanie dla DNA) istotnie krótszy w porównaniu do algorytmu 1-fazowego nawet wtedy, kiedy liczba wygenerowanych reguł jest podobna. Wynika to z faktu, że we wczesnym stadium wykonania algorytmu wielofazowego parametry generowanych reguł są szybciej liczone, ponieważ zbiór obiektów treningowych jest mniejszy. W ten sposób algorytm wielofazowy jest w stanie szybciej uzyskać ten sam zbiór reguł klasyfikacyjnych co algorytm 1-fazowy. Analizując czasu działania w zależności od liczby faz można jeszcze zaobserwować, że zwiększanie liczby faz algorytmu przynosi istotną redukcją czasu działania tylko przy małej liczbie faz i wraz ze wzrostem ich liczby wprowadzenie kolejnej fazy daje coraz mniejsze przyspieszenie.



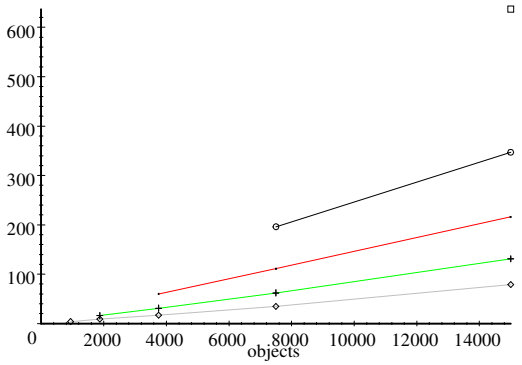
DNA, czas (sek)



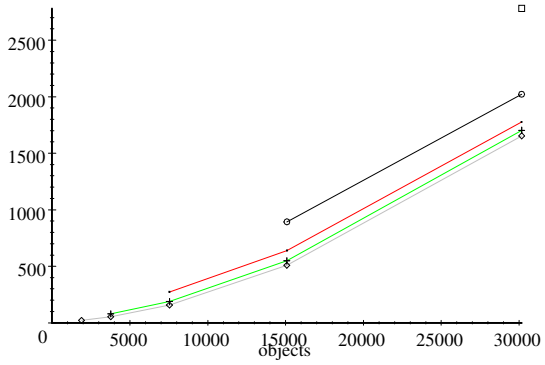
Shuttle, czas (sek)



SAT, czas (sek)



Letter, czas (sek)



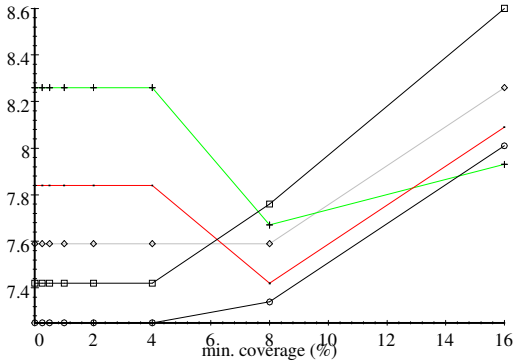
Income, czas (sek)

Łącząc obserwacje z przedstawionych wykresów możemy zauważyć, że dzielenie algorytmu na fazy jest zazwyczaj do pewnego momentu opłacalne, w tym sensie, że nie pogarsza isotnie jakości, natomiast daje dużą szansę na redukcję czasu i pamięci. Wniosek, który się stąd nasuwa, jest taki, że algorytm inkrementacyjny jest nie tylko rozwiązaniem, które stosuje się do dynamicznych zbiorów danych, ale również może stanowić alternatywę dla algorytmu nieinkrementacyjnego w sytuacjach, gdzie ten drugi nie daje się zastosować z powodu nieakceptowalnego kosztu wykonania.

