Politechnika Śląska w Gliwicach

Zaawansowane Biblioteki Programistyczne
13 stycznia 2017

Grupowanie metodą k-średnich

Autor:
Bartłomiej Buchała

Informatyka SSM, semestr II
Rok akademicki 2016/2017
Grupa OS1

Spis treści

1. Wstęp

1.1 Temat projektu

Celem projektu było stworzenie algorytmu wykonującego grupowanie elementów metodą k-średnich w języku C++.

Program miał spełniać następujące warunki:

- Elementy wejściowe zdefiniowane przez zakres.
- Zakres elementu określany jest przez iteratory swobodnego dostępu.
- Na końcu wykonania algorytmu elementy wejściowe zostaną posortowane według grup.
- Funkcje odległości oraz uśredniająca przekazywane do algorytmu.
- Algorytm ma zwrócić wektor zakresów odpowiadających znalezionym grupom (wektor iteratorów wskazujących na początki lub koniec grup).
- Funkcja/Klasa imlementująca algorytm ma być szablonowa i pozwalać na grupowanie danych dowolnego typu.

1.2 Zasada działania algorytmu

Algorytm k-średnich (nazywany również algorytmem centroidów) jest jednym z algorytmów stosowanych w analizie skupień – analizy polegającej na szukaniu i wyodrębnianiu grup obiektów podobnych (skupień). Należy do grupy algorytmów niehierarchicznych (charakteryzuje się koniecznością podania liczby skupień).

Przy pomocy metody k-średnich zostanie utworzonych k różnych możliwie odmiennych skupień. Algorytm ten polega na przenoszeniu obiektów ze skupienia do skupienia tak długo aż zostaną zoptymalizowane zmienności wewnątrz skupień oraz pomiędzy skupieniami (lub alternatywnie zostanie wykonana maksymalna liczba iteracji).

Działanie algorytmu można uprościć w kilku punktach:

1. **Ustalenie liczby grup** – wybór ustalany odgórnie, w zależnośći od potrzeb. Jedną z metod ustalenia ilości skupień jest umowny jej wybór i ewentualna późniejsza zmiana tej liczby w celu uzyskania lepszych wyników.
2. **Ustalenie początkowych wartości środków skupień (centroidów)** – centroidy można dobrać na kilka sposobów:
   - Losowy wybór k obserwacji
   - Wybór k pierwszych obserwacji
   - Wybór taki, aby zaksymalizować odległości pomiędzy centroidami
   - Kilkakrotne uruchomienie algorytmu (z losowym wyborem) i wybór najlepszego spośród modeli.

   Jeżeli nie zna się dokładnie typu grupowanych danych, zaleca się aby początkowymi wartościami środków skupień były losowo wybrane pośród elementów wejścjowych.
3. **Obliczenie odległości obiektów od środków skupień** – określenie odległości elementów od centroidów przy użyciu określonej metryki (najczęściej stosuje się Euklidesową).
4. **Przypisanie obiektów do skupień** – Dla danej obserwacji porównywane są odległości od wszystkich centroidów i przypisanie elementu do skupienia, do którego środka ma najbliżej.
5. **Ustalenie nowych współrzędnych centroidów** – nowe wartości dla środków skupień wyznaczane zą sa pomocą funkcji uśredniającej (z reguły jest to średnia arytmetyczna).
6. **Wykonywanie kroków 3-5 do czasu, aż warunek zatrzymania zostanie spełniony** – warunkiem stopu może być ilość iteracji zadana na początku lub brak przesunięć obiektów pomiędzy skupieniami w 2 następujących po sobie iteracjach.

2. Analiza problemu

Przed przystąpieniem do pracy nad algorytmem, należało rozwiązać kilka problemów projektowych.

