Automatyczne wyodrębnianie reguł

Jedną z form reprezentacji wiedzy jest jej zapis w postaci zestawu reguł. Ta forma ma szereg korzyści: daje się łatwo interpretować, można zrozumieć sposób działania zbudowanego systemu ekspertowego, jak również prześledzić sposób podejmowania decyzji przez system. Te elementy są szczególnie istotne z punktu widzenia szeregu zastosowań. Przykładowo w przemyśle inżynier odpowiadający za realizację procesu technologicznego nie zawierzy automatowi, którego działanie nie rozumie, dlatego też w takich zastosowaniach konieczna jest taka forma reprezentacji wiedzy, która umożliwia zrozumienie podjętej decyzji. Identyczne zachowania obserwuje się również w innych obszarach na styku człowieka z maszyną, jak ekonomii, medycynie, gdzie obok decyzji równie ważne jest prześledzenie sposobu wnioskowania przeprowadzonego przez komputer.

W wielu obszarach nauki wiedza dziedzinowa jest jasna i przejrzysta, wówczas można ją bezpośrednio zaimplementować do systemu ekspertowego w postaci zbioru reguł lub kombinacji reguł i modeli. Jednak w pewnych zagadnieniach taka wiedza jest niedostępna. Przykładowo takie nauki jak psychologia, socjologia, medycyna i ekonomia w głównym stopniu opierają się na analizie zebranych danych doświadczalnych, dla których nie istnieje model fizyczny zachodzących zjawisk. Jednak nawet w dobrze ugruntowanych obszarach nauki, jak inżynieria materiałowa i metalurgia gdzie modele fizyczne są dobrze znane jest wiele przypadków, gdzie właśnie te modele fizyczne nie dostarczają niezbędnych informacji lub dostarczają je jedynie w postaci przybliżonej.

Dlatego dysponując danymi pomiarowymi można postarać się zbudować system, który wydobytą wiedzę zapisuje w postaci reguł. Można tutaj wymienić dwa standardowe podejścia do wyodrębniania reguł:

- Algorytm sekwencyjnego pokrywania
- Algorytm drzew decyzji

Poniżej te dwie rodziny algorytmów zostaną opisane. Na wstępie jednak trzeba wprowadzić pewne założenia. Indukcja reguł odbywa się na podstawie dostarczonego zbioru danych D, który składa się z pary $\{\mathbf{X}, \mathbf{y}\}$ tak, że $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$ jest zbiorem *n* wektorów opisanych w przestrzeni $\aleph$, a $\mathbf{y} = [y_1, y_2, \dots, y_n]$, jest zmienną zależną taką, że każdemu wektorowi $\mathbf{x}_i$ przypisana jest jedna wartość y_i, taka że $y_i \in [c_1, c_2, \dots, c_k]$, czyli $y_i$należy do jednej z *k* kategorii. Na przestrzeń $\aleph$ mogą składać się zarówno zmienne symboliczne, jak i ciągłe. Takie zagadnienie nosi nazwę problemu klasyfikacyjnego.

Każde zagadnienie indukcji wiedzy, w tym również indukcji reguł wiąże się z problemem zapewnienia odpowiedniej generalizacji zbudowanego systemu. Generalizację należy tutaj rozumieć, jako zapewnienie odpowiedniej ogólności uzyskanych reguł. Problem ten wynika z faktu, iż zwykle dane na podstawie, których chcemy dokonać klasyfikacji są zaszumione, i/lub przestrzeń $\aleph$ ma zbyt niski wymiar, co oznacza, że zarejestrowaliśmy niedostateczną liczbę zmiennych. Wówczas algorytm indukcji nie jest w stanie zbudować bezbłędnego zestawu reguł lub też zbudowany system regułowy staje się zbyt rozbudowany, czyli posiada nadmierną liczbę reguł. Duża liczba reguł ogranicza możliwości interpretacji uzyskanych wyników oraz prowadzi do małego wsparcia reguł. Niska wartość współczynnika wsparcia reguły oznacza, że została ona wygenerowana dla małego podzbioru

przypadków, a to może poddawać w wątpliwość poprawność reguły. Dlatego też generując reguły powinno się szukać kompromisu pomiędzy dokładnością a złożonością systemu.