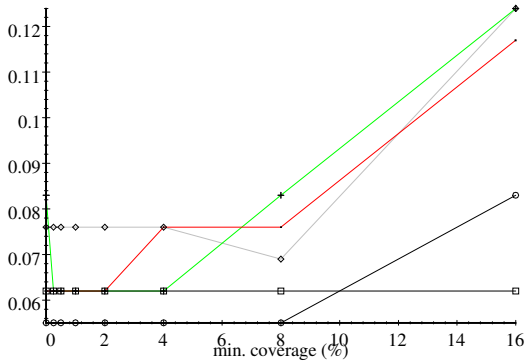
Druga część wyników przedstawia porównanie skuteczności i wydajności algorytmu inkrementacyjnego przy zastosowaniu różnych ograniczeń monotonicznych do reguł klasyfikacyjnych. W poprzednim eksperymencie testy były wykonywane dla każdej kombinacji zbioru danych i liczby faz pomiędzy 1 i 5. Teraz dla każdej takiej ustalonej kombinacji wykonane zostało 8 testów z następującymi ograniczeniami na zbiór reguł: $coverage > 0\%$, $coverage > 0.25\%$, $coverage > 0.5\%$, $coverage > 1\%$, $coverage > 2\%$, $coverage > 4\%$, $coverage > 8\%$ i $coverage > 16\%$. Podobnie jak poprzednio, w każdym teście mierzone były błąd klasyfikacji, czas liczenia reguł oraz ilość reguł. Wyniki na wykresach przedstawiają końcowe wartości parametrów mierzone po zakończeniu wszystkich faz.

Z pierwszej grupy wykresów przedstawiającej błąd klasyfikacji możemy zauważyć, że dla każdego zbioru danych istnieje pewien zakres wartości progu pokrycia reguł $coverage$, dla którego jakość klasyfikacja nie odbiega istotnie od jakości klasyfikacji bazującej na zbiorze wszystkich reguł ($coverage > 0\%$). Dla prostszych danych (DNA, Shuttle) jest to zakres od 0 do 8 %, dla pozostałych $0 - 1$ %. Przy dużych zbiorach danych (Letter,

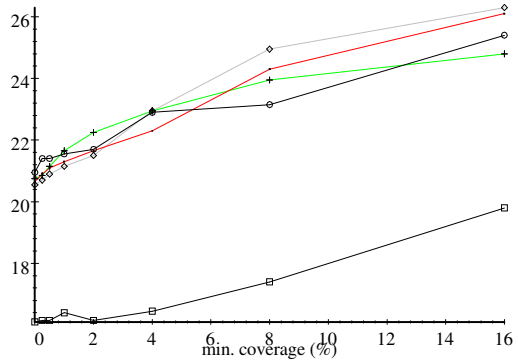
Income) zdarza się, że wyniki klasyfikacji dla reguł spełniających nieduże ograniczenie są lepsze od wyników klasyfikacji dla wszystkich reguł.



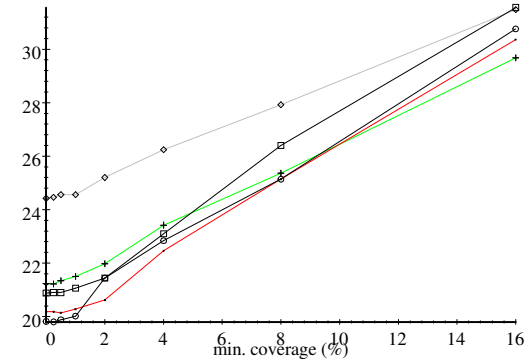
DNA, błąd klasyfikacji (%)



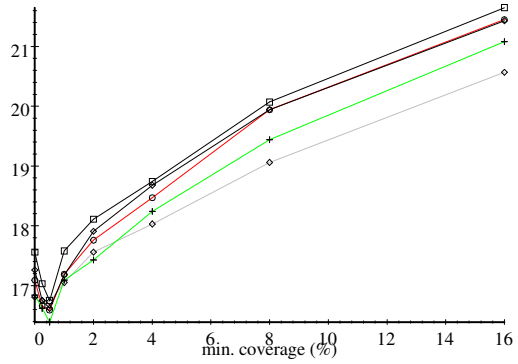
Shuttle, błąd klasyfikacji (%)



