ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2002
Informatyka – Zeszyt 1

Jarosław Stepaniuk[1], Leszek Góralczuk

ALGORTYM GENEROWANIA REGUŁ PIERWSZEGO RZĘDU WYKORZYSTUJĄCY METODY ZBIORÓW PRZYBLIŻONYCH

Streszczenie: W pracy przedstawiono algorytm generowania reguł pierwszego rzędu, tzn. zależności, które w poprzedniku mają koniunkcję formuł atomowych bądź ich negacji a w następniku formułę atomową. Technikę zbiorów przybliżonych wykorzystano w procesie doboru literałów mogących wchodzić w skład przesłanki generowanej reguły. Kryterium doboru opiera się na tym, aby reguła po dołączeniu do jej przesłanki kandydującego literału jak najlepiej rozróżniała przykłady pozytywne i negatywne, które do tej pory nie były rozróżnialne.

Słowa kluczowe: zbiory przybliżone, indukcyjne programowanie logiczne

1. Wstęp

Zastosowanie Zbiorów Przybliżonych (RS) [9], [14], [5], [6] w Indukcyjnym Programowaniu Logicznym (ILP) ma na celu usprawnienie procesu uczenia się z danych niepełnych. W artykule przedstawiono algorytm, który na podstawie indukcyjnego programowania logicznego i zbiorów przybliżonych umożliwia odkrywanie reguł pierwszego rzędu. Prezentowany algorytm bazuje na systemie otoczeń, przy czym na wejściu otrzymuje system decyzyjny pierwszego rzędu, a na wyjściu produkuje zbiór reguł pierwszego rzędu [8]. Idea wykorzystania zbiorów przybliżonych do znajdowania reguł bazuje na schemacie sekwencyjnego pokrywania. Główną operacją schematu sekwencyjnego pokrywania jest utworzenie pewnej koniunkcji literałów, która posłuży jako część warunkowa reguły – przesłanka [13]. Skutkiem dołączania kolejnych literałów do przesłanki jest zwiększenie liczby poprawnie pokrytych przykładów pozytywnych i zmniejszanie liczby pokrytych przez regułę przykładów negatywnych. Tym, co odróżnia przedstawiany

[1] Wydział Informatyki, Politechnika Białostocka, ul. Wiejska 45A, 15-351 Białystok

algorytm od innych, jest kryterium wyboru dołączanych literałów do przesłanki [3]. Kryterium opiera się na tym, aby reguła po dołączeniu literału jak najlepiej rozróżniała przykłady negatywne i pozytywne. Dopiero spośród takich literałów-kandydatów wybieramy tego, który po dołączeniu do części warunkowej reguły sprawia, że reguła jak najlepiej pokrywa zbiór przykładów. Aby ocenić rozróżnialność przykładów, stosuje się technikę zbiorów przybliżonych.

W ostatniej części pracy przedstawiono wyniki eksperymentów przeprowadzonych przy wykorzystaniu systemów ILP i opisywanego algorytmu. Eksperymenty przeprowadzono na zbiorze danych opisujących położenie bloków tekstu w dokumentach.

2. Indukcyjne programowanie w logice i zbiory przybliżone

2.1. Wybrane pojęcia indukcyjnego programowania logicznego

Indukcyjne programowanie w logice (ILP) jest obszarem badań znajdującym się na pograniczu dwóch dziedzin: uczenia się maszyn *Machine Learning* (ML) i programowania logicznego *Logic Programming* (LP) [13], [3]. Indukcyjne logiczne programowanie wykorzystuje do reprezentacji wiedzy język rachunku predykatów. Umożliwia to przedstawianie nie tylko klasyfikacji pojedynczych obiektów dziedziny do pewnych kategorii, lecz także związków zachodzących między różnymi obiektami. System ILP może wykorzystywać wiedzę dziedzinową bardziej naturalnie i efektywnie, ponieważ w ILP zarówno przykłady, wiedza dziedzinowa jak i wygenerowane reguły są wyrażane w tym samym języku.

Głównym celem systemu ILP jest znajdowanie reguł dla relacji celu. Relacje te reprezentowane są przez predykaty docelowe. Wnioskowanie odbywa się na podstawie wiedzy trenującej, przedstawionej jako przykłady oraz na podstawie znanych już zależności, które zachodzą pomiędzy relacjami – czyli wiedzy dziedzinowej.

W systemach ILP istnieje problem przy przetwarzaniu niedokładnych danych wejściowych. Uzyskany wynik może okazać się niewłaściwy, jeśli dane wejściowe są niedokładne, a w konsekwencji tej niedokładności indukowane hipotezy wyjściowe H mogą być błędne. Standardowe algorytmy ILP nie radzą sobie dobrze z takiego typu problemami. Nie sprawdzają się one dla danych niedokładnych, niekompletnych, zbyt ogólnikowych i nieścisłych, występujących zarówno w informacji trenującej, wiedzy dziedzinowej, jak też i wygenerowanych regułach.

Na niedokładne dane wejściowe składają się:

1. Zakłócone dane (ang. *noisy data*) np. błędne wartości argumentów w przykładach, błędna klasyfikacja przykładów.

2. Za rzadkie dane np. podane obserwacje zawarte w przykładach uczących są niewystarczające do otrzymania właściwej hipotezy H.
3. Brakujące dane, np. niektóre argumenty w niektórych przykładach uczących mają nieznane wartości.
4. Dane sprzeczne, np. niektóre spośród przykładów należą zarówno do E^+ jak i E^-.

