Algorytm grupowania danych typu kwantyzacji wektorów

Wstęp

Definicja problemu: Typowe, problemem często spotykanym w zagadnieniach eksploracji danych (ang. data mining) jest zagadnienie grupowania danych lub klasteryzacji. Polega ono na odnalezieniu w zbiorze danych **X** składającym się z *n*-wektorów X = [$\mathbf{x}_1$, $\mathbf{x}_2$ … $\mathbf{x}_n$] zbioru obiektów podobnych.
Zagadnienie to jest problemem nietrywialnym i wymagającym zbudowania odpowiedniej heurystyki. W tym celu opracowano szereg podejść pozwalających na odnalezienie klastrów/podgrup danych. Poniżej zostaną omówione dwa wybrane algorytmy należące do grupy metod tzw. minimalizacji skalarnego współczynnika jakości. Idea ich działania sprowadza się do minimalizacji funkcji celu – jest to więc to problem optymalizacyjny. W celu zdefiniowania odpowiednich algorytmów / metod opracowano szereg rozwiązań w tym wywodzące się z obszaru statystyki – jak algorytm k-średnich, czy też z grupy sieci neuronowych jak np. algorytm kwantyzacji wektorów – VQ. W rzeczywistości obydwa starają się minimalizować wariancję w poszczególnych grupach reprezentowanych przez środki klastrów – centroidy zwane również prototypami, wektorami referencyjnymi czy też wektory kodujące.

Przykładem zastosowania metod klasteryzacji danych jest problem znajdywania grup podobnych dokumentów zwracanych przez wyszukiwarkę. (np. http://clusty.com/), czy też odnajdywanie ludzi o podobnych preferencjach / zainteresowaniach co poprawia skuteczność reklamy.

Algorytm VQ

Algorytm VQ (Victor quantization) został opracowany przez T Kohonenna całość to optymalizacja funkcji kosztu poprzez tzw. gradient stochastyczny. Algorytm rozpoczyna od wylosowania k – wektorów kodujących – potencjalnych środków klastrów, które następnie iteracyjnie zmieniają swoje położenie. Całość procedury odbywa się poprzez adaptację położenia wektora kodującego $\boldsymbol{p_i}$ po prezentacji $\mathbf{x_j}$ ‘tego wektora danych wg. zależności:

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

Gdzie
α - współczynnik uczenia
p_i – i-ty wektor kodujący leżący najbliżej wektora treningowego x_j

Wartości α powinny być aktualizowane wg. zależności $\alpha = \frac{\alpha}{1+\alpha}$ po każdej iteracji algorytmu.
Algorytm VQ do działania wymaga zdefiniowania odpowiednich wartości α_0 oraz liczby wektorów kodujących *k*.

Opis algorytmu

Uczenie

1. Wylosuj położenie *k* neuronów (prototypów) - w tym celu najlepiej jest wykorzystać funkcję *P* = *inicializacja*(***X***,*k*)

2. Iteracyjnie z-razy
   Dla każdego wektora treningowego
   Znajdź najbliższy wektor kodujący czyli dla danego x_i znajdź najbliższych wektor z ***P***
   Dokonaj aktualizacji położenia (wag) neuronu zgodnie z zależnością (1)
   Dokonaj aktualizacji wsp. α wg. zależności (2)
1. Ponumeruj wektory kodujące od 1:k
2. Dla każdego wektora zbioru danych
   a. Policz odległości pomiędzy wektorem danych $\mathbf{x}_i$ a wszystkimi neuronami (wektorami kodującymi)
   b. Znajdź neuron leżący najbliżej wektora danych (min(odległości2($\mathbf{x}_i$,$\mathbf{p}_j$)))
   c. Przypisz numer najbliższego wektora kodującego do danego wektora testowego

