Zadanie 3 - Przetwarzanie i analiza dużych zbiorów danych

November 18, 2020

1 Algorytm k-średnich

Barbara Morawska 234096
Andrzej Sasinowski 234118
Marcin Markiewicz 234090

Napisaliśmy program, który implementuje algorytm k-średnich z uwzględnieniem dwóch miar: euklidesowej oraz Manhattan, oraz dwóch strategii rozmieszczenia centroidów początkowych dla losowego rozmieszczenia 3b.txt oraz dla najdalej oddalonych od siebie punktów 3c.txt

1.0.1 Kod programu

[ ]:
```
import sys
import math
from typing import List, Tuple


# Define constants
DIMENSIONS = 58
MAX_FLOAT_VALUE = sys.float_info.max
DISTANCE_MEASURE_METHODS = ['EUCLIDEAN','MANHATTAN']
INITIAL_ARRANGEMENTS  = ['3b.txt','3c.txt']

def prepare_data(txt: str) -> List[List[float]]:
    return  sc.textFile(txt)  \
           .map(lambda line: line.split(" ")) \
           .map(lambda line: [float(x) for x in line])
```

[ ]:
```
def euclidian_distance_sqr(x: List[float], y: List[float]) -> float:
    return sum([(a - b) ** 2 for a, b in zip(x, y)])

def manhattan_distance(x: List[float],y: List[float]) -> float:
    return sum(abs(a-b) for a,b in zip(x,y))
```

[ ]:
```
def clone_centroids(centroids: List[List[float]]) -> List[List[float]]:
      return [x[:] for x in centroids]


def point_add(point1: List[float], point2: List[float]):
    for i in range(len(point1)):
        point1[i] += point2[i]
```

[ ]:
```
def nearest_centroid(point: List[float], method: str) -> Tuple[int, List[float]]:
    closest_centroid = None
    closest_distance = MAX_FLOAT_VALUE
    for i, centroid in enumerate(centroids):
        if method == 'MANHATTAN':
            point_distance = manhattan_distance(centroid, point)
        else :
            point_distance = euclidian_distance_sqr(centroid, point)

        if point_distance < closest_distance:
            closest_distance = point_distance
            closest_centroid = i

    return closest_centroid, point

def update_centroid(centroid: Tuple[int, List[List[float]]], method: str) ->␣
↪Tuple[int, List[float], float]:
    centroid_id, points = centroid
    num_points = len(points)
    cost = 0

    new_centroid = [0] * DIMENSIONS

    for point in points:
        if method == 'MANHATTAN':
            cost += manhattan_distance(centroids[centroid_id], point)
        else :
            cost += euclidian_distance_sqr(centroids[centroid_id], point)
        point_add(new_centroid, point)

    return centroid_id, [value / num_points for value in new_centroid], cost
```

