Klasteryzacja obrazów z wykorzystaniem algorytmu Fuzzy-C-means

1. Wstęp

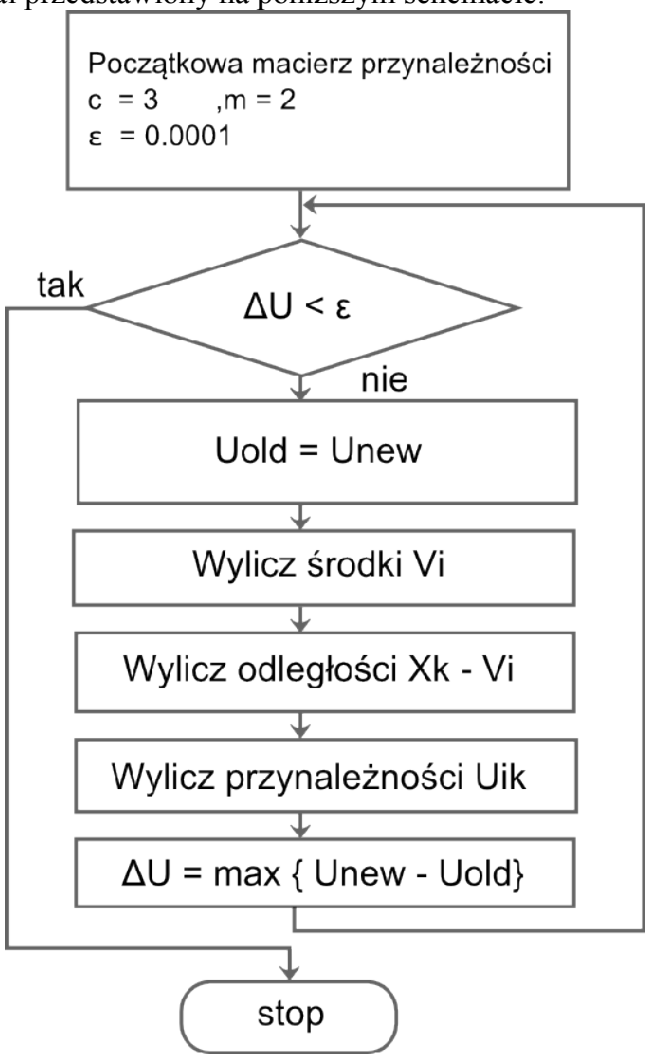
Proces klasteryzacji ma na celu podzielenie zbioru obiektów na kilka podzbiorów, tak, aby w każdym z nich znalazły się obiekty do siebie podobne. Odgrywa on kluczową rolę w wielu dziedzinach m. in. przy rozpoznawaniu określonych wzorców (tzw. 'pattern recognition'), analizie obrazów, przetwarzaniu dokumentów oraz badaniach rynkowych.
Istnieje wiele metod skutecznej klasteryzacji, możemy jednak wyszczególnić główne podgrupy algorytmów:

- klasteryzacja hierarchiczna
- klasteryzacja oparta na centroidach
- klasteryzacja oparta o modele dystrybucji
- klasteryzacja w oparciu o gęstość danych

2. Algorytm Fuzzy-C-means

Algorytm Fuzzy-C-means został po raz pierwszy zaprezentowany przez Dunn' a w 1973 r., a następnie rozszerzony przez Jamesa C. Bezdek'a w 1981 r. Podstawową cechą tego sposobu klasteryzacji jest to, iż każdy z obiektów może należeć do dowolnego zbioru w pewnym stopniu.

Przebieg algorytmu został przedstawiony na poniższym schemacie:



gdzie:

1) Podanie stopnia przynależności każdego ze sprawdzanych elementów do poszczególnych podzbiorów (macierz U^0).