SAT, błąd klasyfikacji (%)



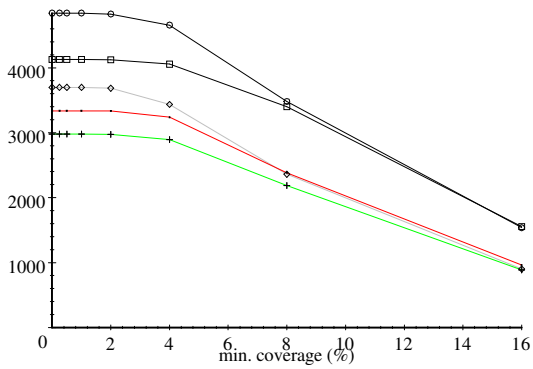
Letter, błąd klasyfikacji (%)



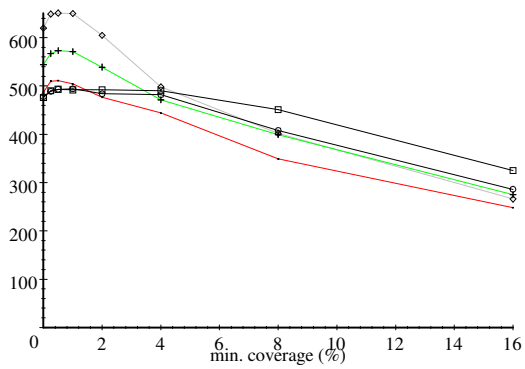
Income, błąd klasyfikacji (%)

Zestawienia przedstawiające liczbę wygenerowanych reguł wskazują, że w zakresie tych progów wartości pokrycia, dla których jakość klasyfikacji nie jest gorsza niż dla wszystkich reguł, liczba reguł pozostaje przeważnie na zbliżonym poziomie. Wyjątkiem jest

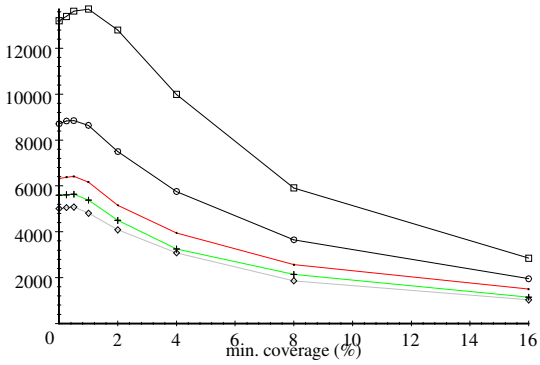
tutaj zbiór Income, dla którego liczba reguł wygenerowanych dla progu $coverage > 1\%$ jest 3 razy mniejsza, choć jakość klasyfikacji dla tego progu jest ciągle nie gorsza niz dla $coverage > 0\%$. Wyjaśnienia tego faktu należy doszukiwać się w rozmiarze i trudności danych. Jeżeli w danych występuje dużo wyjątków, algorytm liczący reguły minimalne bez ograniczenia generuje bardzo specyficzne reguły o bardzo małym pokryciu. Zastosowanie odpowiedniego ograniczenia powoduje, że takie reguły są uszczegóławiane tylko do poziomu ustalonego progu pokrycia, a pojawianie się kolejnych wyjątków powoduje jedynie zmianę parametrów takich reguł bez usuwania ich ze zbioru reguł klasyfikacyjnych.



