Zad3

November 17, 2020

[1]:
```
from pyspark import SparkConf, SparkContext
import math
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import StrMethodFormatter
```

[2]:
```
sc = SparkContext('local[*]')
```

[3]:
```
def parse_line(line):
    return [float(x) for x in line.split(' ')]


file = sc.textFile('3/3a.txt')
points = file.map(parse_line)

ITERATIONS = 20
```

0.0.1 Metrics declaration

[4]:
```
class Euclidean:
    def distance( point, centroid):
        distances = [(pi-ci) ** 2 for pi, ci in zip(point, centroid)]
        return math.sqrt(sum(distances))


    def cost(point, centroid):
        return Euclidean.distance(point, centroid) ** 2
```

[5]:
```
class Manhattan:
    def distance(point, centroid):
        distances = [abs(pi-ci) for pi, ci in zip(point, centroid)]
        return sum(distances)


    def cost(point, centroid):
        return Manhattan.distance(point, centroid)
```

0.0.2 KMeans declaration

[6]:
```
def map_nearest_centroid(point, centroids, metric):
    center_distance = [(index, metric.distance(point, c)) for index, c in␣
↪enumerate(centroids)]
    nearest_center = min(center_distance, key=lambda k: k[1])
    return (nearest_center[0], point)


def pair_cost(centroid_to_point, centroids, metric):
    c_index, point = centroid_to_point
    center = centroids[c_index]
    return metric.cost(point, center)


def calculate_mean(points):
    return np.mean(list(points), axis=0)


def kmeans_cost(max_iters, points, centroids, metric):
    iter_cost = []
    for _ in range(max_iters):
        centroid_to_point_pairs = points.map(lambda point:␣
↪map_nearest_centroid(point, centroids, metric))

        cost = centroid_to_point_pairs.map(lambda pair: pair_cost(pair,␣
↪centroids, metric)).sum()
        iter_cost.append(cost)

        centroids = centroid_to_point_pairs.groupByKey() \
                        .mapValues(calculate_mean) \
                        .values().collect()

    return iter_cost
```

0.0.3 Plotting methods

[7]:
```
def print_costs(costs):
    for i, c in enumerate(costs):
        print(f'{i}:  {c:14.2f}')

    cost_change = (iter_costs[0]-iter_costs[9])/iter_costs[0]
    print(f'\nCost change: {cost_change*100:04.2f}%')


def plot_costs(iter_costs, metric_name):
    plt.plot(iter_costs, marker='o', color='b', )
```

```python
    plt.ylabel('cost')
    plt.xlabel('iterations')
    plt.title(metric_name)
    plt.gca().ticklabel_format(axis='both', style='plain')
    plt.show()
```