W matlabie

1. Uruchom skrypt ***zad2_vq.m***, w celu jego poprawnego działania musisz zaimplementować funkcję ***vq.m***

Funkcja:

```
function [C,P] = vq( X, k )
%Funkcja służy do wykonania klasteryzacji w oparciu o algorytm
kwantyzacji wektorów VQ
%Parametrami wejściowymi funkcji są X - macierz, w której kolejne
%wiersze to kolejne rekordy danych, k-oznacza liczbę klastrów do
%znalesienia. C- to kolumna zawierająca przydział kolejnych wektorów
%z X do odpowiednich klastrów numerowanych od 1:k
%Przykład:
%   X = [1 1;
%            1 2;
%            5 5;
%            6 5;
%            5 6];
%   k=2;
%   C = vq(X,k);
%   wówczas
%   C=[1;
%      1;
%      2;
%      2;
%      2];
%bo pierwsze dwa punkty leżą obok sibie, podobnie kolejne 3

%stałe:
z = 100;
alpha = 0.01;
P = inicializacja(X,k);
[n,m]=size(X); %gdzie n - liczba wektorów, m-liczba kolumn
%============== Wyznaczanie położenia środków klastró
================
for a=1:z
    %dla każdego wektora treningowego
        %policz odległości wektora treningowego do wszystkich
wektorów kodujących
```

```
        %d = odległość między X(i,:) a wszystkimi wektorami P -
możesz tutaj
        %wykorzystać funkcję odleglosc2()
        % znajdź najbliższy wektor kodujący
        %dokonaj aktualizacji najbliższego wektora kodującego
(wekotra o indeksie min_id) wg. zależności (1)
    %zmodyfikuj wsp. uczenia wg. zalezności (2)
end;
%=============== Przydzielenie wektorów zbioru danych do klastrów
C = zeros(n,1); %Rezerwacja pamięci przeznaczonej na wyniki
for i=1:n
    %Znajdź numer wektora kodującego, który leży najbliżej wekora
X(i,:)
    %Przypisz numer wektora kodującego do zmiennej C zawierajacej
    %informację o przynależności danego wektora do klastra
    C(i) = min_id;
end;
```

Algorytm k-śrtednich (k-means)

Jest wiele odmian algorytmu k-średnich, jego najczęściej spotykana wersja działa na zasadzie wsadowej (batch) tzn. najpierw tworzona jest tablica/macierz przynależności *U* o wymiarach: *n* x *m* gdzie *m* to liczba klasterów, *n* – to liczba wektorów danych. Macierz ta ma postać binarną tzn. przyjmuje wartości 1 jeśli dany wektor danych należy dodanego klastra oraz 0 jeśli dany wektor danych do danego klastra nie należy.

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \begin{array}{c} \begin{array}{ccc} p_1 & p_2 & p_3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{array}$$

Każdy wiersz powyższej macierzy powinien spełniać zależność iż suma elementów wiersza musi być mniejsza od liczby wektorów danych. $\underset{j\in[1,l\_w]}{\forall}\sum_{i=1}^{l\_n} U_{j,i} < l\_n$

Takie ograniczenie pozwala zabezpieczyć się aby pojedynczy klaster nie zagarnął wszystkich wektorów danych. Drugie ograniczenie służy zapewnieniu by dany wektor przynależał dokładnie do jednej grupy, a sprowadza się ono do zależności $\underset{i\in[1,l\_n]}{\forall}\sum_{j=1}^{l\_w} U_{j,i} = 1$

Algorytm k-średnich działa na zasadzie realizacji dwóch kroków 1) wyznaczenia położenia środków centrów klasterów *P* na podstawie macierzy U 2) aktualizacji macierzy U na podstawie nowych środków centrów *P* W poniższym kodzie algorytmu kŚrednich brakuje implementacji powyższych dwóch funkcji. Uzupełnij go o brakujące funkcję i wykonaj pozostałe zadania laboratoryjne

Opis algorytmu

Uczenie

1. Losowo wygeneruj macierz podziału U
2. Na podstawie macierzy U wyznacz środki klastrów

3. Dokonaj aktualizacji macierzy U, (przypisz dane do odpowiednich środków klasterów P)
4. Porównaj postać macierzy podziału U. Jeśli nie uległy zmianie to koniec
5. Idź do 2