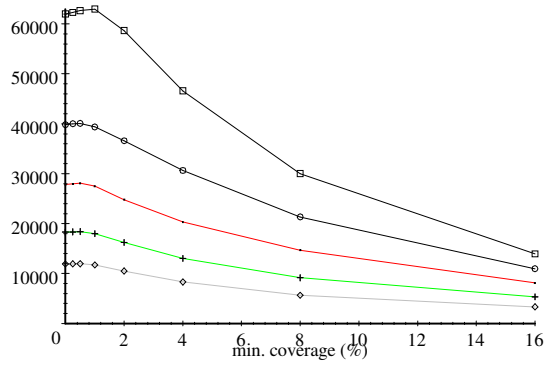
DNA, liczba reguł



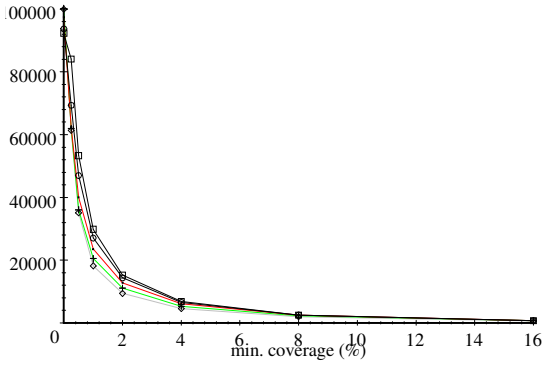
Shuttle, liczba reguł



SAT, liczba reguł

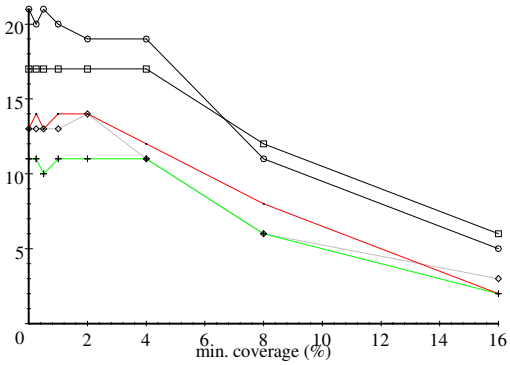


Letter, liczba reguł

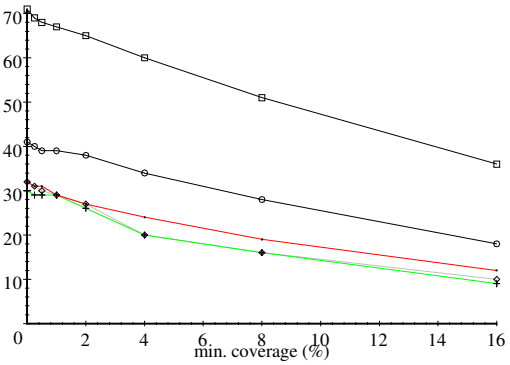


Income, liczba reguł

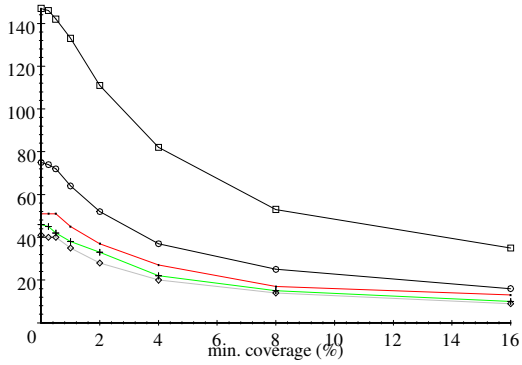
Jak już wcześniej wspomnieliśmy, czas działania algorytmu jest silnie skolerowany z liczba reguł, więc ma podobna charkaterystykę. Nieznaczna różnica daje się zauważyć, kiedy analizujemy wykresy nieco dokładniej: wraz ze wzrostem progu pokrycia reguł *coverage* czas działania maleje nieco szybciej niż liczba reguł. Na szczególną uwagę zasługuje natomiast fakt, że duża redukcja reguł w przypadku zbioru Income odzwierciedla się wyraźnie w redukcji czasu działania algorytmu na tym zbiorze.