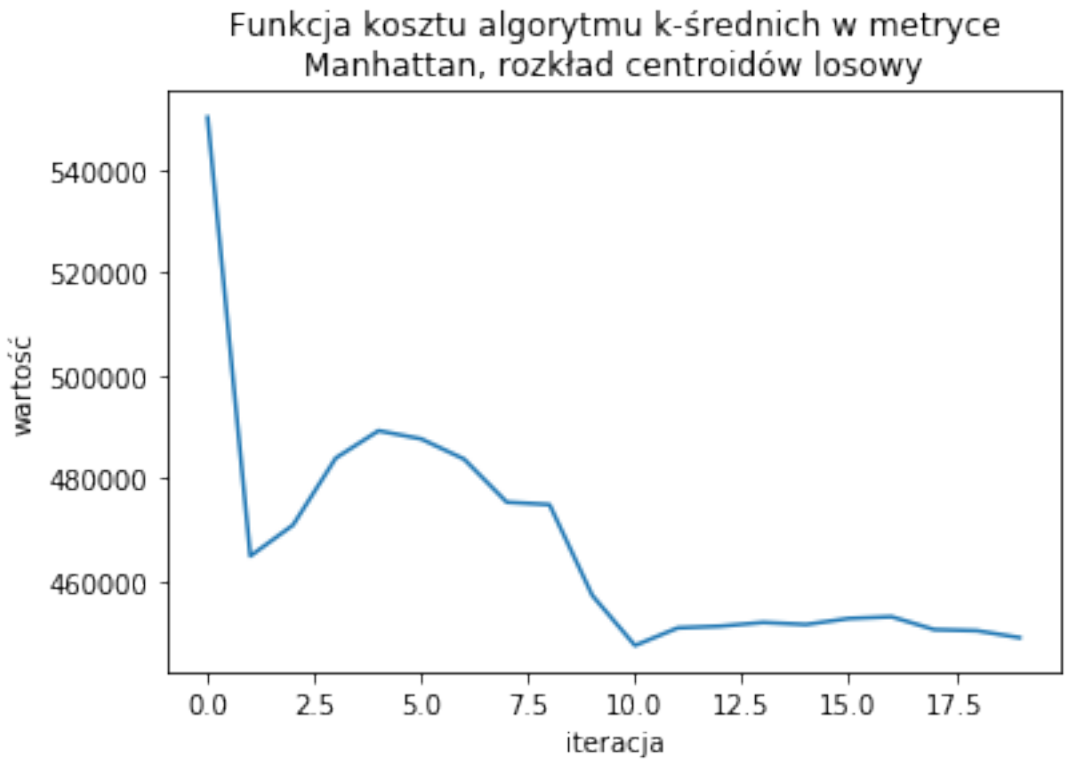
[ ]:
```
def display_results(centroids, costs,method,arrangement):
  # print centroids
  print('\n\nFOR FILE {} WITH {} DISTANCE \nCOMPLETED IN {} ITERATIONS\n\n '.
↪format(arrangement, method, iterations ))
  print('COSTS\n=======================================')
  print('{:>12}  {:>12} {:>12}'.format('iteration', 'cost', 'percentage cost␣
↪difference'))
```

```
  for i, cost_i in enumerate(costs):
      print('{:>12}  {:>12} {:>12}'.format(i, round(cost_i, 4),␣
 ↪str(round((costs[0]-costs[i])/costs[0] * 100,2))+'%'))
  print('\n\n')
  print('PERCENTAGE CHANGE IN COST {}% FOR {} DISTANCE'.
 ↪format(round((costs[0]-costs[9])/costs[0] * 100,2), method))
```

