Klasyfikator

Dokumentacja

Autorzy:
Monika Kogut
Marta Kornaszewska
Katarzyna Kuzawska

Spis treści

1. Cel dokumentu

Celem dokumentu jest opis zasady działania oraz implementacji klasyfikatora. Projekt dotyczył implementacji trzech etapów różniących się rodzajem wykorzystywanego automatu.

2. Opis zadania

Zadanie polega na implementacji automatu klasyfikującego symbole z danego zbioru do odpowiednich klas. W pierwszym etapie implementacja obejmuje automat deterministyczny, w drugim niedeterministyczny, a w trzecim automat rozmyty.

Konstrukcja automatu odbywa przy użyciu zbioru uczącego. Funkcja przejścia optymalizowana jest dzięki wykorzystaniu algorytmu PSO (Particle Swarm Optimization). Testowanie automatu odbywa się przy użyciu zbioru testowego.

2.1 Problem klasyfikacji

$$x_i \rightarrow (x_{i_1,} \dots x_{i_k})$$

$$T = \{x_1, x_2, \dots, x_n\}$$

Dany jest skończony zbiór symboli. Każdy symbol opisany jest za pomocą k wartości z dziedziny pewnych atrybutów warunkowych (cech) .

Problemem klasyfikacji jest wyznaczenie klasy, do której dany symbol należy na podstawie zbioru wartości jego cech. Klasyfikacja odbywać się będzie za pomocą automatu wykorzystującego do klasyfikacji cechy danego symbolu.

2.2 Konstrukcja klasyfikatora

Ogólny schemat, wspólny dla wszystkich trzech etapów implementacji przedstawiony jest poniżej. Poszczególne etapy opisane są bardziej szczegółowo w dalszej kolejności.

1. Budujemy zbiór uczący i testowy zawierający pewną liczbę wystąpień (kopii) symbolu. Każda kopia reprezentowana jest za pomocą cech. W implementacji tworzyliśmy po 1 zestawie cech dla każdego rodzaju symboli (tworzyliśmy po jednym „idealnym" reprezentancie danego symbolu), a następnie da każdej z cech zaburzaliśmy ją rozkładem normalnym w celu uzyskania pozostałych reprezentantów danego symbolu.
2. Wartości cech muszą być znormalizowane, najlepiej z zakresu [0, 1].
3. Zbiór testowy tworzony jest jak zbiór uczący.
4. Zbiór uczący wykorzystywany jest przez algorytm PSO do zoptymalizowania funkcji przejścia, natomiast zbiór testowy wykorzystywany jest do sprawdzenia poprawności działania automatu.

5. Wartości cech dzielimy na podprzedziały i każdemu z podprzedziałów przypisujemy opis, np. literkę. Zbiór cech stanie się wtedy zbiorem symboli, a każda klasa zbiorem słów (będzie tworzyć język).
6. Automat liczący będzie miał tyle stanów, ile jest klas. Między pewnymi stanami będą przejścia. Obliczenie automatu dla symbolu zakończy się w klasie, do której go zaklasyfikujemy.
7. Automat wskazywać będzie symbole obce za pomocą dodatkowego stanu.
8. Losujemy początkową konfigurację automatu i podajemy do PSO funkcję błędu jako funkcję przejścia, którą będziemy optymalizować. Funkcja błędu przyjmuje wartość 0 jeśli element wejściowy został zaklasyfikowany poprawnie i 1 w przeciwnym przypadku. Na tej podstawie algorytm PSO optymalizuje rozwiązanie i po podanej liczbie iteracji (jako parametr wywołania) zwraca macierz funkcji przejścia automatu.
9. Dla tak uzyskanego automatu , po podaniu mu na wejście słowa opisującego dany symbol powinniśmy dostać odpowiedź do jakiej klasy należy.
10. Działanie utworzonego automatu testujemy na danych testowych, które wcześniej wygenerowaliśmy (pkt. 3).

2.4 Algorytm PSO

Algorytm PSO (Particle Swarm Optimization) to metoda optymalizująca zadany problem. Może służyć zarówno do jego maksymalizacji, jak i minimalizacji.