Innym zjawiskiem związanym z generalizacją systemu jest efekt przeuczenia. Jest on bezpośrednio związany z istnieniem szumu w danych. Zjawisko przeuczenia polega na wyodrębnienie zbyt dokładnego zbioru reguł, które zaczynają odwzorowywać szum zawarty w danych. Zjawisko to jest bardzo niebezpieczne, dlatego też generując zestaw reguł należy kontrolować generalizację systemu. Najprostszą metodą oceny generalizacji jest tak zwany test krzyżowy polegający na kilkukrotnym podziale zbioru D na dwie rozłączne części – treningowa i testową, a następnie nauczeniu systemu na części treningowej oraz przetestowaniu na części testowej.

Algorytm sekwencyjnego pokrywania

Algorytm sekwencyjnego [2] pokrywania polega na generowaniu reguł tak, iż w pierwszym kroku generowana jest pierwsza z reguł, która pokrywa maksymalnie dużo przypadków z danej kategorii c_i , a w kolejnym kroku poszukiwana dodawana jest nowa reguła, która pokrywa niepokryte dotychczas przypadki. Proces den odbywa się aż do momentu, gdy wszystkie przypadki są pokryte przez reguły, lub też do momentu, gdy akceptowalna liczba przypadków (z góry założona) jest niepoprawnie klasyfikowana. Schemat algorytmu sekwencyjnego pokrywania przedstawia Rys. 1

```
function sekwencyjnePokrywanie(D)
        D zbiór danych uczących D = [X y]
R = ∅; P = D
while P ≠ ∅
        k = znajdźKompleks(D,P);
        w = kategoria(k,D,P);
        R = R ∪ {k->w};
        P = P – Pk  /Pk – zbiór wektorów pokrytych przez kompleks k
end
return R
```

Rys. 1 Schemat algorytmu sekwencyjnego pokrywania

Gdzie **R**- to zbiór reguł, funkcja *znajdźKompleks()* odpowiada za znajdywanie części warunkowej reguły (kompleksu), natomiast funkcja *kategoria()* odpowiada za przypisanie kategorii (etykiety) dla danego kompleksu, czyli konkluzję reguły.

Cechą charakterystyczną algorytmu sekwencyjnego pokrywania jest to, iż każda kolejna reguła jest coraz bardziej szczegółowa, tzn pokrywa coraz mniej przypadków. Zdarza się, iż ostatnie z reguł pokrywają zaledwie kilka, lub w szczególnych przypadkach jeden wektor, co odpowiada niskiemu współczynnikowi wsparcia tak powstałej reguły.

Istnieje szereg różnych algorytmów implementujących funkcję *znajdźKompleks()* szczegóły można znaleźć w [2]. Składają się one z dwóch koncepcji – algorytmu określającego sposób poszukiwania kompleksu oraz odpowiedniej metody jego oceny (odpowiedniej heurystyki). Do realizacji tego drugiego zagadnienia zwykle używa się różnych współczynników statystycznych takich jak *informacja wzajemna,* czy też *indeks Gini*.

Drzewa decyzji

Drzewa decyzji [1,2] stanowią alternatywę dla algorytmów z rodziny sekwencyjnego pokrywania. Ich działanie polega na stopniowym podziale przestrzeni wejściowej na coraz mniejsza podprzestrzenie.

Korzysta się przy tym z koncepcji *dziel i rządź*. Takie podejście zapewnia niską złożoność obliczeniową algorytmu, co stanowi o dużej popularności drzew decyzji.

Ogólnie koncepcję budowy drzewa można przedstawić jak na schemacie Rys. 2

```
function [t] = drzewo(D)
[n_p, n_n] = size(D)
If (n_p. > th) & (n_n> th)
   [D_L ,D_P,Θ] = Podziel(D);
   t = t ∪ { Θ}
Else
  L =określLiść(D)
   t = t ∪ { L }
  return t
end;

 t_L = drzewo(D_L);
 t = t ∪ { t_L }

 t_P = drzewo(D_P);
 t = t ∪ { t_P }
return t;
```