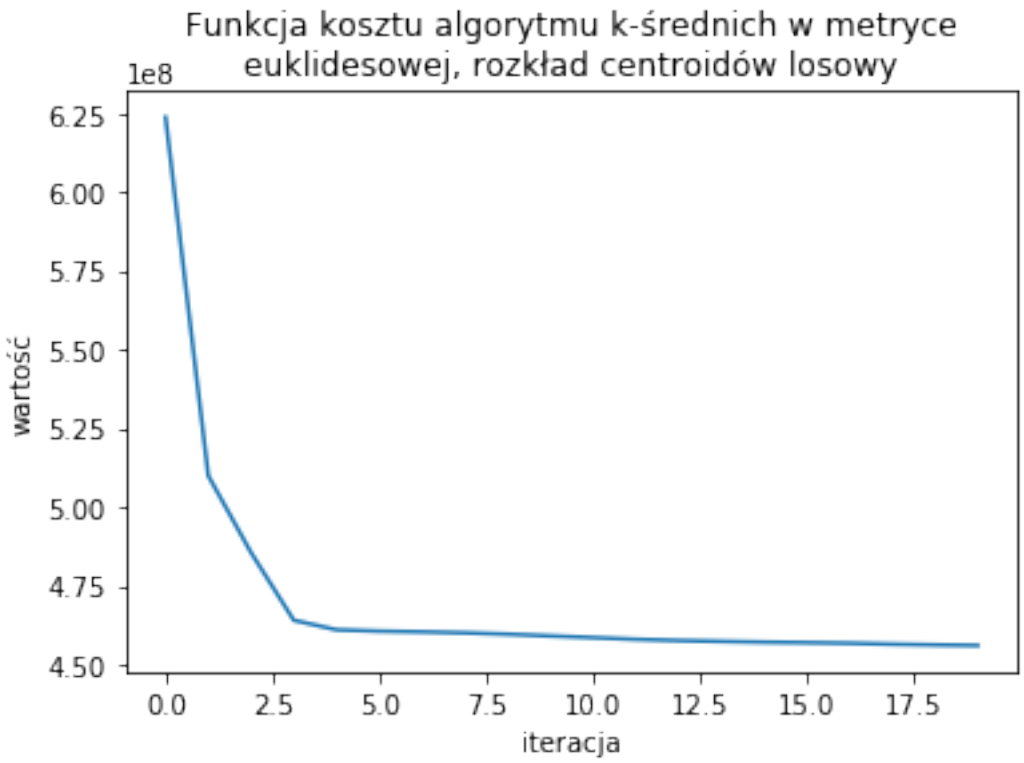
[ ]:
```
iterations = 20
allcosts = []
costs = []
dataset=prepare_data("dbfs:/FileStore/Zadanie3/3a.txt")
for arrangement in INITIAL_ARRANGEMENTS:
  for method in DISTANCE_MEASURE_METHODS:
    centroids = prepare_data("FileStore/Zadanie3/"+arrangement).collect()
    updated_centroids = clone_centroids(centroids)
    for i in range(0, iterations):
        centroids = clone_centroids(updated_centroids)
        nearest_centroids = dataset.map(lambda x: nearest_centroid(x,method))
        new_centroids = nearest_centroids.groupByKey().map(lambda x:␣
 ↪update_centroid(x,method)).collect()

        total_cost = 0
        for index, centroid_i, cost_i in new_centroids:
            total_cost += cost_i
            updated_centroids[index] = centroid_i

        costs.append(total_cost)
    display_results(centroids, costs, method, arrangement)
    allcosts.append(costs.copy())
    costs.clear()
```

1.0.2 Analiza wyników

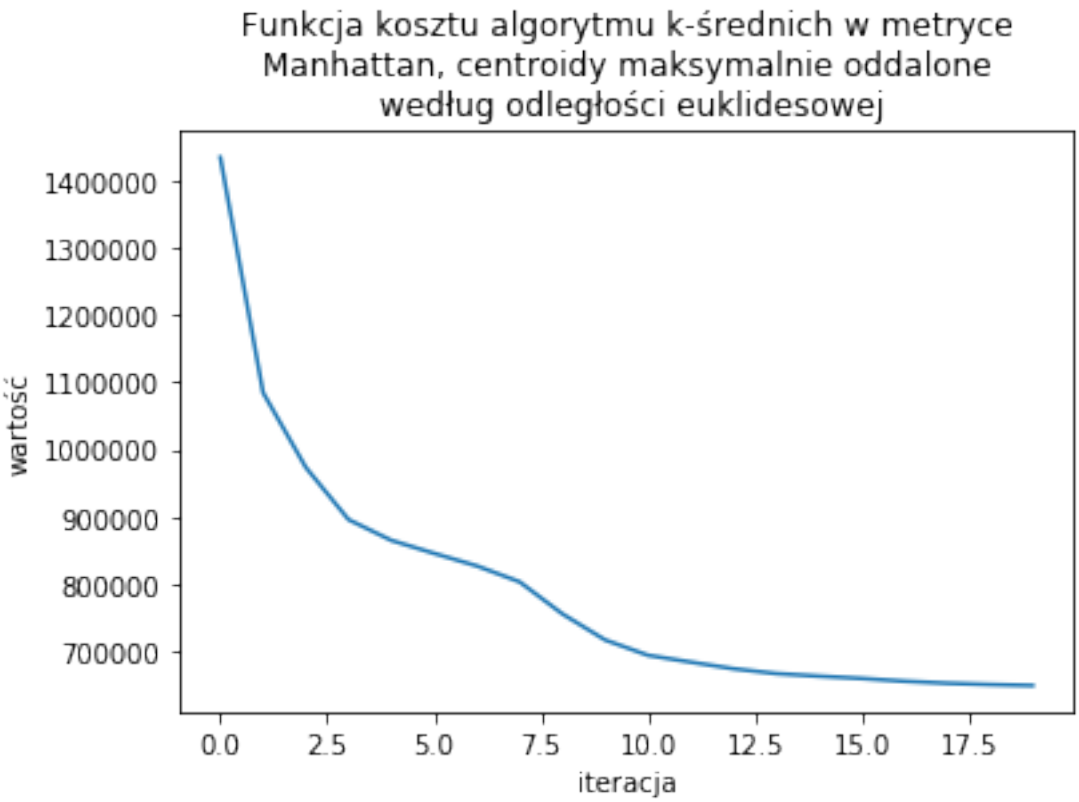
[ ]:
```
import matplotlib.pyplot as plt
plt.plot(allcosts[0])
plt.xlabel('iteracja')
plt.ylabel('wartość')
plt.title('Funkcja kosztu algorytmu k-średnich w metryce\nManhattan, rozkład␣
 ↪centroidów losowy')
plt.show()
```