0.0.4 Centers randomly selected from points

[8]:
```python
centroids = np.loadtxt('3/3b.txt')
```

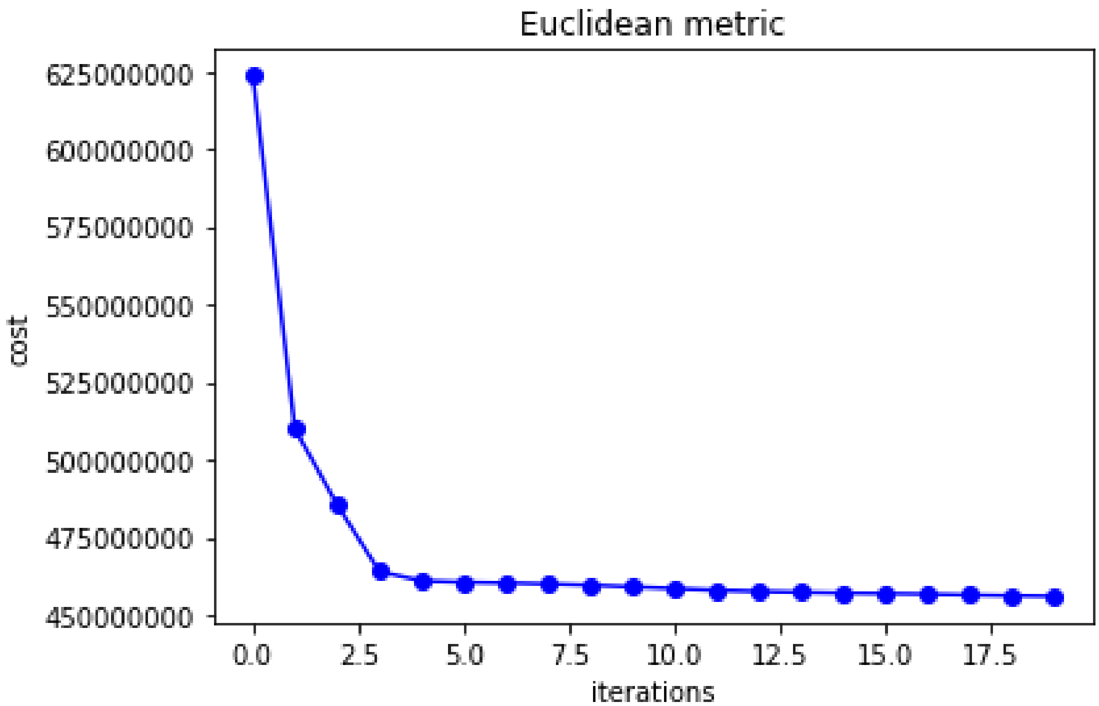
Euclidean metric

[9]:
```python
iter_costs = kmeans_cost(ITERATIONS, points, centroids, Euclidean)

print_costs(iter_costs)
plot_costs(iter_costs, 'Euclidean metric')
```

```
0:    623660345.31
1:    509862908.30
2:    485480681.87
3:    463997011.69
4:    460969266.57
5:    460537847.98
6:    460313099.65
7:    460003523.89
8:    459570539.32
9:    459021103.34
10:    458490656.19
11:    457944232.59
12:    457558005.20
13:    457290136.35
14:    457050555.06
15:    456892235.62
16:    456703630.74
17:    456404203.02
18:    456177800.54
19:    455986871.03

Cost change: 26.40%
```



Manhattan metric

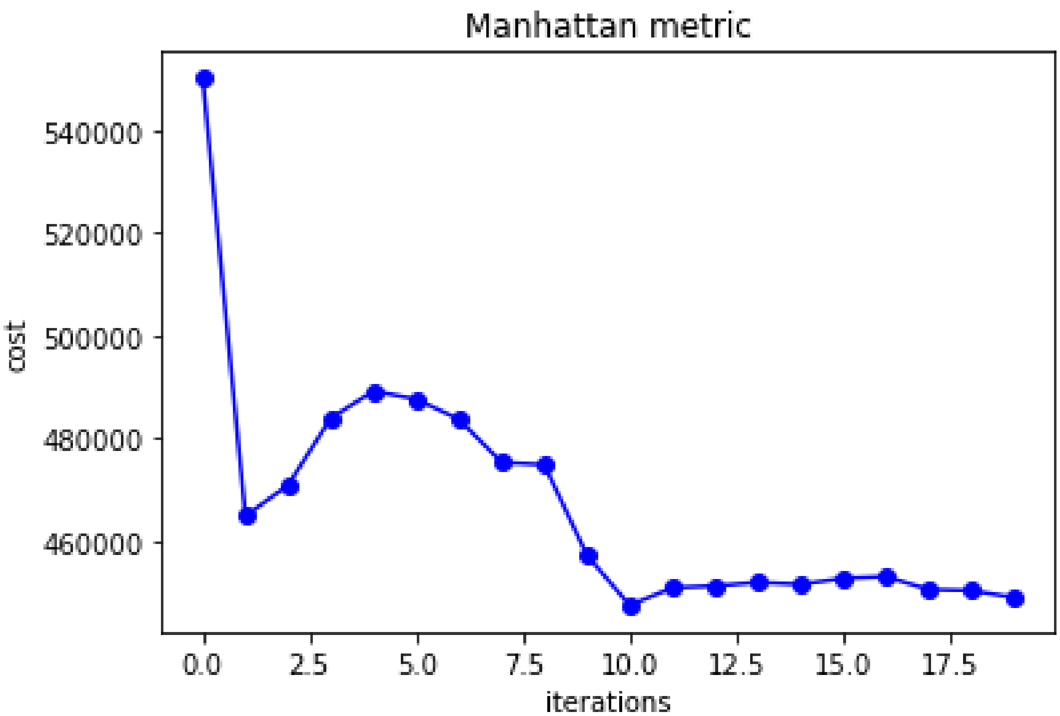
[10]:
```
iter_costs = kmeans_cost(ITERATIONS, points, centroids, Manhattan)

print_costs(iter_costs)
plot_costs(iter_costs, 'Manhattan metric')
```

```
0:        550117.14
1:        464869.28
2:        470897.38
3:        483914.41
4:        489216.07
5:        487629.67
6:        483711.92
7:        475330.77
8:        474871.24
9:        457232.92
10:        447494.39
11:        450915.01
12:        451250.37
13:        451974.60
14:        451570.36
15:        452739.01
16:        453082.73
17:        450583.67
```

```
18:       450368.75
19:       449011.36

Cost change: 16.88%
```