2.2. Wybrane pojęcia teorii zbiorów przybliżonych

Zbiory przybliżone to jedna z technik służących do identyfikacji i rozpoznawania wzorców w danych. Jest ona szczególnie użyteczna w przypadku danych niekompletnych, opiera się na aproksymacji zbiorów. Obiekt zostaje przydzielony do konkretnej klasy na podstawie relacji nierozróżnialności R, zdefiniowanej na zbiorze U wszystkich obiektów.

Oznaczmy przestrzeń aproksymacji jako AS

$$AS = (U,R) \tag{2.1}$$

gdzie U jest zbiorem obiektów, natomiast R jest relacją nierozróżnialności na zbiorze U. Czyli U jest podzielone przez R na klasy nierozróżnialności [4].

Jeżeli $X \subseteq U$, to obie aproksymacje, dolną i górną, możemy zdefiniować następująco:

$$\underline{Apr_{AS}}(X) = \bigcup_{[x]_R \subseteq X} [x]_R = \{x \in U | [x]_R \subseteq X\} \tag{2.2}$$

$$\overline{Apr_{AS}}(X) = \bigcup_{[x]_R \cap X \neq \emptyset} [x]_R = \{x \in U | [x]_R \cap X \neq \emptyset\} \tag{2.3}$$

Gdzie $[x]_R$ oznacza klasę równoważności, elementu x.

3. Algorytm konstrukcji reguł

3.1. Pojęcia wstępne

W dalszych rozważaniach zakładamy, że generowane reguły nie są regułami rekurencyjnymi, tzn. żaden predykat nie może występować jednocześnie w części warunkowej i decyzyjnej reguły.

W celu sprecyzowania, które predykaty mają się znajdować w części warunkowej reguły, a które w części decyzyjnej, będziemy używać, podobnie jak w PROGOLU [12], poleceń *modeh* i *modeb*:

modeh – definiuje predykaty wchodzące w skład konkluzji reguł.
modeb – definiuje predykaty mogące wchodzić w skład przesłanek reguł.

Definicja relacji mogącej wchodzić w skład przesłanki reguły ma postać:

$$modeb(r, p(t_1,...,t_n)),$$

gdzie
$r \geq 1$ jest maksymalną liczbą wystąpień predykatu w przesłance,
p jest nazwą n-elementowej relacji,
t_i jest i-tym argumentem relacji.

Dla każdego argumentu t_i można określić czy zmienna odpowiadająca temu argumentowi jest wejściowa, czy też wyjściowa. Wejściowe argumenty poprzedzane są znakiem '+', wyjściowe znakiem '–'. Analogicznie wygląda definicja *modeh*.

Przykład. $modeb(1, ojciec(+osoba, -osoba))$ definiuje predykat o nazwie ojciec, który może maksymalnie raz wchodzić w skład przesłanki reguły. Predykat ojciec jest dwuargumentowy, oba argumenty są argumentami typu osoba, z czego pierwszy jest argumentem wejściowym, natomiast drugi jest argumentem wyjściowym.

Jeżeli dany jest literał a, to przez $In(a)$ oznaczamy zbiór zmiennych wejściowych literału a, natomiast przez $Out(a)$ zbiór zmiennych wyjściowych literału a.

Zbiór zmiennych wejściowych koniunkcji literałów jest definiowany jako suma zbiorów zmiennych wejściowych poszczególnych literałów.

3.2 System decyzyjny pierwszego rzędu

System decyzyjny pierwszego rzędu definiowany jest jako czwórka $\langle B, E, M_h, M_b \rangle$, gdzie B jest zbiorem reguł wyróżnionych jako wiedza dziedzinowa, E jest zbiorem przykładów $E = E_h \cup E_b$, gdzie E_h oznacza przykłady odpowiadające predykatom zdefiniowanym w M_h, E_b oznacza przykłady odpowiadające predykatom zdefiniowanym w M_b, M_h jest zbiorem deklaracji wskazujących predykaty mogące wchodzić w skład konkluzji reguły (*modeh*), M_b jest zbiorem deklaracji określających predykaty mogące wchodzić w skład przesłanki reguły (*modeb*) [3].

Zarówno przykłady odpowiadające predykatom zdefiniowanym w M_h, jak i przykłady odpowiadające predykatom zdefiniowanym w M_b, można podzielić na pozytywne i negatywne: $E_h = E_h^+ \cup E_h^-$ i $E_b = E_b^+ \cup E_b^-$.

Przykładowa definicja predykatu wchodzącego w skład M_h:

$modeh(1, dziadek(+osoba, +osoba))$

Przykładowa definicja predykatu wchodzącego w skład M_b:

$modeb(1, ojciec(+osoba, -osoba))$

W M_h może znajdować się wiele deklaracji predykatów docelowych, w $E_h = E_h^+ \cup E_h^-$ mogą znajdować się przykłady dla wszystkich predykatów zadeklarowanych w M_h. Oznaczmy więc symbolem $E_h^+(p)$ zbiór przykładów należących do E_h^+, w których występuje predykat p i analogicznie przez $E_h^-(p)$ zbiór przykładów należących do E_h^-, w których występuje predykat p.