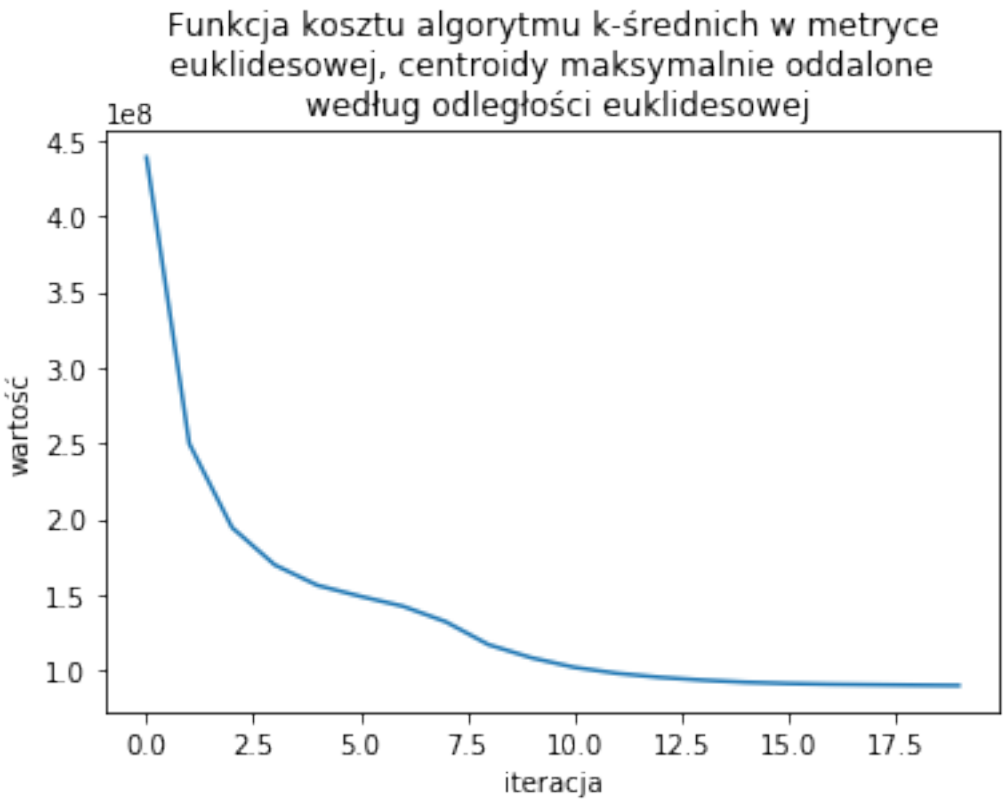
[ ]:
```
plt.plot(allcosts[1])
plt.xlabel('iteracja')
plt.ylabel('wartość')
plt.title('Funkcja kosztu algorytmu k-średnich w metryce\neuklidesowej, rozkład␣
 ↪centroidów losowy')
plt.show()
```