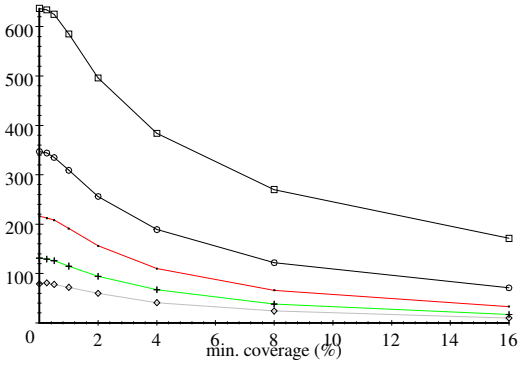
DNA, czas (sek)



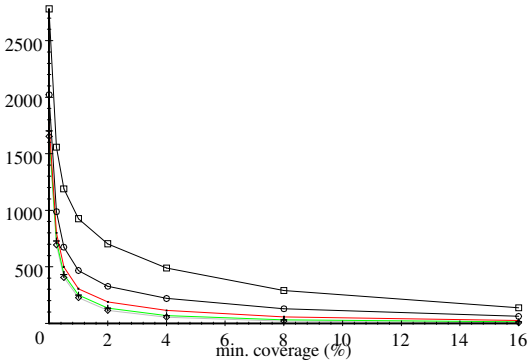
Shuttle, czas (sek)



SAT, czas (sek)



Letter, czas (sek)



Income, czas (sek)

Najtrudniejszym z zaprezentowanych w pracy zbiorów danych jest niewątpliwie zbiór Income, gdyż z jednej strony jest to zbiór danych rzeczywistych, w którym decyzja nie jest bezpośrednio związana z wartościami atrybutów i nie zawsze jest od nich zależna, a z drugiej strony rozmiar tego zbioru jest duży, mniejszy jedynie od zbioru Shuttle. Przyglądając się przedstawionym wynikom możemy zauważyć, że właśnie na tym zbiorze zastosowanie metod opisanych w tej pracy dało najlepszy efekt. Najlepszym przykładem jest tutaj porównanie jakości i czasu działania klasycznego, nieinkrementacyjnego algorytmu liczącego wszystkie reguły minimalne oraz 4-fazowego algorytmu inkrementacyjnego z ograniczeniem $coverage > 1\%$. Czas działania skrócił sie w tym przypadku z około 45 minut do 4 minut i jednocześnie jakość klasyfikacji poprawiła się o około 1.2 %.

Po przedstawieniu wszystkich tych wniosków widać wyraźnie, że przynajmniej w niektórych zastosowaniach, zwłaszcza tam, gdzie mamy do czynienia z dużymi rozmiarami danych, połączenie inkrementacyjnego wykonania i monotonicznych ograniczeń daje

bardzo dobry efekt w kategoriach wydajnościowych. Nalezy jednak pamietać, że sam proces doboru najlepszych parametrów wykonania jest również trudny i moze wymagać wielu eksperymentów.

ROZDZIAŁ 8

PODSUMOWANIE

W niniejszej pracy przedstawiliśmy inkrementacyjną metodę liczenia reguł klasyfikacyjnych, która stosuje się głównie do dynamicznie rozszerzających się zbiorów danych, niemniej można ją stosować także do typowych zbiorów danych, szczególnie w sytuacjach, kiedy ich rozmiar jest duży.

Metoda opiera się na pojęciu obiektu i funkcji boolowskiej i aby można ją było zastosować do dowolnych danych, należy zastosować opisaną w rozdziale 3 binaryzację przestrzeni danych, która dla dowolnego zbioru atrybutów nominalnych i numerycznych definuje przekształcenie przestrzeni nad tymi atrybutami w przestrzeń obiektów boolowskich. Binaryzacja pozwala na konwersję oryginalnego wektora wartości atrybutów opisującego obiekt w wektor wartości binarnych, który stanowi poprawne dane wejściowe dla algorytmu inkrementacyjnego.