2) Obliczenie środków (centroidów) każdego klastra wg. wzoru:

(1) $$v_{ij} = \frac{\sum_{k=1}^{n} (\mu_{ik})^m \cdot x_{kj}}{\sum_{k=1}^{n} (\mu_{ik})^m}$$

3) Wyliczenie odległości każdego z elementów od poszczególnych środków podzbiorów z użyciem normy Euklidesowej:

(2) $$d_{ij} = ||v_i - x_j|| = \sqrt{\sum_{k=1}^{c} (v_{ik} - x_{jk})^2}$$

4) Zaktualizowanie macierzy (macierz μ_{i+1}) informującej nas o przynależności poszczególnych elementów do określonych grup

(3) $$\mu_{ik} = \frac{1}{\sum_{s=1}^{c} \left(\frac{d_{ik}}{d_{sk}}\right)^{\frac{2}{m-1}}}$$

5) Jeśli

(4) $$\Delta U = max\left\{\left|\mu_{ik}^{t+1} - \mu_{ik}^{t}\right|\right\} > \varepsilon \quad ,$$

przejdź do kroku 2, traktując macierz U^{t+1} jako macierz wejściową,
w przeciwnym przypadku macierz U^{t+1} jest macierzą wynikową algorytmu

Rezultat klasteryzacji zależy od następujących parametrów:

c - ilość podzbiorów danych, które chcemy uzyskać

m - parametr rozmycia

ε - tolerancja algorytmu, im niższa wartość, tym lepsze otrzymane wyniki, ale też dłuższy czas obliczeń

Możemy zauważyć, że wyznaczane środki grup są średnią wszystkich punktów, przeskalowaną przez stopień przynależności danego punktu do podzbioru. Jednocześnie to jak bardzo dany element należy do danej grupy jest odwrotnie proporcjonalne do odległości od tego klastra. Zmniejszając błąd wyznaczenia przynależności obiektów w kolejnych iteracjach, FCM zwiększa dokładność procesu klasteryzacji.

3. Implementacja algorytmu Fuzzy-C-means

Do realizacji projektu wybrano język Java 7, mając na celu zapewnienie stabilności i niezależności od konfiguracji sprzętowej.

Dane do przetworzenia z użyciem FCM są pobierane z obrazu. Kolor każdego piksela jest określany za pomocą 3 składowych: czerwonej, zielonej i niebieskiej. W celu otrzymania wyniku bardziej zgodnego z rzeczywistością, następuje krok pośredni zamiany wartości kanałów na skalę szarości (Y, Cb, Cr, Cg) z użyciem następujących transformacji:

$$\begin{bmatrix} Y \\ Cb \\ Cr \\ Cg \end{bmatrix} = \begin{bmatrix} 0.299\,R & 0.587\,G & 0.114\,B \\ -0.169\,R & -0.331\,G & 0.500\,B \\ 0.500\,R & -0.419\,G & -0.081\,B \\ -0.169\,R & 0.500\,G & -0.331\,B \end{bmatrix}$$

Dane te są przechowywane w specjalnej tablicy typu int, dostęp do poszczególnych elementów jest realizowany z użyciem metody value(), wymagającej podania numeru piksela do którego się odwołujemy, oraz identyfikatora konkretnej wartości. Ze względu na duże zapotrzebowanie na pamięć w przypadku obrazów o znacznych wymiarach, wartości (mieszczące się w granicach [0, 255]) są pozyskiwane z pojedynczej liczby typu int przy pomocy masek bitowych.

Na wstępie algorytm konstruuje macierz przynależności pikseli do wszystkich podzbiorów. Ze względu na to, iż liczbę bajtów zajmowaną przez tą macierz możemy wyliczyć ze wzoru:

*s = liczba pikseli obrazu * parametr c symulacji * wielkość pojedynczej wartości typu float*

(pomijamy w tej chwili m.in. konieczność dodania 12 bajtów będących narzutem 'bycia obiektem' oraz wymóg podzielności ilości bajtów zajmowanych przez obiekt przez 8).
dla obrazu o wielkości 6 Mpix i c = 3 potrzebujemy zaalokować:

*s = 6000000 * 3 * 4b = ~68,7 Mb*

Jest to duże obciążenie dla 32 bitowej wirtualnej maszyny Javy o domyślnej wielkości sterty 256 Mb. Z tego względu wykorzystano możliwości klasy ByteBuffer z pakietu java.nio. Posiada ona statyczną metodę allocateDirect(), która jako parametr przyjmuje ilość bajtów, które ma zarezerwować w ramach bufora. Należy też wspomnieć, iż pamięć używana w ten sposób nie jest widoczna dla mechanizmu garbage collector i nie może zostać zwolniona gdy zachodzi taka potrzeba. Macierz ta jest początkowo wypełniana losowymi wartościami przy zachowaniu następujących wymogów:

$$\sum_{i=1}^{c} \mu_{ik} = 1\,,\ \forall\, k\,; \qquad 0 < \sum_{k=1}^{n} \mu_{ik} < n\,,\ \forall\, i$$

Pierwszym krokiem algorytmu jest wyliczenie środków podzbiorów, korzystając ze wzoru (1). Wyniki są zapisywane w niewielkiej tablicy. Na ich podstawie, dla każdego elementu wyliczana jest odległość od centroidu poszczególnych grup. Zgodnie ze wzorem (2), jedną z czynności, które należy wykonać jest pierwiastek kwadratowy z sumy. Dokładne badania z użyciem programu profilującego wykazały, że czas wykonania tej funkcji korzystając z biblioteki Math wchodzącej w skład Javy stanowi ~80% czasu wykonania wszystkich operacji w tej fazie algorytmu FCM. W przypadku, gdy ilość iteracji algorytmu jest duża, dokładność wyliczenia wszystkich danych dla poszczególnej iteracji jest mniej ważna niż szybkość działania. W tym celu zastąpiono odwołania do Math.sqrt() poniższą funkcją bazującej na algorytmie InvSqrt ze źródeł gry Quake III Arena dostępnych na licencji General Public Licence (jej autorstwo jest przypisywane Johnowi

Carmackowi lub Garemu Tarolli lub Terje Mathisenowi) :

```
public static float sqrt( float x ) {
    float xhalf = 0.5f * x;
    int i = Float.floatToIntBits( x );
    i = 0x5f375a86 - ( i >> 1 );
    x = Float.intBitsToFloat( i );
    x = x * ( 1.5f - xhalf * x * x );
    return 1 / x;
}
```

Działa ona o ponad połowę szybciej niż bibliotekowy odpowiednik, zachowując przy tym dużą dokładność obliczeń. Za każdym razem, gdy wyliczymy odległość danego elementu od któregoś z klastrów, zapisujemy wartość w tablicy pomocniczej, na podstawie której możemy obliczyć nowe stopnie przynależności do podzbiorów. Proces ten następuje dla każdego z pikseli obrazu(wzór(3)).

Algorytm Fuzzy-C-means powtarza wyżej opisane czynności, aż nie zostanie uzyskana odpowiednia dokładność danych (określona za pomocą parametru symulacji ε i wzoru (4)). W praktyce okazuje się jednak, że w przypadku niektórych obrazów proces ten trwa nawet kilka godzin. W celu ograniczenia ilości czasu spędzonego na analizie pojedynczego zdjęcia wprowadzono dodatkowe metody zakończenia działania algorytmu:

- gdy zostanie wykonana określoną ilość iteracji
- gdy użytkownik wpisze polecenie 'stop' w konsoli (w tym przypadku program dokończy obecnie wykonywane obliczenia nim przejdzie do następnej części procesu)

Ostatnią czynnością jest defuzzyfikacja rezultatu i przydzielenie pikseli do określonych podzbiorów. W tym celu analizowana jest ostateczna macierz przynależności U. Każdy piksel jest przydzielany do klastra do którego wartość ' należenia' jest największa. Często jednak 2 lub więcej z grup mają podobną wartość, co może delikatnie wpłynąć na rezultat. Obraz będący wynikiem działania algorytmu jest połączeniem defuzzyfikacji oraz tabeli kolorów, które mają zostać zastosowane do malowania (zadeklarowane w klasie main.Main).