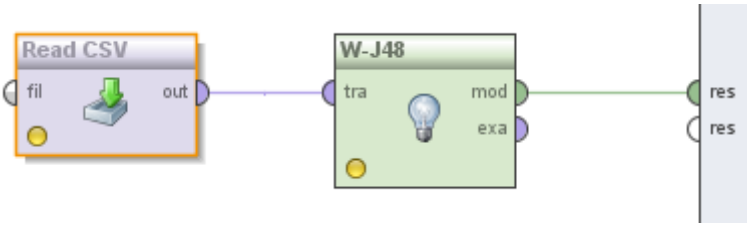
Rys. 2 Schemat budowy drzewa decyzji

Proces budowy drzewa decyzji jest procesem rekurencyjnym. Rozpoczyna się od spełnienia czy dostarczone dane *D* pozwalają na stworzenie kolejnego węzła. Sprawdzenie to odbywa się przez analizę liczby wektorów należących do poszczególnych klas. Jeśli ich liczba jest mniejsza niż określony próg *th* (dla uproszczenia zakładamy problem binarny, w którym występują: klasa pozytywna i negatywna – liczebności poszczególnych klas oznaczono odpowiednio n_p, n_n) wówczas nowy węzeł definiowany jest jako liść i przypisywana jest mu odpowiednia kategoria.. Jeśli natomiast kryteria minimalnej liczebności są spełnione zbiór danych dzielony jest na rozłączne podzbiory za pomocą funkcji *Podziel*. Funkcja ta, w najprostszym przypadku, dokonuje przeszukania przestrzeni cech, analizując każdą zmienną z osobna, i próbując dokonać podziału (para *zmienna + próg*) na rozłączne części. W schemacie z Rys. 2 założono dwie rozłączne części, co powoduje że tworzone jest tak zwane drzewo binarne. W następnym kroku rozpoczyna się rekurencja po wszystkich podziałach zbioru danych

Zadania

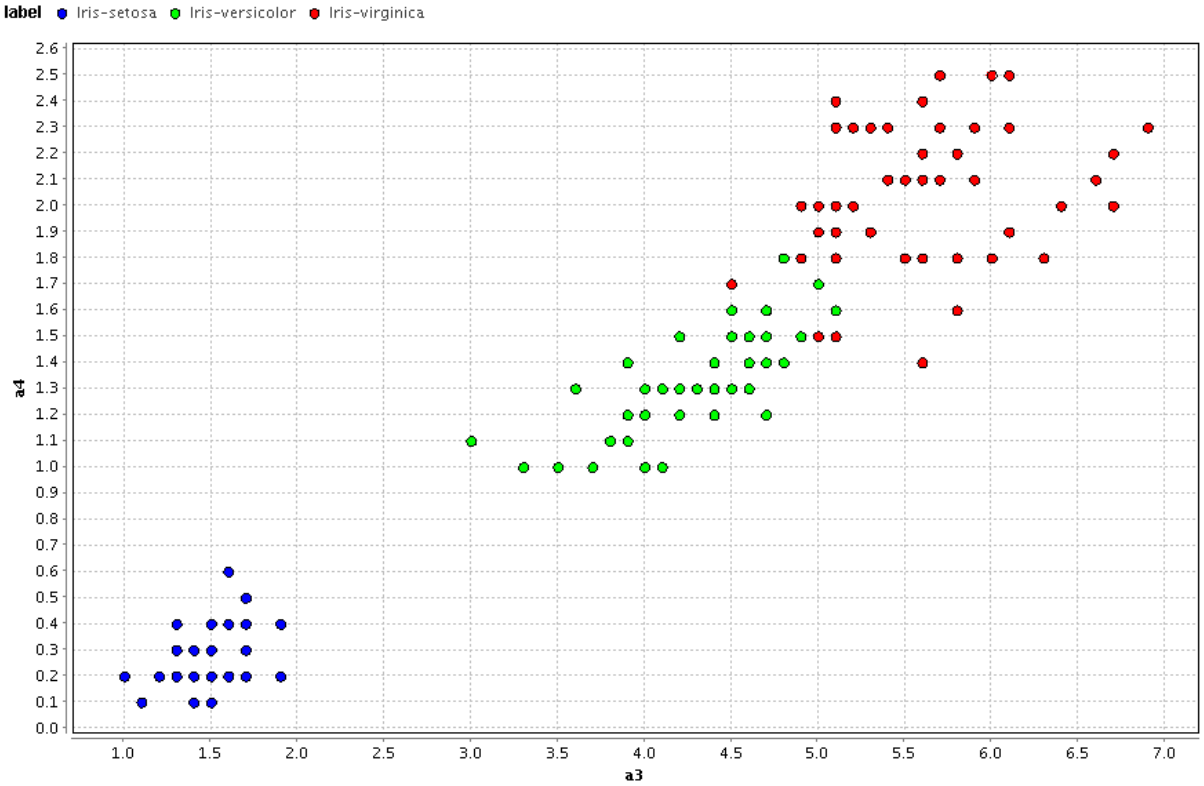
Zad. 1.