Algorytm opiera się na trzech głównych wartościach:
- wartości docelowej
- najlepszej znalezionej wartości ze wszystkich cząstek (najmniej odległej od pożądanego rozwiązania)
- wartości stopu, kiedy rozwiązanie nie zostanie znalezione.

Dla każdej cząstki przetrzymywane są także podane wartości:
- możliwe rozwiązanie
- wartość o jaką cząstka może być zmieniona
- najlepszej znalezionej wartości przez daną cząstkę

3. Środowisko programowania

Program napisany został z wykorzystaniem środowiska MATLAB.
Automat wykorzystywał gotowy algorytm PSO.

4. Etap pierwszy

Etap pierwszy obejmuje implementację klasyfikatora z użyciem automatu deterministycznego. W tym etapie nie uwzględniane były symbole obce.

4.1 Tworzenie zbioru uczącego

Zbiór uczący przekazywany jest w postaci macierzy. Każdy wiersz w macierzy odpowiada jednemu symbolowi, opisanemu znormalizowanym wektorem cech.

Przykładowy zbiór testowy przedstawiono poniżej. s_i oznacza i-ty symbol, natomiast c_j oznacza j-tą cechę. Klasa mówi, do której z klas symbol powinien zostać sklasyfikowany.

	klasa	$\mathbf{c_1}$	$\mathbf{c_2}$	**...**
$\mathbf{s_1}$	a	0.25	0.53	
$\mathbf{s_2}$	a	0.22	0.57	
$\mathbf{s_3}$	b	0.55	0.31	
...				

Zbiór uczący tworzony jest następująco:

$$n \times (m+1)$$

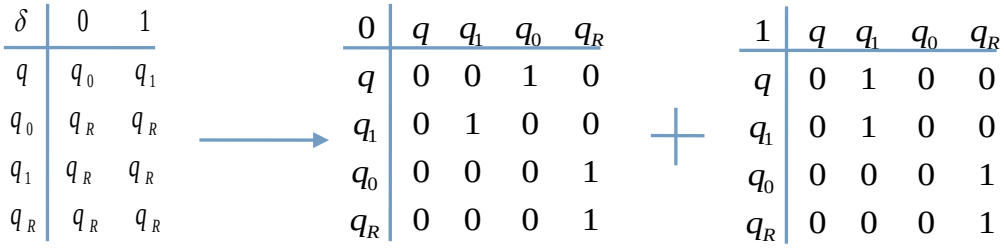
1. Tworzona jest macierz , gdzie n to liczba symboli uczących przemnożona przez liczbę kopii symboli, a m to liczba cech opisujących pojedynczy symbol.
2. W pierwszej kolumnie macierzy wpisywana jest oczekiwana klasa, do której na podstawie cech powinien zostać sklasyfikowany dany symbol. Przy czym każdy z symboli ma ustaloną liczbę kopii, np. dla liczby symboli 3 i kopii 2 otrzymamy podaną macierz:

	klasa	**...**
$\mathbf{s_1}$	a	
$\mathbf{s_2}$	a	
$\mathbf{s_3}$	b	
$\mathbf{s_4}$	b	
$\mathbf{s_5}$	c	
$\mathbf{s_6}$	c	

3. Kolejne kolumny wypełniane są losowymi wartościami dla pierwszej kopii każdego symbolu (dane generowane rozkładem jednorodnym).
4. Pozostałe kopie symboli uzupełniane są wartościami z pierwszej kopii symbolu, zaburzonymi rozkładem normalnym (o wartości średniej równej 0 i wariancji podawanej jako jeden z parametrów wywoływania programu)

4.2 Tworzenie funkcji przejścia

Tabelkę funkcji przejścia przejścia można zdekomponować do n tabelek przejścia, gdzie n to liczność alfabetu.



Funkcja przejścia dla automatu ma wtedy postać macierzy o trzech wymiarach. Przyjmijmy, że pierwsze dwa wymiary, określone dla konkretnego symbolu, nazywać będziemy tabelą przejścia dla symbolu. Takich tabel przejścia jest tyle, ile symboli liczy alfabet, nad którym konstruujemy automat (czyli ile podprzedziałów, na które dzieliliśmy zbiór wartości cech).