2.1 Problem doboru punktów początkowych

Podstawowym krokiem przy każdym wywołaniu algorytmu jest ustalenie punktów początkowych. Wyniki algorytmu k-średnich silnie zależą od wyboru punktów początkowych, jednak nie ma ściśle określonej reguły, jak je należy wybierać. W zależności od startowej wartości centroidów, wyniki grupowania mogą być inne z każdym uruchomieniem algorytmu (nawet przy identycznych parametrach). Początkowo wydaje się, że wybór losowych współrzędnych jest dobrym rozwiązaniem. Przeszkodą jest jednak założenia generyczności algorytmy – istnieje możliwość wyboru losowego punktu pośród wartości typu int lub klasy reprezentującej punkt na płaszczyźnie dwuwymiarowej, jednak zadanie jest znacznie utrudnione w przypadku np. typu string lub klasy zewnętrznej pochodzącej z innego projektu. W takim przypadku warunkiem uniemożliwiającym wybór losowych współrzędnych jest brak wiedzy na temat budowy typu lub nawet niemożliwość utworzenia obiektu o losowych współrzędnych.

Rozwiązaniem problemu jest wybór losowych elementów z zakresu podanego do grupowania. W pierwszej iteracji środki skupień znajdują się dokładnie w tym samym miejscu, w którym istnieją niektóre z danych wejściowych.

2.2 Miara odległości

Algorytm ma zapewniać generyczność – dlatego dla typu danych na którym chcemy wykonać grupowanie musi istnieć miara odległości (metryka), pozwalająca określić odległość pomiędzy dwoma dowolnymi elementami tego typu (w najczęstszym scenariuszu - pomiędzy daną wejściową, a centroidem). Zakłada się, że odległość ewaluowana jest jako wartość typu `double`. W celu zapewnienia poprawności działania algorytmu, konieczne jest przekazanie do funkcji grupującej obiektu funkcyjnego (z przeciążonym operatorem `()`) zwracającą wartość zmiennoprzecinkową, będącą miarą odległości pomiędzy 2 elentami. Przykład implementacji dla typy `int` znajduje się poniżej.

```
/// <summary>
/// Obiekt funkcyjny, bedacy miara odleglosci pomiedzy dwoma wartosciami typu int.
/// </summary>
struct Int_distance
{
        inline double operator()(int p1, int p2)
        {
                return (double)abs(p1 - p2);
        }
};
```

Sposób w jaki implementowana jest metryka leży w pełni w kwestii osoby korzystającej z algorytmu. Dla punktów w przestrzeni dwuwymiarowej może to być metryka euklidesowa lub miejska (Manhattan), dla typu char odległość dzieląca znaki w tablicy ASCII, itp.

2.3 Funkcja uśredniająca

Podobnie jak w przypadku metryki, z powodu generyczności funkcji konieczne jest zadeklarowanie obiektu funkcyjnego określającego, w jaki sposób obliczany będzie nowy centroid spośród n elementów należących do skupienia. Funktor powinien pozwalać na 2 operacje:

- Kumulację obiektów – dodanie nowego obiektu (przekazanego jako referencję) do skumulowanej wartości oraz inkrementacja licznika dodanych punktów. Realizowane przez przeciążony operator +=.
- Uśrednienie – wyliczenie nowych współrzędnych centroidu na podstawie skumulowanej wartości oraz liczby skumulowanych elementów. Realizowane przez przeciążony operator `()`. Jako parametr należy przekazać obiekt obliczanego typu – jest to poprzednia wartość środku skupienia. Jeżeli liczba obiektów należących do danej grupy jest większa niż 0, przekazany obiekt jest aktualizowany do wartości nowego centroidu. Jeżeli liczba elementów w grupie jest równa 0, centroid nie jest nadpisywany