0.0.5 Centers from points located furthest away from each other

[11]:
```python
centroids = np.loadtxt('3/3c.txt')
```

Euclidean metric

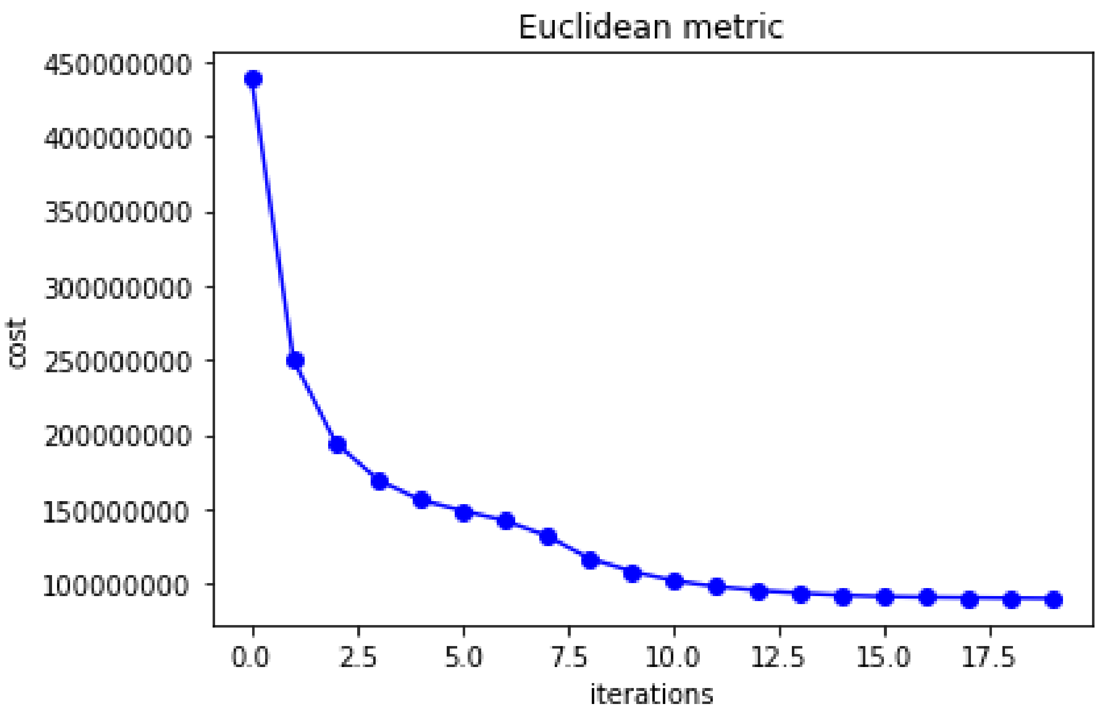
[12]:
```python
iter_costs = kmeans_cost(ITERATIONS, points, centroids, Euclidean)

print_costs(iter_costs)
plot_costs(iter_costs, 'Euclidean metric')
```

```
0:    438747790.03
1:    249803933.63
2:    194494814.41
3:    169804841.45
4:    156295748.81
5:    149094208.11
6:    142508531.62
7:    132303869.41
```

```
8:     117170969.84
9:     108547377.18
10:     102237203.32
11:      98278015.75
12:      95630226.12
13:      93793314.05
14:      92377131.97
15:      91541606.25
16:      91045573.83
17:      90752240.10
18:      90470170.18
19:      90216416.18

Cost change: 75.26%
```