Pojedyncza tabela przejścia określa, z którego stanu możemy przejść w jaki stan, czytając na wejściu symbol tabeli. Wiersze i kolumny odpowiadają kolejnym stanom automatu. Tabela przejścia tworzona jest dla automatu deterministycznego, co oznacza, że z danego stanu przejść można jedynie do jednego innego stanu. Oznacza to, że w tabeli przejścia w każdym wierszu znajdować się może tylko jedna jedynka.

Przykładowo:

δ_i	q_1	q_2	q_3
q_1	1	0	0
q_2	0	0	1
q_3	0	1	0

Jest poprawną tabelą przejścia dla symbolu δ_i.

4.3 Symulacja automatu - Mnożenie macierzy

W przypadku automatu deterministycznego, do symulacji automatu używane jest zwykłe mnożenie macierzy (funkcji przejścia dla aktualnego symbolu) przez wektor (aktualny stan). W wyniku otrzymamy wektor będący nowym stanem, w którym automat znajdzie się po przejściu. Dla pierwszego mnożenia jako stan aktualny przyjmujemy stan losowy automatu.
W następnym kroku uaktualniamy symbol i mnożona jest macierz (funkcję przejścia dla symbolu) przez ostatni wynikowy wektor, który staje się nowym aktualnym stanem. W wyniku dostajemy kolejny stan, w jakim znajdzie się automat. Mnożenie wykonujemy dopóki nie wykorzystamy każdego symbolu w słowie. Ostatecznym stanem jest wynik ostatniego mnożenia.

5. Etap drugi

Etap drugi obejmuje implementację klasyfikatora z użyciem automatu niedeterministycznego. W tym etapie uwzględniane były symbole obce. Automat ma za zadanie rozpoznanie i odrzucenie znalezionego symbolu obcego.

5.1 Tworzenie zbioru uczącego

Zbiór uczący, podobnie jak w etapie pierwszym, przekazywany jest w postaci macierzy. Każdy wiersz w macierzy odpowiada jednemu symbolowi, opisanemu znormalizowanym wektorem cech. Dodatkowo zbiór uczący zawiera symbole obce, które nie należą do żadnej z klas rozpoznawanych docelowo przez automat.

5.2 Tworzenie funkcji przejścia

Funkcja przejścia, podobnie jak w etapie pierwszym jest macierzą o trzech wymiarach. Zawiera ona tyle stron, ile jest symboli alfabetu. Każda z tabel ma tyle kolumn i wierszy, ile jest stanów automatu.

Automat ma tyle stanów, ile jest klas symboli, oraz dodatkowy stan określony dla symbolu obcego. Jeśli automat znajdzie się w tym stanie, oznacza to, że rozpoznawany symbol jest symbolem obcym.

Pojedyncza tabela przejścia określa, z którego stanu możemy przejść w jaki stan, czytając na wejściu symbol tabeli. Wiersze i kolumny odpowiadają kolejnym stanom automatu. Tabela przejścia tworzona jest dla automatu niedeterministycznego, co oznacza, że z danego stanu przejść można do wielu stanów. Oznacza to, że w tabeli przejścia w każdym wierszu znajdować się może więcej niż jedna jedynka.

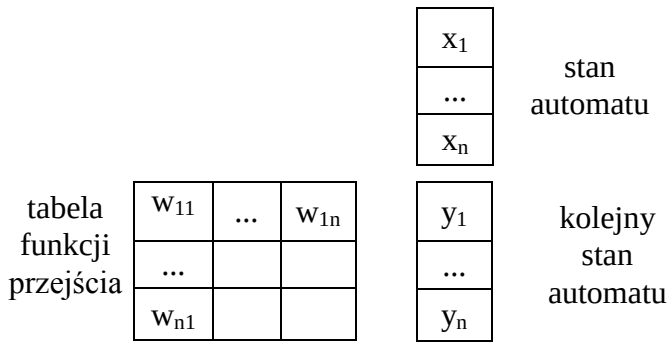
Przykładowo:

δ_i	q_1	q_2	q_3	q_R
q_1	1	1	0	1
q_2	0	1	1	1
q_3	0	1	0	1
q_R	0	0	0	1

Jest tabelą przejścia dla symbolu δ_i, a q_R jest stanem odrzucającym symbol obcy.