Ze strony podanej przez prowadzącego pobierz zbiór danych *test1.csv* a następnie wykorzystując oprogramowanie RapidMiner [5] oraz dodatek Weka zbuduj schemat jak na rysunku (Instrukcja dla programu RapidMiner dostępna jest w [6]):



Rys. 3 Proces testowania zachowania algorytmu drzew decyzji oraz algorytmu sekwencyjnego pokrywania

Operatorem *Read CSV* wczytaj pobrany zbiór danych. Operator W-J48 stanowi implementację drzewa decyzji C4.5. Uruchom proces i zaobserwuj zbudowane drzewo. Wrysuj w poniższy rysunek przedstawiający zbiór danych granice podziału drzewa decyzji.



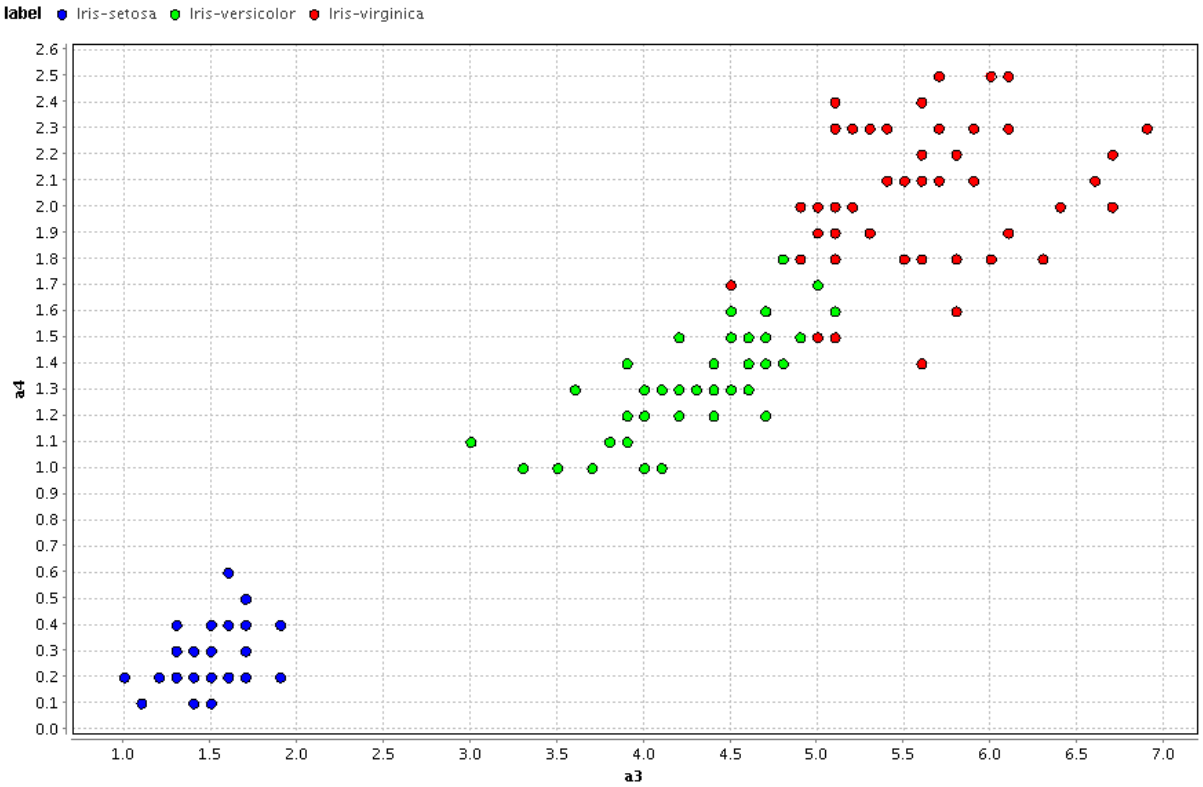
Rys. 4 Rozkład punktów zbioru test1.csv

Zamień uzyskane drzewo decyzji na odpowiadającą mu listę reguł.

Dla każdej reguły wyznacz współczynnik wsparcia i ufności reguły.

Zad. 2.

Powtórz zadanie Zad 1, zamieniając operator W-J48 na algorytm indukcji reguł W-JRip. Operator ten stanowi implementację algorytmu Ripper indukcji reguł. Podobnie jak w zadaniu 1, dokonaj wizualizacji uzyskanych wyników poprzez wrysowanie uzyskanych reguł w wykres



Rys. 4. Zastanów się i odpowiedz na pytanie czy listę reguł zawsze można przedstawić w postaci drzewa. Jeśli tak, to dla uzyskanych wyników zaproponuj przekształcenie listy reguł w odpowiadające jej drzewo decyzji

Dla każdej reguły wyznacz współczynnik wsparcia i ufności reguły.

Zad. 3.

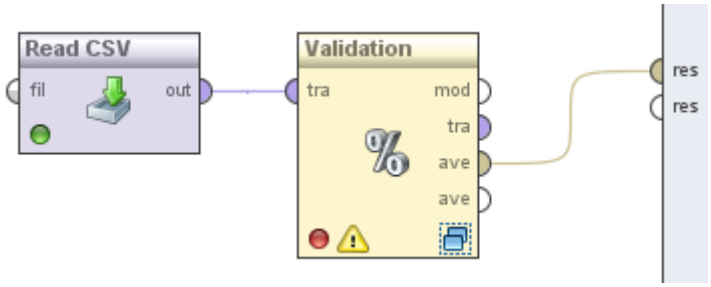