Głębokość Niech h będzie konkluzją, a b przesłanką reguły. Głębokość zmiennej X w regule IF b THEN h, którą będziemy również zapisywać jako $h \leftarrow b$, definiujemy następująco:

1. Jeżeli $X \in In(h)$, to Głębokość$(h \leftarrow b, X) = 0$
2. Jeżeli $X \in Out(b)$, to Głębokość$(h \leftarrow b, X) = \min$ (Głębokość$(h \leftarrow c, U)) + 1$ dla $c \subset b$ i $U \in In(b)$
3. ∞ w przeciwnym przypadku.

Głębokość reguły jest równa maksymalnej głębokości zmiennych w niej występujących.

Przykład. Zbadajmy głębokość reguły $z(A) \leftarrow q(A,B) \wedge q(B,C)$ przy następujących deklaracjach predykatów $modeh(1, z(+typ))$, $modeb(2, q(+typ, -typ))$. Głębokość zmiennej A wynosi 0, ponieważ $A \in In(h)$. Ponieważ zmienna B jest zmienną wyjściową wprowadzaną przez literał $q(A,B)$, głębokość B jest równa głębokości A plus jeden, czyli 1. Głębokość zmiennej C wynosi głębokość B plus jeden, czyli 2.

Generowanie kandydatów

Zdefiniujmy funkcję $Generuj\_Kandydatów(h,b,k)$, gdzie h jest konkluzją reguły, b jest przesłanką reguły oraz k jest maksymalną głębokością zmiennych. Niech q będzie literałem kandydującym do dodania do przesłanki b takim, że [3]:

$q = p(s_1,...,s_n)$ jeżeli $modeb(r, p(t_1,...,t_n))$ jest w M_b i dla $1 \le i \le n$

a) $s_i = X$ jeżeli $t_i = +t$, $X \in Out(b) \cup In(h)$ i $X \notin Out(neg(b))$

b) $s_i = X$ jeżeli $t_i = -t$, i X jest nową zmienną, która nie występuje ani w h ani w b oraz spełniony jest warunek Głębokość$(h \leftarrow b \wedge q, X) \le k$.

p pojawia się co najwyżej $r-1$ razy w b, $neg(b)$ oznacza zbiór literałów negatywnych wchodzących w skład przesłanki.

Do przesłanki można dodać kandydata, którego deklaracja jest zamieszczona w M_b, oraz spełnione są warunki:

a) Argument wejściowy kandydata musi być zmienną, która występuje jako argument wyjściowy w literałach już zawartych w przesłance, lub zmienną, która występuje jako argument wejściowy literału docelowego. Zmienne reprezentujące argumenty wyjściowe przesłanki opisanej przez negatywny literał nie mogą być używane jako zmienne wejściowe w argumentach innych literałów wchodzących do przesłanki reguły – dlatego $X \notin Out(neg(b))$.

b) Argument wyjściowy musi być nową zmienną, która nie występuje jako argument wejściowy predykatu docelowego, ani nie występuje jako argument wyjściowy w przesłance. Należy sprawdzić czy po dodaniu do przesłanki kandydata reguła będzie miała odpowiednią głębokość. $Głębokość(h \leftarrow b \wedge q, X) \leq k$.

Literał p, na podstawie którego został stworzony kandydat q może pojawiać się co najwyżej $r-1$ razy w przesłance b.

Przykład. Chcemy znaleźć opis relacji *dziadek* – będzie to nasz predykat docelowy. Opis tworzymy na podstawie relacji *ojciec* i *rodzic*. Deklaracje predykatów:

M_h: $modeh(1, dziadek(+osoba, +osoba))$

M_b: $modeb(1, ojciec(+osoba, -osoba))$, $modeb(1, ojciec(+osoba, +osoba))$
$modeb(1, matka(+osoba, -osoba))$, $modeb(1, matka(+osoba, +osoba))$
$modeb(1, rodzic(+osoba, +osoba))$

Załóżmy, że mamy do dyspozycji poniższą wiedzę dziedzinową i dane treningowe:

$$B = \begin{cases} rodzic(X,Y) \leftarrow ojciec(X,Y) \\ rodzic(X,Y) \leftarrow matka(X,Y) \end{cases} \quad E_b^+ = \begin{cases} ojciec(Adam, Tadeusz) \\ ojciec(Adam, Eliza) \\ ojciec(Tadeusz, Katarzyna) \\ matka(Anna, Tadeusz) \\ matka(Anna, Eliza) \\ matka(Katarzyna, Donald) \\ matka(Katarzyna, Zuzanna) \end{cases}$$

$$E_h^+ = \begin{cases} dziadek(Adam, Katarzyna) & (e_1) \\ dziadek(Tadeusz, Donald) & (e_2) \end{cases}$$

$$E_h^- = \begin{cases} \neg dziadek(Anna, Katarzyna) & (e_3) \\ \neg dziadek(Anna, Eliza) & (e_4) \\ \neg dziadek(Adam, Tadeusz) & (e_5) \end{cases}$$

Zakładamy, że w literale docelowym $dziadek(A,B)$ występują zmienne A i B.
Niech przesłanka $b = \{\neg matka(A,C)\}$.
Generujemy zbiór kandydatów do dołączenia do przesłanki:
$ojciec(A,D)$, $ojciec(B,D)$, $ojciec(A,A)$, $ojciec(A,B)$, $ojciec(B,A)$, $ojciec(B,B)$,
$matka(A,A)$, $matka(A,B)$, $matka(B,A)$, $matka(B,B)$,
$rodzic(A,A)$, $rodzic(A,B)$, $rodzic(B,A)$, $rodzic(B,B)$.