Przykład implementacji uśredniającego obiektu funkcyjnego dla typi `int` znajduje się poniżej.

```
struct Int_average
{
        /// Wartosc domyslna nowego centroidu.
        int newCentroid = 0;

        /// Licznik kumulowanych wartosci.
        int count = 0;

        /// <summary>
        /// Przeciazony operator (), wyliczajacy nowa wartosc centroidu usredniona z n
            elementow.
        /// </summary>
        /// <param name="oldCentroid">Aktualna wartosc centroidu</param>
        inline void operator()(int &oldCentroid)
        {
                if (count != 0)
                        newCentroid = (int)(newCentroid / count);
                else
                        newCentroid = oldCentroid;

                oldCentroid = newCentroid;

                newCentroid = 0;
                count = 0;
        }

        /// <summary>
        /// Przeciazony operator +=, kumulujacy kolejna wartosc.
        /// </summary>
        /// <param name="value">Kolejna kumulowana wartosc.</param>
        inline void operator +=(const int & value)
        {
                newCentroid += value;

                count++;
        }
};
```

Warunkiem koniecznym do poprawnego działania algorytmu jest zadeklarowanie obiektu funkcyjnego z przeciążonymi operatorami () oraz += przyjmującymi po jednym parametrze typu tożsamego do grupowanych danych. Reszta szczegółów dotyczących implementacji leży w kwestii użytkownika.

2.4 Warunki stopu

Zaimplementowana funkcja pozwala na wybór jednego z dwóch (lub obu jednocześnie) warunku zatrzymania wykonywania algorytmu.

- Maksymalna liczba iteracji – program kończy grupowanie elementów po wykonaniu określonej liczby iteracji zadanej jako parametr wejściowy.
- Osiągnięcie stanu stabilnego – jako stan stailny określamy sytuację, w której w dwóch kolejnych iteracjach nie nastąpi żadna zmiana przynależności do skupienia.

Podanie warunku stopu jako parametr odbywa się przez typ wyliczeniowy.

3. Specyfikacja zewnętrzna

3.1 Wykorzystanie algorytmu

Program został przygotowany zgodnie z myślą wykorzystania podobną do algorytmu sortowania znajdującego się w standardowej bibliotece wzorców (`std::sort()`). Wykorzystanie algorytmu odbywa się w kilku krokach:

1. Dołączenie pliku nagłówkowego do projektu poprzez zastosowanie komendy:

```
#include "K_means.h"
```

2. Utworzenie elementu klasy szablonowej o konkretnym typie:

```
K_means<Typ_Danych> k_means;
```

Gdzie <Typ_Danych> oznacza dowolny, dostarczony przez użytkownika typ danych.

3. Wykonanie algorytmu grupowania k-średnich

```
result = k_means.Group(first, last, distanceMeasure, groupAverage, k, MaxIterations,
    StopCondition, PrintOutput);
```

Elementem zwróconym będzie wektor iteratorów wskazujących na pierwsze elementy grup posortowanych elementów.

4. Opcjonalnym krokiem jest wywołanie wyświetlenia elementów kolekcji za pomocą funkcji DisplayCollection. Aby funkcja działała poprawnie, elementy w zakresie muszą implementować operator «.

```
DisplayCollection(Iterator first, Iterator last)
```

3.2 Opis parametrów wywołania funkcji Group

first – jest iteratorem swobodnego dostępu (używanym w takich strukturach jak na przykład vector czy deque ze standardowej biblioteki wzorców) wskazującym na pierwszy element z grupowanego zakresu.

last – jest iteratorem swobodnego dostępu (używanym w takich strukturach jak na przykład vector czy deque ze standardowej biblioteki wzorców) wskazującym na ostatni element z grupowanego zakresu.

distanceMeasure – obiekt funkcyjny opisujący metrykę pomiędzy elementami danego typu. Więcej informacji można znaleźć w podrozdziale 2.2.

groupAverage – obiekt funkcyjny opisujący funkcję uśredniającą elementy danego typu. Więcej informacji można znaleźć w podrozdziale 2.3.

k – określa maksymalną liczbę iteracji algorytmu. W przypadku, gdy parametr `StopCondition` ustawiony jest na `StableState`, wartość tego parametru jest nieistotna.