Pobierz zbiory zadań test2.csv oraz test3.csv, a następnie dokonaj analizy dokładności zbudowanego drzewa i wpływu parametru *confidence* na dokładność oraz rozmiar drzewa (liczbe liści, liczbe węzłów i głębokość drzewa). Dla każdego ze zbiorów wykreśl wykres pokazujący zależność dokładność = f(liczba_węzłów). Skomentuj uzyskany wynik. Uwaga: Realizację zadania można uprościć poprzez wykorzystanie operatora *Loop* lub *Loop parameters.* Parametr *Confidence* przetestuj w zakresie [0.001-0.5]

Zad. 4.

Powtórz zadanie 3 dla algorytmu sekwencyjnego pokrywania i zarejestruj wpływ parametru N (minimal weights of instances). Parametr N przetestuj w zakresie (0.01 - 20)

Zad. 5.

Wyniki uzyskane z zadań 3 powtórz używając do oceny dokładności test krzyżowy. W tym celu zbuduj projekt jak na Rys. 5



Rys. 5 Proces w programie RapidMiner z wykorzystaniem testu krzyżowego

Podobnie jak w zadaniu 3 dokonaj rejestracji dokładności modelu w funkcji parametru *Confidence*. Zwróć uwagę, iż obecnie program ocenia dokładność na danych których nie używał do procesu uczenia. Nanieść uzyskany wykres na rysunek z zad. 3. Skomentuj uzyskane wyniki

Zad. 6.

Powtórz obliczenia z zad 4, jednak do estymacji dokładności skorzystaj z operatora *Validation*. Podobnie jak w zad 5. nanieś uzyskany wynik na wykres z zadania 4. Skomentuj uzyskane wyniki

Zad. 7.

Porównaj zestawy reguł będących wynikiem działania drzewa i algorytmu sekwencyjnego pokrywania. Skomentuj uzyskane wyniki (porównując uwzględnij liczbę przesłanek w regułach, liczbę reguł oraz dokładność)