Powyżej nie ma kandydatów dla predykatu $matka(+osoba,-osoba)$, ponieważ predykat matka znajduje się już w przesłance, a w deklaracji $modeb(1, matka(+osoba,-osoba))$ maksymalna liczba wystąpień tego predykatu w przesłance została ograniczona do jednego wystąpienia.
Wśród kandydatów zabrakło również:
$ojciec(C,D)$, $ojciec(A,C)$, $ojciec(C,C)$, $ojciec(C,A)$, $ojciec(C,B)$, $ojciec(C,C)$,
$matka(A,C)$, $matka(C,C)$, $matka(C,A)$, $matka(C,B)$, $matka(C,C)$,
$rodzic(A,C)$, $rodzic(C,C)$, $rodzic(C,A)$, $rodzic(C,B)$, $rodzic(C,C)$,
ponieważ zmienna C została wprowadzona poprzez literał negatywny $\neg matka(A,C)$, wobec tego nie może być użyta jako argument wejściowy w żadnym z kandydatów.

3.3. System otoczeń

Pokrycie Niech b będzie przesłanką i h niech będzie literałem docelowym. Pokrycie reguły $h \leftarrow b$ oznaczamy jako $Pokrycie(h \leftarrow b)$.

Reguła $h \leftarrow b$ pokrywa przykład $h(A_1, A_2, ..., A_n)$, jeżeli istnieje podstawienie nadające wartości wszystkim zmiennym występującym w regule, dla którego spełnione są wszystkie literały zawarte w przesłance b. Oznaczamy przez $Pokrycie^+(h \leftarrow b)$ zbiór przykładów pozytywnych pokrytych przez regułę $h \leftarrow b$ i przez $Pokrycie^-(h \leftarrow b)$ zbiór przykładów negatywnych pokrytych przez regułę $h \leftarrow b$. Definiujemy miarę trafności reguły $h \leftarrow b$ jako

$$Oceń\_Pokrycie(h \leftarrow b) = \frac{|Pokrycie^+(h \leftarrow b)|}{|Pokrycie^-(h \leftarrow b)| + 1}. \qquad (3.1)$$

Załóżmy, że $\{H1, H2, ..., Hn\}$ jest zbiorem reguł i $e \in E_h$ jest przykładem.

System otoczeń $NS_{\{H1,H2,...,Hn\}}(e)$ jest sumą pokryć wszystkich reguł, które pokrywają element e tzn.

$$NS_{\{H1,H2,...,Hn\}}(e) = \bigcup_{i=1}^{n}\{Pokrycie(Hi) \mid e \in Pokrycie(Hi)\} \tag{3.2}$$

Dolna i górna aproksymacja zbioru przykładów X dla zbioru reguł $\{H1,H2,.....,Hn\}$:

$$\underline{Apr_{\{H1,H2,...,Hn\}}}(X) = \{e \in E_h \mid NS_{\{H1,H2,...,Hn\}}(e) \subseteq X\} \tag{3.3}$$

$$\overline{Apr_{\{H1,H2,...,Hn\}}}(X) = \{e \in E_h \mid NS_{\{H1,H2,...,Hn\}}(e) \cap X \neq \emptyset\} \tag{3.4}$$

Przykład. Załóżmy, że dla literału docelowego $h = dziadek(A,B)$ mamy przesłankę $b = \{ojciec(A,C)\}$. Rozpatrujemy dodanie następujących literałów $q = ojciec(C,B)$ i $\neg q = \neg ojciec(C,B)$. Wyznaczmy pokrycia dla przesłanki $b \wedge q = \{ojciec(A,C) \wedge ojciec(C,B)\}$
oraz przesłanki $b \wedge \neg q = \{ojciec(A,C) \wedge \neg ojciec(C,B)\}$

$$NS_{\{h\leftarrow b\wedge q,\, h\leftarrow b\wedge\neg q\}}(e_1) = Pokrycie(h \leftarrow b \wedge q) = \{e_1\},$$
$$NS_{\{h\leftarrow b\wedge q,\, h\leftarrow b\wedge\neg q\}}(e_2) = Pokrycie(h \leftarrow b \wedge \neg q) = \{e_2, e_5\},$$
$$NS_{\{h\leftarrow b\wedge q,\, h\leftarrow b\wedge\neg q\}}(e_3) = \emptyset,\ NS_{\{h\leftarrow b\wedge q,\, h\leftarrow b\wedge\neg q\}}(e_4) = \emptyset,$$
$$NS_{\{h\leftarrow b\wedge q,\, h\leftarrow b\wedge\neg q\}}(e_5) = Pokrycie(h \leftarrow b \wedge \neg q) = \{e_2, e_5\},$$
$$\underline{Apr_{\{h\leftarrow b\wedge q,\, h\leftarrow b\wedge\neg q\}}}(E_h^{+}) = \{e_1\} \text{ i } \overline{Apr_{\{h\leftarrow b\wedge q,\, h\leftarrow b\wedge\neg q\}}}(E_h^{+}) = \{e_1, e_2, e_5\}.$$

3.4. Algorytm znajdujący zbiór reguł

Algorytmy sekwencyjnego pokrywania [13] polegają na takim dodawaniu kolejnych literałów do przesłanki reguły, aby uzyskać jak najlepsze pokrycie przykładów. W kolejnych iteracjach do przesłanki dołączamy taki literał, który rozróżnia jak najwięcej spośród dotychczas nierozróżnialnych przykładów.