Metoda opisana w tej pracy łączy dwa podejścia: opracowaną przez Ziarko i Shana inkrementacyjną metodę liczenia reguł klasyfikacyjnych bazującą na pojęciach z teorii zbiorów przybliżonych oraz zastosowanie monotonicznych ograniczeń umożliwiających redukcję przeszukiwania przestrzeni reguł. Monotoniczne ograniczenia, które zaproponowaliśmy w tej pracy, opierają sie na często wykorzystywanym w zastosowaniach teorii zbiorów przyblizonych parametrze jakościowym reguły, mianowicie na pokryciu klasy decyzyjnej przez regułę. Zastosowanie ograniczenia do reguł spełniających ustalony próg wartości pokrycia pozwala na odrzucenie wielu reguł, które są nieistotne dla jakości końcowego zbioru reguł.

Jak pokazują eksperymenty, połączenie tych podejść nie pogarsza jakości klasyfikacji opartej na wygenerowanym zbiorze danych i jednocześnie daje istotną redukcję czasu i pamięci oraz możliwość ciągłej poprawy zbioru reguł. Znaczenie tej metody jest szczególnie duże, kiedy mamy do czynienia z dużymi i trudnymi zbiorami danych, gdyż dla takich zbiorów orginalny algorytm przestaje być akceptowalny w terminach

wymaganej pamięci oraz czasu wykonania. Mając to na uwadze możemy powiedzieć, że opisany algorytm rozszerza istotnie klasę zbiorów danych, dla których możemy stosować inkrementacyjne metody uczenia bazujące na teorii zbiorów przybliżonych.

Literatura

[1] H. Aizenstein, L. Pitt, *Exact learning of read-k disjoint DNF and not-so-disjoint DNF*, Proceedings of the 5th Annual Workshop on Computational Learning Theory, Pittsburgh, 1992, The ACM Press 1992, 71-76.

[2] D. Angluin, *Inductive inference of formal languages from positive data*, Information and Control, 45, 1980, 117-135.

[3] M. Anthony, N. Biggs, *Computational learning theory*, Cambridge University Press 1992.

[4] J. Bazan, *Metody wnisokwań aproksymacyjnych dla syntezy algorytmów decyzyjnych*, rozprawa doktorska, Uniwersytet Warszawski 1998.

[5] J. Bazan, A. Skowron, P. Synak, *Dynamic reducts as a tool for extracting laws from decision tables*, Proceedings of the Symposium on Methodologies for Intelligent Systems, Charlotte, 1994, Lecture Notes in Artificial Intelligence, 869, Springer-Verlag 1994, 346-355.

[6] U. Berggren, *Linear time deterministic learning of k-term DNF*, Proceedings of the 6th International Conference on Computational Learning Theory, Santa Cruz, 1993, The ACM Press 1993, 37-40.

[7] A. Blum, M. Singh, *Learning functions of k terms*, Proceedings of the 3rd Annual Workshop on Computational Learning Theory, Rochester, 1990, Morgan Kaufmann Publishers 1990, 314-326.

[8] A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth, *Classifing learnable geometric concepts with the Vapnik-Chervonenkis dimension*, Proceedings of the 18th Annual ACM Symposium on the Theory of Computing, The Association for Computing Machinery, New York, 1986.

[9] A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth, *Learnability and the Vapnik-Chervonenkis dimension*, Journal of the ACM, 36(4), 1989, 929-965.

[10] R. Bryant, *Symbolic boolean manipulation with ordered binary decision diagrams*, ACM Computing Surveys 24, 1992, 293-318.

[11] T. Cormen, C. Leiserson, R. Rivest, *Introduction to algorithms*, The MIT Press 1990.

