Amrita_report_kcenter

February 15, 2019

1 REPORT FOR 2.29.2 GRAPH MINING PROJECT: K CENTER CLUSTERING WITH OUTLIERS - OFFLINE AND STREAMING

(TO VIEW IMAGES OF THE GRAPHS PRODUCED CHECK /figs for STREAMING and /img for OFFLINE)

This is a brief report of the findings of the two algorithms proposed. We analyse them by various parameters. However, since the streaming algorithm was not parallelised, the best approximation we received was of 16 in case of the streaming algorithm. In future, with the parallelisation, it can be enhanced to upto almost 4, provided we have multiple instances running in parallel.

We first analyse the offline algorithms for various factors. Note that the computer on which the program was run was not efficient enough for full perusal of the 1 million tweets, and hence, most benchmarkings (for k vs e(epsilon) vs outlier count) are for value 10000. And the maximum value of the perusal is 100000 tweets.

```
In [3]: from copy import deepcopy
        import numpy as np
        import pandas as pd
        from matplotlib import pyplot as plt
        plt.rcParams['figure.figsize'] = (16, 9)
        plt.style.use('ggplot')

        # Importing the dataset
        data = pd.read_csv('benchmarking/outlier_benchmarking.csv')
        print("Input Data and Shape")
        print(data.shape)
        data.head()
        data.tail()
```

```
Input Data and Shape
(10, 6)
```

Out[3]:

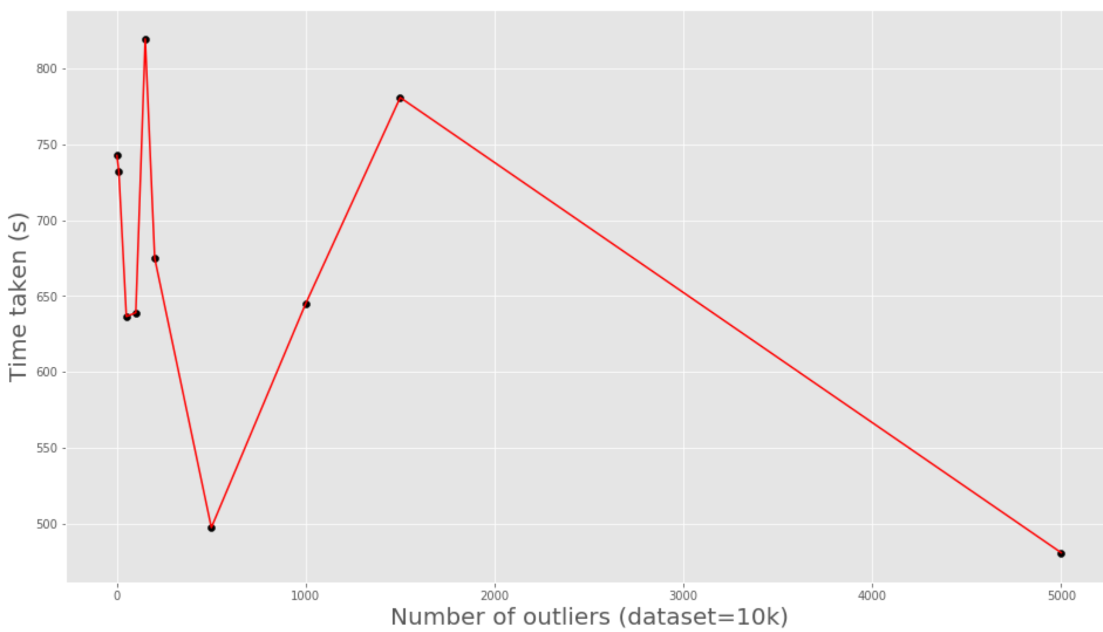
	k	e	s	z	r	t
5	4	0.5	10000	200	25.628906	674.980537
6	4	0.5	10000	500	17.085938	497.231371
7	4	0.5	10000	1000	17.085938	645.101280

```
8  4  0.5  10000  1500  17.085938  780.952300
9  4  0.5  10000  5000   5.062500  480.653113
```

The following graph shows the variation of the time taken with the number of outliers. Observe that the performance does not increase too much with an increase in outliers (until the increase is significant). This is because the data in the dataset is well spaced, and a small change in outliers does not affect the radius of the clusters.

```
In [4]: f1 = data['z'].values
        f2 = data['r'].values
        f3 = data['t'].values
        X = np.array(list(zip(f1, f2)))
        plt.plot(f1, f3, c='red')
        plt.scatter(f1, f3, c='black')
        plt.xlabel('Number of outliers (dataset=10k)', fontsize=20)
        plt.ylabel('Time taken (s)', fontsize=20)
```