W matlabie

1. Uruchom skrypt ***zad1_ksredich.m***, w celu jego poprawnego działania musisz zaimplementować funkcję ***ksrednich.m***

Funkcja:

```
function [C,P] = ksrednich(X,k)
%Funkcja powinna dokonać analizy skupień na danych "X". W jej wyniku
ma
%powstać "k" klastrów. Wektor "C" oznacza przynależność do danego
klastra, natomiast P to zbiór wektorów określających położenie
środków klastrów
%Przykład:
%   data = [1 1;
%           1 2;
%           5 5;
%           6 5;
%           5 6];
%   k=2;
%   C = ksrednich(data,k);
%   %wówczas
%   C=[1;
%      1;
%      2;
%      2;
%      2];
%bo pierwsze dwa punkty leżą obok sibie, podobnie kolejne 3
%Przydatne funkcje:
%   d = odleglosc2(x,y) %funkcja wyznacza długość między dwoma
wektorami x
%   i y
%Przydatne funkcje:
%   d = odleglosc2(x,y) %funkcja wyznacza długość między dwoma
wektorami x
%   i y
%   [n,m]=size(data); %gdzie n - liczba wektorów, m-liczba kolumn
%stałe:
U = inicializacja(X,k);
maxIteracji = 20;
%============== Wyznaczanie położenia środków klastrów
for i = 1 : maxIteracji

    P = obliczP(X,U);
    oldU = U;
    U = obliczU(X,P);

    if all(all(U == oldU)), %Zakończ pętlę jeśli macierz podziału
nie ulega zmianie
        break
    end;
end;
```

```
%=============== Przydzielenie wektorów zbioru danych do klastrów
n = size(X,1);
C = zeros(n,1);
for i=1:n
    id = find(U(:,i));
    C(i) = id;
end;
end %Koniec funkcji ksrednich

function U = obliczU(X,P)
%Napisz funkcję obliczającą macierz przynależności U
end %Koniec funkcji obliczU

function P = obliczP(X,U)
%Napisz funkcję obliczającą aktualne środki klastrów
end %Koniec funkcji obliczP

function U = inicializacja(X,k)
    n = size(X,1);
    r = randperm(n);
    U = false(k,n);
    for i=1:k
        U(i,r(i:k:n)) = true;
    end;
end %Koniec funkcji inicjalizacja
```

Standaryzacja

Standaryzacja jest standardową procedurą poprzedzającą działanie większości algorytmów eksploracji danych. Polega ona na doprowadzeniu do sytuacji, w której każda kolumna (zmienna/echa) ma średnią = 0 i odchylenie std. = 1. Innymi słowy dla każdej kolumny X(:,i) należy wyznaczyć średnią za pomocą funkcji *mean()* oraz odchylenie standardowe za pomocą funkcji *std()* a następnie dokonać transformacji

$$z = \frac{x - \mu}{\sigma}$$

Gdzie

μ - oznacza średnią

σ - oznacza odchylenie standardowe

Zadanie:

1. Zaimplementuj algorytm VQ
2. Zaimplementuj algorytm kSrednich
3. Do weryfikacji użyj odpowiednich skryptów *zad1 /zad2*, które testują wyniki klasteryzacji i wyświetlają je w postaci wykresu.
4. Uruchom skrypt *zad3* i zapisz wyniki dla różnych wartości parametru *k*
   UWAGA spróbuj uruchomić skrypt kilka razy dla tej samej wartości parametru *k* (dla k około 5) czy są różnice i jakie jest ich źródło

5. Uruchom skrypt *zad3* dla k=2 i dokonaj interpretacji wyników – skomentuj uzyskany wynik .
6. Dla algorytmu VQ umieść na końcu pętli: `for a=1:z` moduł rysujący trajektorię zmiany położenia wektorów kodujących.

7. Dokonaj poprawy działania algorytmu poprze wstępną standaryzację danych. W tym celu uruchom *zad4*, który wykonuje skrypt *zad3*z uwzględnieniem wspomnianej standaryzacji. Czy wyniki uległy poprawie?