[ ]:
```
plt.plot(allcosts[2])
plt.xlabel('iteracja')
plt.ylabel('wartość')
plt.title('Funkcja kosztu algorytmu k-średnich w metryce\nManhattan, centroidy␣
 ↪maksymalnie oddalone\n według odległości euklidesowej')
plt.show()
```



[ ]:
```
plt.plot(allcosts[3])
plt.xlabel('iteracja')
plt.ylabel('wartość')
plt.title('Funkcja kosztu algorytmu k-średnich w metryce\neuklidesowej,␣
 ↪centroidy maksymalnie oddalone\n według odległości euklidesowej')
plt.show()
```



Numer iteracji	Miara Manhattan - plik 3b.txt	Miara Manhattan - plik 3c.txt	Miara euklidesowa - plik 3b.txt	Miara euklidesowa - plik 3c.txt
1	550117	1433739	623660345	438747790
2	464869	1084489	509862908	249803934
3	470897	973432	485480682	194494814
4	483914	895935	463997012	169804841
5	489216	865128	460969267	156295749
6	487630	845847	460537848	149094208
7	483712	827220	460313100	142508532
8	475331	803590	460003524	132303869
9	474871	756040	459570539	117170970
10	457233	717333	459021103	108547377
11	447494	694588	458490656	102237203
12	450915	684445	457944233	98278016
13	451250	674575	457558005	95630226
14	451975	667409	457290136	93793314
15	451570	663557	457050555	92377132
16	452739	660163	456892236	91541606
17	453083	656041	456703631	91045574
18	450584	653037	456404203	90752240

Numer iteracji	Miara Manhattan - plik 3b.txt	Miara Manhattan - plik 3c.txt	Miara euklidesowa - plik 3b.txt	Miara euklidesowa - plik 3c.txt
19	450369	651112	456177801	90470170
20	449011	649689	455986871	90216416
różnica % między 1 a 10 iteracją	**17%**	**50%**	**26%**	**75%**

Wnioski

- Najlepsze wyniki biorąc pod uwagę najmniejszy koszt operacji po 20 iteracjach został uzyskany przy metryce Manhattan i centroidach losowo wybranych ze zbioru danych.
- Najlepsze wyniki biorąc pod uwagę największą procentową różnicę pomiędzy 1, a 10 iteracją zostały uzyskane przy metryce euklidesowej i centroidach maksymalnie oddalonych według odległości euklidesowej.
- Mniejsze wartości funkcji kosztu zostały uzyskane w przypadku metryki Manhattan, chociaż zarejestrowana została w nich również mniejsza różnica procentowa między 1, a 10 iteracją w odpowiadających sobie rozkładach początkowych centroidów.
- W metryce euklidesowej początkowe wartości w obu przypadkach początkowego rozkładu centroidów były minimum 2 rzędy wielkości większe niż w metryce Manhattan.
- W przypadku obu metryk większą różnicę procentową między 1 i 10 iteracją dawały centroidy maksymalnie oddalone według odległości euklidesowej
- We wszystkich przypadkach w maksymalnie 10 iteracji rozpoczynało się wypłaszczanie funkcji kosztu. W prawie wszystkich przypadkach funkcja ta była monotoniczna, natomiast w przypadku metryki Manhattan i losowo rozmieszczonych centroidach, w funkcji kosztu można było zaobserwować wzrost między drugą i siódmą iteracją.