StopCondition – określa warunek stopu (2.4). Jest to argument typu wyliczeniowego i przyjmuje jedną z 3 wartości:

- MaxIterations – warunkiem stopu jest wykonanie odpowiedniej liczby iteracji (podanej w parametrze `k`).
- StableState – program wykonuje się tak długo, aż w kolejnych iteracjach nie nastąpi zmiana przynależności do skupienia żadnego pośród wszystkich grupowanych elementów.
- Both – połączenie powyższych. Program wykonuje się identycznie jak w przypadku `StableState`, jednak może się zakończyć wcześniej jeżeli została wykonana określona liczba iteracji

PrintOutput – określa, czy informacje o współrzędnych centroidów i przynależności punktów do skupień mają być wypisywane na strumień wyjściowy. Uwaga: może znacznie zwiększyć czas wykonywania algorytmu.

Przykładowe wywołanie funkcji dla wektora o elementach typu całkowitego:

```
result = k_means.Group(vec.begin(), vec.end(), Point2D_distance(), Point2D_average(), 4, 3,
    StableState, false);
```

Gdzie `vec` jest wektorem ze standardowej biblioteki wzorców zawierającym wartości typu `int`, a `Point2D_distance()` i `Point2D_average()` są obiektami funkcyjnymi stworzonymi zgodnie z zasadami opisanymi w podrozdziałach 2.2 i 2.3.

3.3 Opis parametrów wywołania funkcji DisplayCollection

first – jest iteratorem swobodnego dostępu (używanym w takich strukturach jak na przykład vector czy deque ze standardowej biblioteki wzorców) wskazującym na pierwszy element z grupowanego zakresu.

last – jest iteratorem swobodnego dostępu (używanym w takich strukturach jak na przykład vector czy deque ze standardowej biblioteki wzorców) wskazującym na ostatni element z grupowanego zakresu.

Przykładowe wywołanie funkcji dla wektora o elementach typu całkowitego:

```
k_means.DisplayCollection(vec.begin(), vec.end());
```

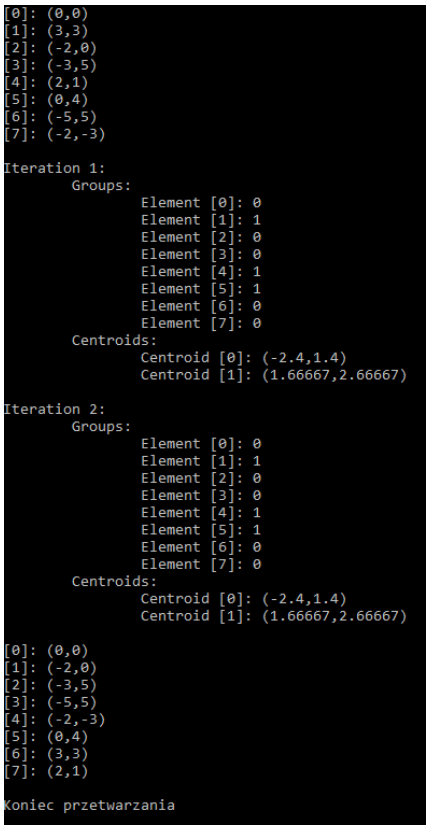
Gdzie `vec` jest wektorem ze standardowej biblioteki wzorców zawierającym wartości typu `int`.

3.4 Przykład scenariusza użycia

Dla poniższego kodu:

```
K_means<Point_2D> k_means;
vector<Point_2D> vec = vector<Point_2D>();

//Fill vec with data

k_means.DisplayCollection(vec.begin(), vec.end());

auto result = k_means.Group(vec.begin(), vec.end(), Point2D_distance(),
    Point2D_average(), 3, 2, StableState, true);

k_means.DisplayCollection(vec.begin(), vec.end());
```

Wynik działania będzie następujący:



Rysunek 1: Przykładowe wywołanie algorytmu

4. Specyfikacja wewnętrzna

5. Testowanie aplikacji

6. Podsumowanie