Niech $Dis\{H1,H2,...,Hn\}$ oznacza zbiór par przykładów rozróżnialnych przez zbiór reguł $\{H1,H2,.....,Hn\}$. Określmy kiedy przykład e_i jest odróżnialny od przykładu e_j. Po pierwsze – przykłady muszą należeć do różnych klas decyzyjnych. Ponieważ mamy dwie klasy decyzyjne – przykłady pozytywne i przykłady negatywne – przyjmujemy, że $e_i \in E_h^{+}$, natomiast $e_j \in E_h^{-}$.

Przykład e_i jest odróżnialny od przykładu e_j za pomocą zbioru reguł $\{H1,H2,...,Hn\}$ wtedy i tylko wtedy, gdy $NS_{\{H_1,H_2,...,H_n\}}(e_i) \neq \emptyset$ lub $NS_{\{H_1,H_2,...,H_n\}}(e_j) \neq \emptyset$ i $NS_{\{H_1,H_2,...,H_n\}}(e_i) \cap NS_{\{H_1,H_2,...,H_n\}}(e_j) = \emptyset$.

Algorytm generujący pojedynczą regułę [3] opiera się na sekwencyjnym dołączaniu literałów do przesłanki b. W kolejnych iteracjach dołączamy do b tego kandydata, który rozróżnia jak najwięcej spośród dotychczas nierozróżnialnych przykładów. Ponieważ reguła może być nieskończona, określamy jej maksymalną głębokość jako $k \in <0,\infty)$.

Zakładamy, że na początku wszystkie przykłady są nierozróżnialne i algorytm sukcesywnie powiększa zbiór przykładów rozróżnionych przez rozszerzanie przesłanki reguły. Algorytm kończy iteracje, gdy wszystkie przykłady zostaną rozróżnione $(R = \emptyset)$ lub R nie może być dalej zredukowany. W drugim przypadku przykłady nie mogą już zostać bardziej rozróżnione; przyjmujemy wtedy, że przykłady negatywne, które są pokryte przez wygenerowaną regułę są wyjątkami. Należy zaznaczyć, że nie dla wszystkich zbiorów redukcja ta jest dobrą metodą, ponieważ mogą istnieć inne przesłanki, które lepiej rozróżniają przykłady.

Główna idea procedury generującej zbiór reguł.

Wejście: h – literał docelowy.
$< B, E, M_h, M_b >$ – system decyzyjny pierwszego rzędu.
k – maksymalna głębokość generowanej reguły.
apr – aproksymacja $apr \in \{dolna, górna\}$
Wyjście: SH – zbiór reguł.

function *Generuj_Zbiór_Reguł*$(h,< B,E,M_h,M_b >,k,apr)$
begin
1. $E_{curr} = E_h^+(h)$, $SH = \emptyset$
2. **while** $E_{curr} \subseteq E$ **do**
3. H = *Generuj_Regułę*$(h,< B,E,M_h,M_b >,k,apr)$
4. $E = E - Pokrycie^+(H)$
5. $SH = SH \cup \{H\}$

done
end.

Komentarze do opisu algorytmu *Generuj_Zbiór_Reguł*:
Na wejściu podajemy $< B,E,M_h,M_b >$ – system decyzyjny pierwszego rzędu, literał docelowy h, którego definicja musi znajdować się w M_h, k – maksymalną

głębokość generowanej reguły oraz *apr* – rodzaj aproksymacji. Na wyjściu algorytm zwraca regułę H.

1. E_{curr} to wszystkie przykłady należące do E_h^+, które są obserwacjami dla predykatu h.
2. Powtarzamy pętlę dopóki SH nie pokrywa wszystkich przykładów pozytywnych.
3. Generujemy regułę H.
4. Z przykładów E usuwamy przykłady pozytywne pokryte przez znalezioną regułę H.
5. Dodajemy znalezioną regułę do zbioru reguł.

Teraz przedstawimy główną ideę procedury *Generuj_Regułę*.

function *Generuj_Regułę*$(h,< B,E,M_h,M_b >,k,apr)$
begin
1. $R=\{< e_i,e_j > \mid \forall e_i \in E_h^+(h), \forall e_j \in E_h^-(h)\}$ $b=\varnothing$
2. **while** $R \neq \varnothing$ **do**
   3. $\forall q \in Generuj\_Kandydatów(h,b,k)$
      oblicz $R(q)=R \cap Dis\{h \leftarrow b \wedge q, h \leftarrow b \wedge \neg q\}$
   4. $p=q$ dla którego $|R(q)|$ jest maksymalne
   5. **if** $R(p)=\varnothing$ **then**
      $\forall e \in Pokrycie^-(h \leftarrow b)$
      $b=b \wedge \neg e$
      exit while
   6. **if** $Oceń\_Pokrycie(h \leftarrow b \wedge p) > Oceń\_Pokrycie(h \leftarrow b \wedge \neg p)$
      then $bp=p$
      else $bp=\neg p$
   7. **if** apr = *górna* **and** $Pokrycie^+(h \leftarrow b) < Pokrycie^+(h \leftarrow b \wedge bp)$ **then**
      exit while;
   8. $b=b \wedge bp$ $R=R-R(p)$
done
9. **return** $h \leftarrow b$
end.

Komentarze do opisu procedury *Generuj_Regułę*:

1. Określamy R jako zbiór par przykładów, które są rozróżnialne. R jest zbiorem par przykładów relacji h, które nie należą do tej samej klasy decyzyjnej.