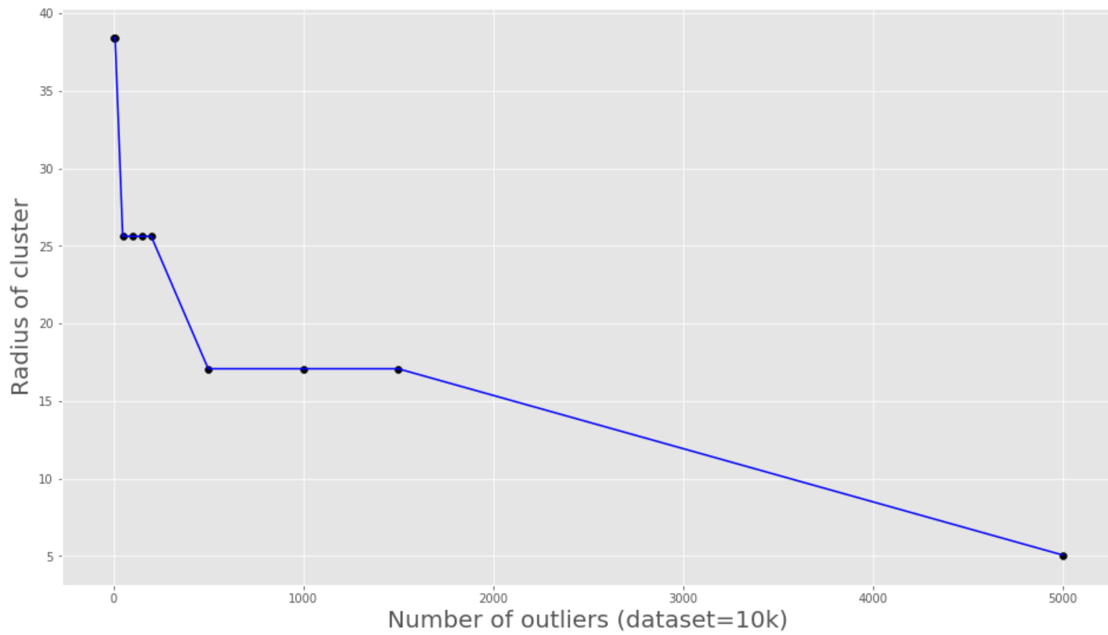
```
Out[4]: Text(0,0.5,u'Time taken (s)')
```



The plotting of the radius vs the number of outliers is once again seen in the graph below. As we can see, the radius does not improve for small changes in the number of outliers.

```
In [5]: plt.plot(f1, f2, c='blue')
        plt.scatter(f1, f2, c='black')
        plt.xlabel('Number of outliers (dataset=10k)', fontsize=20)
        plt.ylabel('Radius of cluster', fontsize=20)
```

```
Out[5]: Text(0,0.5,u'Radius of cluster')
```



We now look at how the radius and time vary when we increase the size of the dataset. To keep the results standard, we assume a general outlier barrier of 5% of the data and keep the e value at 0.5 and the number of clusters as 4.

```
In [6]: data_size = pd.read_csv('benchmarking/window_benchmarking.csv')
        print("Input Data and Shape")
        print(data_size.shape)
        data_size.head()
        data_size.tail()
```

```
Input Data and Shape
(6, 6)
```

Out[6]:

