Algorytm LVQ

Wstęp

Algorytm LVQ jest typem sieci neuronowej. Neurony zwane tutaj również wektorami kodującymi są w rzeczywistości wektorami zdefiniowanymi w przestrzeni wejściowej z przypisanymi dla nich etykietami klas, przy czym liczba neuronów *l_w* << *l_x* jest znacznie mniejsza od liczby wektorów treningowych.
Koncepcja algorytmu LVQ bazuje na przyciąganiu do siebie wektorów kodujących przez wektory z tą samą etykietą klasy i odpychaniu od siebie wektorów kodujących o niezgodnych etykietach klas.
Algorytm uczenia sieci typu LVQ w wersji (1) polega na iteracyjnej aktualizacji położenia wektorów kodujących tak by zminimalizować błąd klasyfikacji. Proces iteracyjny składa się z głównej pętli która zwykle wykonywana jest określoną ilość razy np. 50, oraz pętli w której iteracja następuje po elementach zbioru treningowego. Wówczas dla każdego wektora treningowego poszukiwany jest najbliżej leżący wektor kodujący, jeżeli obydwa wektory (kodujący i treningowy) należą do tej samej klasy wówczas aktualizacja wag wektora kodującego odbywa się wg. zależności (1) jeżeli natomiast etykiety wektora kodującego i treningowego są różne aktualizacja wag odbywa się wg. zależności (2).

$$p_i(k+1) = p_i(k) + \alpha\left(x_j - p_i(k)\right) \quad (1)$$

$$p_i(k+1) = p_i(k) - \alpha\left(x_j - p_i(k)\right) \quad (2)$$

Gdzie:

- p_i – i-ty wektor kodujący
- x_j – j-ty wektor treningowy
- k - nr iteracji
- α - współczynnik uczenia (typowe startowe wartości to 0.01 / 0.02)

W trakcie iteracji wartość wsp. α powinna być aktualizowana wg. Zależności:

$$\alpha(k+1) = \frac{\alpha(k)}{1+\alpha(k)}$$

Opis algorytmu

Uczenie

1. Wylosuj położenie *l_w* neuronów dla każdej z klas. – w tym celu najlepiej jest wykorzystać istniejące już wektory danych i losowo wybrać ze zbioru danych wektory których położenie będzie inicjowało położenie neuronów
2. Iteracyjnie *l_iter* razy
   Dla każdego wektora treningowego
   Znajdź najbliższy wektor kodujący (dla danej metryki)
   1. Jeżeli etykieta neuronu jest zgodna z etykietą wektora treningowego dokonaj aktualizacji położenia (wag) neuronu zgodnie z zależnością (1)
   2. Jeżeli etykieta neuronu nie jest zgodna z etykietą wektora treningowego dokonaj aktualizacji położenia (wag) neuronu zgodnie z zależnością (2)

   Dokonaj aktualizacji wsp. α wg. zależności (3)

Testowanie

1. Policz odległości pomiędzy wektorem testowym a wszystkimi neuronami
2. Znajdź neuron leżący najbliżej wektora testowego (min(odległości(x,y)))
3. Przypisz etykietę najbliższego neuronu jako etykietę wektora testowego

W matlabie

1. Wczytaj dane
2. Dokonaj podziału na trening i test
3. Dokonaj standaryzacji / normalizacji danych zarówno treningowych jak i testowych
4. Wywołaj funkcję knn=lvq(tr,”Euclides”,x,100);
5. Wywołaj funkcję res = knn_test(knn,te);
6. Wyznacz dokładność uzyskanych wyników na przykładowych zbiorach danych

Gdzie

tr – zbiór danych treningowych

te – zbiór danych testowych

typ – przykładowy typ funkcji odległości np. „Euclid”

x – wektor opisujący luczbę wektorów prototypowych których należy znaleźć dla danej klasy. Np. [5 3] oznacza że dla klasy 1 ma być wyznaczonych 5 wektorów prototypowych (wektorów kodujących) natomiast dla klasy 2 mają być wyznaczone 3 wektory kodujące.

Funkcja:

```
function knn=lvq(tr,typ,l_w, l_it)
%uczenie
%  l_w – wektor opisujący liczbę wektorów kodujących dla
każdej z klas
%  typ – typ funkcji odległości
%  l_it – liczba iteracji

Class_n = length(l_w);
P=[];
%Inicjowanie położenia wektorów kodujących
for i=1:class_n
     id = find(tr.Y==i);
     rp = randperm(length(id))
     P.X=[P.X ; tr.X(id(rp(1:l_w(i))),:)];
     P.Y=[P.Y ; tr.Y(id(rp(1:l_w(i))))];
end;
%uczenie
for i=1:l_it
     - dla każdego wektora treningowego
     - znajdź najbliższy wektor kodujący (tutaj wykorzystaj
       funkcję odległości z poprzednich zajęć)
     - dokonaj jego aktualizacji wg. zależności odpowiednio
       (1) lub (2)
     - zmodyfikuj wsp. uczenia wg. zalezności (3)
end;
%Przygotuj strukturę opisująca model kNN
knn.k = 1;
knn.typ = typ;
```

```
knn.tr = P;
```

Zadanie:

1. Zbadaj wpływ funkcji odległości na jakość uczenia algorytmu LVQ
2. Zbadaj wpływ liczby wektorów kodujących na jakość uczenie algorytmu LVQ

Obliczenia przeprowadź dla różnych zbiorów danych.