Ponieważ występują dwie klasy decyzyjne – przykłady pozytywne $E_h^+(h)$ i przykłady negatywne $E_h^-(h)$ – więc jest to zbiór par, w których pierwszy element jest przykładem pozytywnym, drugi jest przykładem negatywnym.

2. Powtarzamy pętlę dopóki zbiór par przykładów R jest niepusty.
3. Dla wszystkich kandydatów q otrzymanych za pomocą metody *Generuj_Kandydatów*(h,b,k) obliczamy $R(q)$ – zbiór par należących do R, które rozróżnia przesłanka $b \wedge q$ lub $b \wedge \neg q$.
4. Spośród zbioru kandydatów wybieramy takiego (oznaczmy go przez p), dla którego liczność zbioru $R(p)$ jest największa.
5. Jeżeli $R(p) = \varnothing$, to kończymy iterację. Zbiór par przykładów nie może zostać bardziej zredukowany, przyjmujemy wtedy, że przykłady negatywne, które są pokryte przez wygenerowaną regułę, są wyjątkami i dodajemy je do przesłanki reguły.
6. Określamy czy ze znakiem negacji czy bez tego znaku ma być dodany kandydat p do przesłanki b. Jeżeli reguła $h \leftarrow b$ po dodaniu do przesłanki b kandydata pozytywnego p ma lepsze pokrycie niż po dodaniu do przesłanki kandydata negatywnego $\neg p$, to $bp = p$ w przeciwnym przypadku $bp = \neg p$.
7. Jeżeli obliczamy górną aproksymację i po dodaniu do przesłanki b kandydata bp zmniejszyła się liczba pokrywanych przykładów pozytywnych, kończymy iterację.
8. Do przesłanki dodajemy kandydata $b = b \wedge bp$. Ze zbioru par R usuwamy te pary, które rozróżnia przesłanka po dodaniu do niej wybranego kandydata.
9. Zwracamy regułę H.

Przykład. W celu zilustrowania działania metody *Generuj_Regułę* posłużmy się przykładem przytoczonym w paragrafie 3.2. $k = 1$, $apr = dolna$, $h = dziadek(A,B)$.

Kolejne kroki działania algorytmu:

1. $R = \{< e_1,e_3 >,< e_1,e_4 >,< e_1,e_5 >,< e_2,e_3 >,< e_2,e_4 >,< e_2,e_5 >\}$.
2. $R \neq \varnothing$.
3. kandydaci $q = \{ ojciec(A,C),\ ojciec(B,C),$
   $ojciec(A,A),\ ojciec(A,B),\ ojciec(B,A),\ ojciec(B,B),$
   $matka(A,C),\ matka(B,C),$
   $matka(A,A),\ matka(A,B),\ matka(B,A),\ matka(B,B),$
   $rodzic(A,A),\ rodzic(A,B),\ rodzic(B,A),\ rodzic(B,B)\ \}$.
   Dla każdego kandydata obliczmy $R(q) = R \cap Dis(h \leftarrow \{b \wedge q\}, h \leftarrow \{b \wedge \neg q\})$.

4. Na podstawie wyników poprzedniego punktu wybieramy kandydata p o największej liczności $R(p)$. Najlepszy rezultat otrzymaliśmy dla 7-ego kandydata $p = matka(A,C)$.
   $R(p) = R \cap Dis(dziadek(A,B) \leftarrow matka(A,C), dziadek(A,B) \leftarrow \neg matka(A,C)) =$
   $\{< e_1, e_3 >, < e_1, e_4 >, < e_2, e_3 >, < e_2, e_4 >\}$
   $|R(p)| = 4$.
5. $R(p) \neq \emptyset$.
6. $Oceń\_Pokrycie(dziadek(A,B) \leftarrow matka(A,C)) <$
   $Oceń\_Pokrycie(dziadek(A,B) \leftarrow \neg matka(A,C))$
   wobec tego $bp = \neg matka(A,C)$.
7. $apr \neq górna$.
8. $b = \neg matka(A,C)$. $R = R - R(p) = \{< e_1, e_5 >, < e_2, e_5 >\}$.
2. $R \neq \emptyset$. Powtarzamy obliczenia – drugi przebieg pętli.
3. kandydaci $q = \{ ojciec(A,D), ojciec(B,D),$
   $ojciec(A,A), ojciec(A,B), ojciec(B,A), ojciec(B,B),$
   $matka(A,A), matka(A,B), matka(B,A), matka(B,B),$
   $rodzic(A,A), rodzic(A,B), rodzic(B,A), rodzic(B,B) \}$.
   Dla każdego kandydata obliczmy $R(q) = R \cap Dis(h \leftarrow \{b \wedge q\}, h \leftarrow \{b \wedge \neg q\})$.
4. Wybieramy kandydata p o największej liczności $R(p)$ – $p = ojciec(A,B)$
   $R(ojciec(A,B)) = \{< e_1, e_5 >, < e_2, e_5 >\}$
   $|R(p)| = 2$.
5. $R(p) \neq \emptyset$.
6. $Oceń\_Pokrycie(dziadek(A,B) \leftarrow \{\neg matka(A,C) \wedge ojciec(A,B)\}) <$
   $Oceń\_Pokrycie(dziadek(A,B) \leftarrow \{\neg matka(A,C) \wedge \neg ojciec(A,B)\})$
   wobec tego $bp = \neg ojciec(A,B)$.
7. $apr \neq górna$.
8. $b = \neg matka(A,C) \wedge \neg ojciec(A,B)$. $R = R - R(p) = \emptyset$.
3. $R = \emptyset$. Kończymy działanie procedury *Generuj_Regułę*.