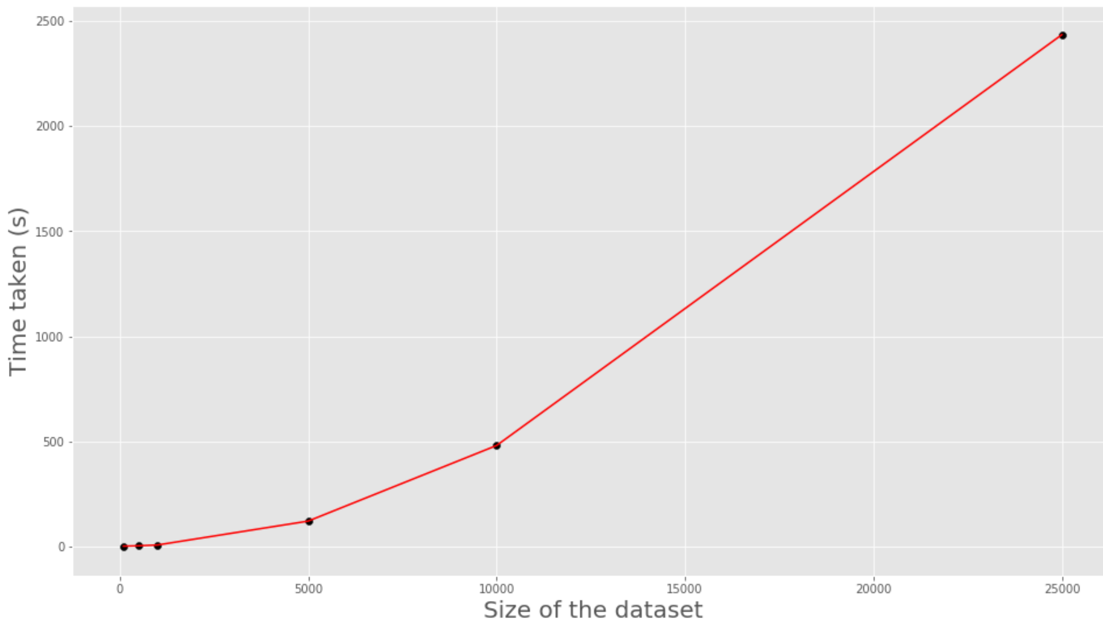
	k	e	s	z	r	t
1	4	0.5	500	25	25.628906	3.622947
2	4	0.5	1000	50	17.085938	7.373335
3	4	0.5	5000	250	17.085938	121.070914
4	4	0.5	10000	500	17.085938	480.777112
5	4	0.5	25000	1250	17.085938	2435.555415

As expected, the time is exponentially increasing with size of the input.

```
In [7]: w1 = data_size['s'].values
        w2 = data_size['r'].values
        w3 = data_size['t'].values
        X = np.array(list(zip(w1, w2)))
        plt.plot(w1, w3, c='red')
```

```
plt.scatter(w1, w3, c='black')
plt.xlabel('Size of the dataset', fontsize=20)
plt.ylabel('Time taken (s)', fontsize=20)
```

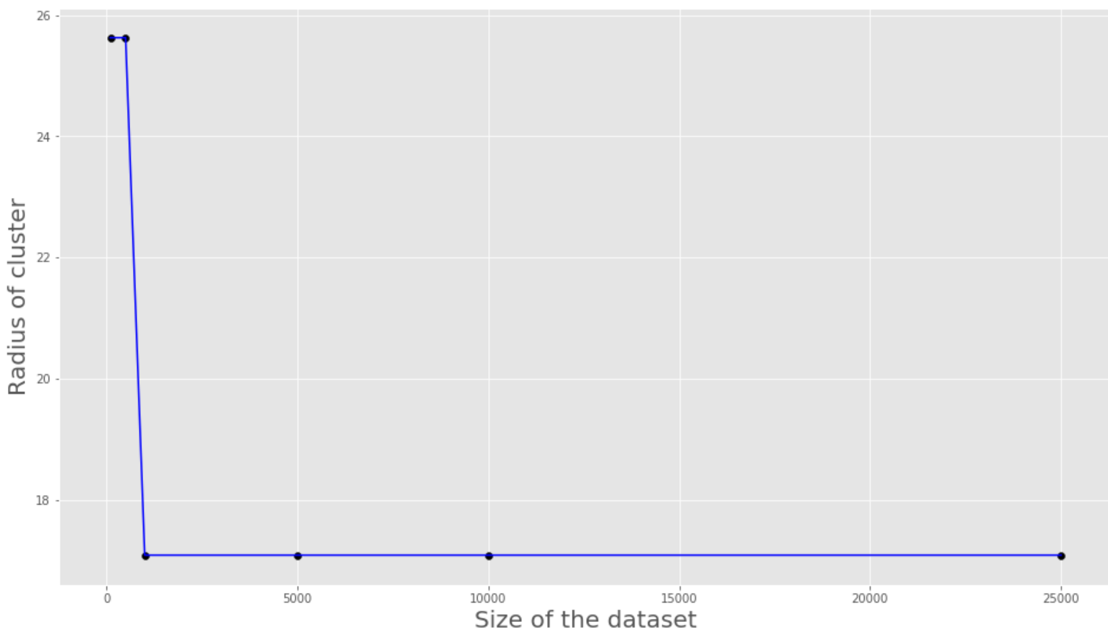
Out[7]: Text(0,0.5,u'Time taken (s)')