5.3 Symulacja automatu - mnożenie macierzy

W przypadku automatu niedeterministycznego, do symulacji automatu używane jest zmodyfikowane mnożenie macierzy, określone wzorem:



$$W_{n\times n} X_{n\times n} = Y_{n\times 1}$$

$$y_k = \sum_{i=1}^{n} w_{ki} x_i$$

$$y_k = \max\{\min\{ w_{ki}, x_i\}\}$$

$$\delta : \Sigma \times Q \times Q \to \{0,1\}$$

6. Etap trzeci

Etap trzeci obejmuje implementację klasyfikatora z użyciem automatu rozmytego. W tym etapie uwzględniane są symbole obce.

6.1 Tworzenie zbioru uczącego

Każdy symbol ze zbioru uczącego może należeć do danej klasy z pewnym prawdopodobieństwem, dlatego też ma on postać macierzy 3 wymiarowej , tzn. każdą cechę określa wektor o długości równej liczbie symboli automatu. Wartość dla poszczególnych klas z kolei wyliczamy za pomocą funkcji gaussowskiej. Funkcje gaussowskie definiujemy tak dla każdego z symboli, aby jej maximum było w środku danego podprzedziału. W naszym przypadku zastosowałyśmy funkcję

$$f(x) = \exp\left(-\left((x - x_0) * (x - x_0)\right)\right)$$

gdzie x_0 jest środkiem podprzedziału.

6.2 Tworzenie funkcji przejścia

Tabele funkcji przejścia, w przeciwieństwie do poprzednich etapów, zawierać będą wartości ułamkowe z przedziału [0, 1]. Wartości te oznaczać będą z jakim prawdopodobieństwem automat znajdujący się w danym stanie i czytający dany symbol znajdzie się w kolejnym stanie.

Przykładowo:

δ_i	q_1	q_2	q_3	q_R
q_1	0.25	0.39	0	0.55
q_2	0	0.11	0.35	0.22
q_3	0.64	0.17	0.41	0.12
q_R	0	0	0	1

Jest tabelą przejścia dla symbolu δ_i, a q_R jest stanem odrzucającym symbol obcy.

6.3 Symulacja automatu - Mnożenie macierzy

W przypadku automatu rozmytego, funkcje max i min używane do mnożenia macierzy automatu niedeterministycznego, zastąpione są normami trójkątnymi:

Mnożenie macierzy wygląda następująco:
Niech i oznacza i-ty symbol alfabetu, k oznacza k-ty wiersz tabeli przejścia, a j oznacza j-ty element wiersza i j-ty element wektora stanu.

1. Przyjmij i = 1.
2. Weź i-tą tabelę przejścia dla symbolu i utwórz kopię wektora opisującego obecny stan automatu. Przyjmij k = 1.
3. Weź k-ty wiersz tabeli przejścia. Przyjmij j = 1.
4. Weź j-ty element z wiersza tabeli (x_1) i j-ty element wektora stanu (x_2). Przeprowadź dla nich obliczenia wg powyższego wzoru. Wynik zapisz w j-tym elemencie wektora stanu.
5. Zwiększ j o 1. Idź do punktu 4, jeśli j nie przekracza długości wektora stanu.
6. Zwiększ k o 1. Idź do punktu 3, jeśli k nie przekracza ilości stanów.
7. Zwiększ i o 1. Idź do punktu 2, jeśli i nie przekracza ilości symboli alfabetu.

Dzięki powyższym obliczeniom otrzymaliśmy tyle wektorów stanów, ile jest symboli alfabetu. Następnie, aby otrzymać stan aktualny automatu należy złożyć te wartości za pomocą następującego wzoru:

max{min{ μ_{S1} , $konf_1$}, min{ μ_{S2} , $konf_2$},...} ,
gdzie min{ μ_{Si} , $konf_i$} to konfiguracja automatu je stopniem przynależności do stanu Si,
$konf_i$ - konfiguracja automatu po wykonaniu obliczenia dla Si