[12] A. Czerwonenkis, V. Vapnik, *On the uniform convergance of relative frequencies of events to their probabilities*, Theory of Probability and its Applications, 16(2), 1971, 264-280.

[13] A. Ehrenfeucht, D. Haussler, M. Kearns, L. Valiant, *A general lower bound on the number of examples needed for learning*, Information and Computation 82, 1989, 247-261.

[14] A. Feigelson, L. Hellerstein, *Learning conjunctions of two unate DNF formulas: computational and informational results*, Proceedings of the 9th International Conference on Computational Learning Theory, Desenzano del Garda, Italy, 1996, The ACM Press 1996, 255-265.

[15] M. Flammini, A. Marchetti-Spaccamela, L. Kućera, *Learning DNF formulae under classes of probability distribution*, Proceedings of the 5th Annual Workshop on Computational Learning Theory, Pittsburgh, 1992, The ACM Press 1992, 85-92.

[16] M. R. Garey, D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman and Company, San Francisco 1979.

[17] M. Gold, *Language identification in the limit*, Information and Control, 10, 1967, 447-474.

[18] G. Grieser, S. Lange, *On the strength of incremental learning*, Proceedings of the 10th International Conference on Algorithmical Learning Theory, Lecture Notes in Artificial Intelligence 1720, Springer-Verlag 1999, 118-131.

[19] T. Hancock, *Learning monotone* 2 μ*-DNF formulas and* k μ *decision tress*, Proceedings of the 4th Annual Workshop on Computational Learning Theory, Santa Cruz, 1991, Morgan Kaufmann Publishers 1991, 199-209.

[20] T. Hancock, Y. Mansour, *Learning monotone* k μ*-DNF formulas on product distributions*, Proceedings of the 4th Annual Workshop on Computational Learning Theory, Santa Cruz, 1991, Morgan Kaufmann Publishers 1991, 179-183.

[21] D. Haussler, N. Littlestone, M. Warmuth, *Predicting* $\{0,1\}$*-functions on randomly drawn points*, Proceedings of the 29th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Washington DC, 1988.

[22] H. Kautz, D. McAllester and B. Selman, *Ten challenges in propositional reasoning and search*, Proceedings of the 15th International Joint Conference on Artificial Intelligence, Nagoya, Aichi, Japan, 1997.

[23] M. Kearns, *The computational complexity of machine learning*, The MIT Press 1990.

[24] J. de Kleer, *An improved incremental algorithm for generating prime implicates*, Proceedings of the 10th National Conference on Artificial Intelligence, San Jose 1992, The AAAI Press/The MIT Press 1992, 780-785.

[25] E. Kushilevitz, *A simple algorithm for learning* $O(\log n)$*-term DNF*, Proceedings of the 9th International Conference on Computational Learning Theory, Desenzano del Garda, Italy, 1996, The ACM Press 1996, 266-269.

[26] S. Lange, T. Zeugmann, *Incremental learning from positive data*, Journal of Computer and System Sciences, 53, 1996, 88-103.

[27] P. Langley, *Machine learning*, The MIT Press 1996.

[28] Y. Mansour, *An* $O\left(n^{\log\log n}\right)$ *learning algorithm for DNF under the uniform distribution*, Proceedings of the 5th Annual Workshop on Computational Learning Theory, Pittsburgh, 1992, The ACM Press 1992, 53-61.

[29] A. Maruoka, Y. Sakai, *Learning monotone log-term DNF formulas*, Proceedings of the 7th International Conference on Computational Learning Theory, New Brunswick, 1994, The ACM Press 1994, 165-172.

[30] Ch. Meinel, T. Theobald, *Ordered binary decision diagrams and their significance in computer-aided design of VLSI circuits*, Electronic Colloquium on Computational Complexity, 5 (039), 1998.

[31] A. Narayan, *Recent advances in BDD based representations for boolean functions: a survey*, Proceedings ot the 12th International Conference on VLSI Design, Goa, India, 1999.