4. Badania eksperymentalne

Badania eksperymentalne przeprowadzono na bazie danych opisujących jednostronicowe dokumenty. Każdy dokument [2] określony jest przez dwa odmienne typy struktur reprezentujące zarówno jego zawartość jak i jego organizację – czyli

strukturę geometryczną i logiczną. Strukturę geometryczną charakteryzują własności takie jak: tekst, linie (pionowe czy poziome), elementy graficzne itd. [1]. Drugą strukturę charakteryzuje zawartość dokumentu taka jak: tytuł, paragraf, sekcja, rozdział, tabela itd. Każdy z logicznych obiektów może, oczywiście, być opisany przez zbiór atrybutów. Atrybutami są własności i relacje zachodzące pomiędzy komponentami. Nas będą interesowały zwłaszcza relacje pomiędzy komponentami, bo za ich pośrednictwem będziemy rozpoznawać bloki dokumentu.

Mamy pięć literałów docelowych. Określają one logiczne komponenty, które mogą być wyspecyfikowane w próbkach listów. W eksperymentach rozpatrujemy 30 pojedynczych stron dokumentów. Wykonano 6 eksperymentów: 20 dokumentów stanowiło podstawę do danych trenujących, pozostałych 10 generowało dane testowe.

Eksperymenty przeprowadzono przy użyciu czterech systemów: FOIL [10], PROGOL [12], LINUS [11] oraz za pomocą prezentowanego algorytmu. W poniższej tabeli zamieszczono wyniki otrzymane dla tych czterech systemów. Kolumna A przedstawia procent przykładów pozytywnych poprawnie zaklasyfikowanych, kolumna B przedstawia procent przykładów negatywnych błędnie zaklasyfikowanych jako pozytywne.

Tabela 1

Wyniki eksperymentów

systemy	FOIL		PROGOL		LINUS		algorytm dolna aprok.		algorytm górna aprok.	
wyniki	A	B	A	B	A	B	A	B	A	B
Eksperyment 1										
logic_type_rif	100	9,09	100	15,91	100	14,77	83,33	9,38	100	11,93
logic_type_sender	100	7,06	100	20,06	100	19,49	80,00	7,06	80	7,06
logic_type_date	92,31	13,39	7,69	1,99	100	25,36	84,62	10,83	84,62	10,83
logic_type_receiver	100	5,65	100	24,86	100	14,12	50,00	5,65	80,00	5,65
logic_type_logo	100	5,65	100	6,21	100	5,65	90,00	5,65	100	5,65
Eksperyment 2										
logic_type_rif	89,47	7,54	100	26,09	73,68	10,43	15,79	7,54	100	10,14
logic_type_sender	91,67	6,82	66,67	7,67	100	19,03	66,67	6,53	91,67	6,82
logic_type_date	100	11,9	100	13,03	100	15,58	90,91	7,37	90,91	12,18
logic_type_receiver	90	5,37	100	12,43	100	12,43	60,00	5,65	100	5,65
logic_type_logo	100	5,65	100	5,65	100	5,65	100	5,65	100	5,65

Eksperyment 3										
logic_type_rif	94,12	8,07	100	13,54	100	26,51	0,00	8,07	100	10,66
logic_type_sender	91,67	6,82	100	19,03	100	27,56	66,67	6,53	91,67	6,82
logic_type_date	100	12,15	100	11,58	100	15,82	100	7,63	100	12,71
logic_type_receiver	80	5,37	100	12,43	100	24,86	0,00	5,65	100	5,65
logic_type_logo	100	5,65	100	5,65	100	5,65	100	5,65	100	5,65
Eksperyment 4										
logic_type_rif	100	10,26	92,31	12,82	100	26,78	0,00	9,12	92,31	9,12
logic_type_sender	100	6,8	90,91	15,58	90,91	24,65	63,64	6,8	90,91	6,8
logic_type_date	92,31	3,42	100	15,95	100	15,10	84,62	6,84	84,62	6,84
logic_type_receiver	90	6,21	100	18,64	90	24,29	100	5,65	100	5,65
logic_type_logo	100	5,65	100	6,21	100	5,65	80	5,65	100	5,65
Eksperyment 5										
logic_type_rif	100	7,98	84,62	13,39	100	27,35	84,62	9,12	100	11,68
logic_type_sender	40	5,73	40	5,73	40	5,73	40	5,73	40	5,73
logic_type_date	84,62	8,55	92,31	10,54	100	15,10	92,31	6,84	100	11,97
logic_type_receiver	80	6,5	100	24,86	90	17,23	50	5,65	100	5,65
logic_type_logo	100	5,65	100	6,21	100	6,78	10	5,65	100	5,65
Eksperyment 6										
logic_type_rif	68,75	8,33	93,75	13,51	100	24,71	6,25	8,33	100	9,77
logic_type_sender	92,31	6,55	84,62	6,84	92,31	24,22	38,46	6,27	92,31	6,55
logic_type_date	84,62	1,71	92,31	10,54	100	15,10	84,62	6,84	100	15,1
logic_type_receiver	90	5,08	100	12,43	100	24,86	70	5,65	100	5,65
logic_type_logo	100	5,65	100	6,21	100	5,65	100	5,65	100	5,65

Należy zaznaczyć, że w systemach FOIL, PROGOL i LINUS nie wykorzystuje się zjawiska dolnej i górnej aproksymacji. Wyniki otrzymane za pomocą prezentowanego algorytmu są zbliżone do wyników otrzymanych za pomocą systemu FOIL, są natomiast lepsze od wyników otrzymanych za pomocą systemów PROGOL i LINUS. Najgorsze wyniki otrzymano dla predykatu logic_type_sender w 5-tym eksperymencie. Najlepsze wyniki dla wszystkich systemów uzyskano dla predykatu logic_type_logo. Najgorsze rezultaty otrzymano dla systemu LINUS.