Manhattan metric

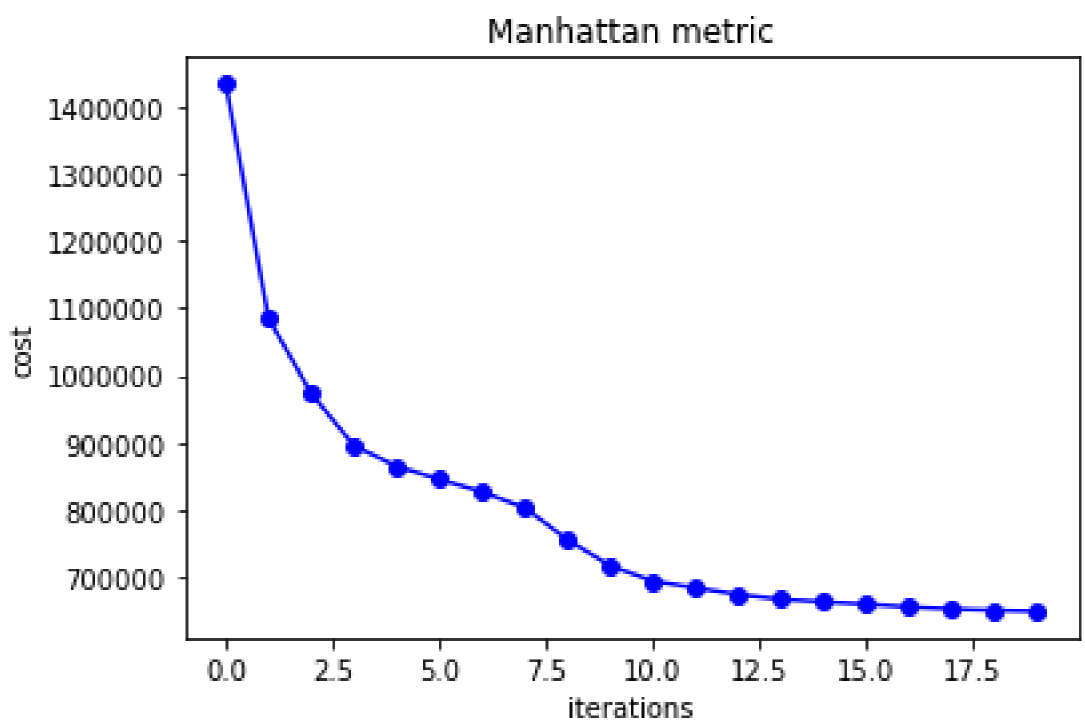
[13]:
```
iter_costs = kmeans_cost(ITERATIONS, points, centroids, Manhattan)

print_costs(iter_costs)
plot_costs(iter_costs, 'Manhattan metric')
```

```
0:       1433739.31
1:       1084488.78
2:        973431.71
```

```
3:        895934.59
4:        865128.34
5:        845846.65
6:        827219.58
7:        803590.35
8:        756039.52
9:        717332.90
10:        694587.93
11:        684444.50
12:        674574.75
13:        667409.47
14:        663556.63
15:        660162.78
16:        656041.32
17:        653036.75
18:        651112.43
19:        649689.01

Cost change: 49.97%
```



[14]:
```
sc.stop()
```