[32] Son H. Nguyen, *Discretization of real value attributes. boolean reasoning approach*, rozprawa doktorska, Uniwersytet Warszawski 1997.

[33] Son. H. Nguyen, A. Skowron, *Quantization of real value attributes*, Proceedings of the 2nd Annual Join Conference on Information Sciences, Wrightsville Beach, NC, 1995, 34-37.

[34] M. Orłowska, M. Orłowski, *Maintenance of knowledge in dynamic information systems*, R. Słowiński (redaktor), Intelligent decision support - handbook of applications and advances of the rough sets theory, Kluwer Academic Publishers 1992, 315-330.

[35] Z. Pawlak, *Rough sets - theoretical aspects of reasoning about data*, Kluwer Academic Publishers, Dordrecht, 1991.

[36] C. Rauszer, A. Skowron, *The discernibility matrices and functions in information systems*, R. Słowiński (redaktor), Intelligent decision support - handbook of applications and advances of the rough sets theory, Kluwer Academic Publishers 1992, 331–362.

[37] L. Polkowski, A. Skowron (redaktorzy), *Rough sets in knowledge discovery 1 - methodology and applications*, Physica-Verlag, Heidelberg, 1998.

[38] L. Polkowski, A. Skowron (redaktorzy), *Rough sets in knowledge discovery 2 - applications, case studies and software systems*, Physica-Verlag, Heidelberg, 1998.

[39] L. Polkowski, A. Skowron, *Towards adaptive calculus of granules*, J. Kacprzyk, L. A. Zadeh (redaktorzy), Computing with Words in Information/Intelligent Systems 1 - Foundations, Springer-Verlag Group (Physica-Verlag), Studies in Fuzziness and Soft Computing, 30, 1999, 201-228.

[40] N. Shan, W. Ziarko, *An incremental learning algorithm for constructing decision rules*, W. Ziarko (redaktor), Rough sets, fuzzy sets and knowledge discovery, Workshop in Computing, Springer-Verlag & British Computer Society, London, Berlin, 1994, 326-334.

[41] D. Sieling, I. Wegener, *Graph driven BDDs — a new data structure for boolean functions*, Theory of Computer Science 141 (1,2), 1995, 283-310.

[42] A. Skowron, *Boolean reasoning for decision rules generation*, J. Komorowski, Z. W. Ras (redaktorzy), Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems, Trondheim, Norway, 1993, Lecture Notes in Computer Science, 689 (1993), 295-305.

[43] R. Susmaga, *Experiments in incremental computation of reducts*, L. Polkowski, A. Skowron (redaktorzy), Rough sets in knowledge discovery 1 - methodology and applications, Physica-Verlag, Heidelberg, 1998, 530-553.

[44] S. Tsumoto, *Extraction of medical diagnostic knowledge based on rough set model*, Proceedings of the 7th International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems, Paris, 1998, 1546-1551.

[45] L. Valiant, *A theory of the learnable*, Communications of the ACM, 27(11), 1984, 1134-1142.

[46] V. Vapnik, *The nature of statistical learning theory*, Springer Verlag, New York, 1995.

[47] R. Wiehagen, *Limes-Erkennung rekursiver Funktionen durch spezielle Strategien*, Journal of Information Processing and Cybernetics, 12, 1976, 93-99.

[48] J. Wróblewski, *Finding minimal reducts using genetic algorithms*, Proceedings of the 2nd Annual Join Conference on Information Sciences, Wrightsville Beach, NC, 1995, 186-189.

[49] J. Wróblewski, *Covering with reducts - a fast algorithm for rule generation*, Proceedings of the 1st International Conference on Rough Sets and Current Trends in Computing, Warsaw, Poland, 1998, Lecture Notes in Artificial Intelligence, 1424, Springer Verlag, Berlin, Heidelberg, 1998, 402-407.