W systemie FOIL maksymalna głębokość zmiennych w otrzymanych regułach wynosi 2. W systemie FOIL dopuszczalne są reguły rekurencyjne i takie w rozwiązaniu występują. W przesłankach nie używamy literałów negatywnych. W przypadku systemu PROGOL wynik jest o wiele bardziej spójny niż otrzymany przy udziale systemu FOIL. PROGOL może produkować reguły, w skład których

wchodzą jedynie literały pozytywne - w przesłankach nie występują literały negatywne. Dla omawianego algorytmu otrzymano zbiory zawierające od 2 do 11 reguł, maksymalna głębokość zmiennych wynosi 4, w regułach występują literały negatywne. Pewną wadą wyników otrzymanych przy zastosowaniu dolnej aproksymacji jest za duża szczegółowość rozwiązania, uzyskujemy wiele reguł i przesłanki składają się z wielu literałów. Przy zastosowaniu górnej aproksymacji rozwiązaniem jest jedna reguła dla każdego z przeprowadzonych eksperymentów.

Podsumowanie

W przedstawionym algorytmie podstawowym kryterium wyboru literałów, które mają być dodane do przesłanki reguły, jest to, żeby po ich dodaniu przesłanka jak najlepiej rozróżniała przykłady pozytywne od negatywnych. Jest to podejście odmiennie niż reprezentowane w systemach FOIL, PROGOL czy LINUS, gdzie preferuje się literały, po których dodaniu część warunkowa reguły pokrywa jak najwięcej przykładów pozytywnych. Można zauważyć, że wyniki otrzymane przy udziale FOIL'a są nieco lepsze od tych otrzymanych za pomocą PROGOL'a, choć reguły otrzymane za pomocą PROGOL'a posiadają w przesłankach mniejszą liczbę literałów. Dokładność zbioru reguł otrzymanych dla omawianego algorytmu nie odbiega od dokładności zbioru reguł otrzymanych przy udziale FOIL'a i jest lepsza od dokładności reguł otrzymanych przy użyciu PROGOL'a i LINUS'a.

Ciekawym pomysłem, o który można rozszerzyć algorytm może okazać się generowanie reguł na podstawie przykładów negatywnych.

Literatura

[1] F. Esposito, D. Malerba, G.Semerano, M.Pazzani: *A Machine Learning Approach To Document Understanding,* Proceedings of the Second International Workshop on Multistrategy Learning, R. S. Michalski, G. Tecuci, (Eds.), 1993, 276-292.

[2] ftp://ftp.mlnet.org/ml-archive/general/data/doc-understanding/ – baza danych dokumentów.

[3] H. Midelfart, J. Komorowski: *A Rough Set Approach to Inductive Logic Programming,* W. Ziarko, Y. Yao, (Eds.), Proceedings of the 2nd International Conference on Rough Sets and Current Trends in Computing (RSCTC-2000), 2000, 158-166.

[4] J. Stepaniuk: *Knowledge Discovery by Application of Rough Set Models,* L. Polkowski, S. Tsumoto, T.Y. Lin, (Eds.), Rough Sets: New Developments, Physica-Verlag, Heidelberg, 2000, 137-233.
[5] L. Polkowski, A. Skowron (Eds.): *Rough Sets in Knowledge Discovery 1: Methodology and Applications.* Physica-Verlag, Heidelberg, 1998.
[6] L. Polkowski, A. Skowron (Eds.): *Rough Sets in Knowledge Discovery 2: Applications, Case Studies and Software Systems.* Physica-Verlag, Heidelberg, 1998.
[7] N. Lavrac, S. Dzeroski: *Inductive Logic Programming, Techniques and Applications,* Ellis Horwood Limited 1994.
[8] P. Cichosz: *Systemy uczące się.* WNT, 2000.
[9] S.K. Pal, A. Skowron (Eds.): *Rough Fuzzy Hybridization: A New Trend in Decision-Making.* Springer-Verlag, Singapore, 1999.
[10] System FOIL: http://www.cse.unsw.edu.au/~quinlan/.
[11] System LINUS: ftp://ftp.mlnet.org/ml-archive/ILP/public/software/linus/.
[12] System PROGOL: http://www-users.cs.york.ac.uk/~stephen/progol.html.
[13] T. Mitchell: *Machine Learning.* McGraw Hill, 1997.
[14] Z. Pawlak: *Rough Sets. Theoretical Aspects of Reasoning about Data,* Kluwer Academic Publishers, Dordrecht, 1991.

AN ALGORITHM GENERATING FIRST ORDER RULES BASED ON ROUGH SET METHODS

Summary: The aim of this paper is to introduce and investigate an algorithm for finding first order rules. Rough set theory is used in the process of selecting literals, which may be part of the rule. The criterion of selecting literals reads as follows: only those literals are selected, which adding to the rule makes that the rule discerns the most examples from those, which were yet undiscerned.

Key words: rough sets, inductive logic programming