Once again, since the data is almost consistent, it seems like the radius remains unchanged with increasing amount of data. However, like expected, the outlier percentage is more observed when the data is lesser.

```
In [8]: plt.plot(w1, w2, c='blue')
        plt.scatter(w1, w2, c='black')
        plt.xlabel('Size of the dataset', fontsize=20)
        plt.ylabel('Radius of cluster', fontsize=20)
```

Out[8]: Text(0,0.5,u'Radius of cluster')



We now compare the streaming algorithm with the offline one.

```
In [9]: data_size_stream = pd.read_csv('benchmarking/window_benchmarking_streaming.csv')
        print("Input Data and Shape")
        print(data_size_stream.shape)
        data_size_stream.head()
        data_size_stream.tail()
```

```
Input Data and Shape
(6, 8)
```

Out[9]:

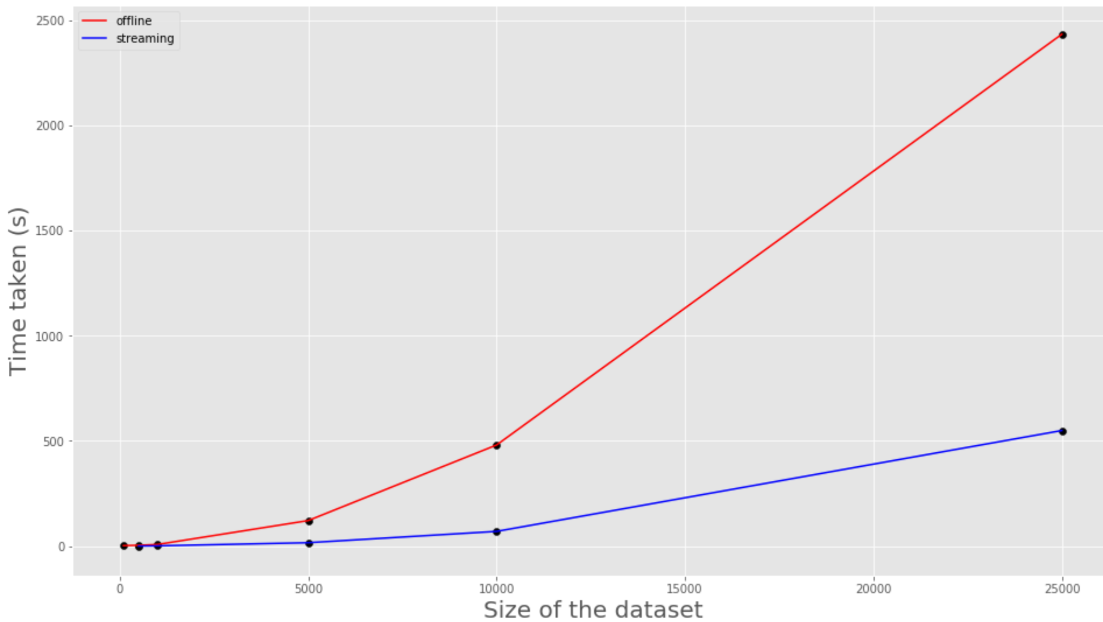
	k	z	alpha	beta	n	r	t	size
1	4	25	4.0	8	16	6.311266	0.096360	500.0
2	4	50	4.0	8	16	10.240000	0.866252	1000.0
3	4	250	4.0	8	16	5.290044	15.848242	5000.0
4	4	500	4.0	8	16	7.189937	69.689951	10000.0
5	4	1250	4.0	8	16	6.385821	549.618785	25000.0

```
In [10]: w1_s = data_size_stream['size'].values
         w2_s = data_size_stream['r'].values
         w3_s = data_size_stream['t'].values
```

```
In [11]: line_1, =plt.plot(w1, w3, c='red', label='offline')
         line_2, =plt.plot(w1_s, w3_s, c='blue', label='streaming')
         plt.scatter(w1, w3, c='black')
         plt.scatter(w1_s, w3_s, c='black')
         plt.xlabel('Size of the dataset', fontsize=20)
```

```
plt.ylabel('Time taken (s)', fontsize=20)
plt.legend(handles=[line_1, line_2])
```

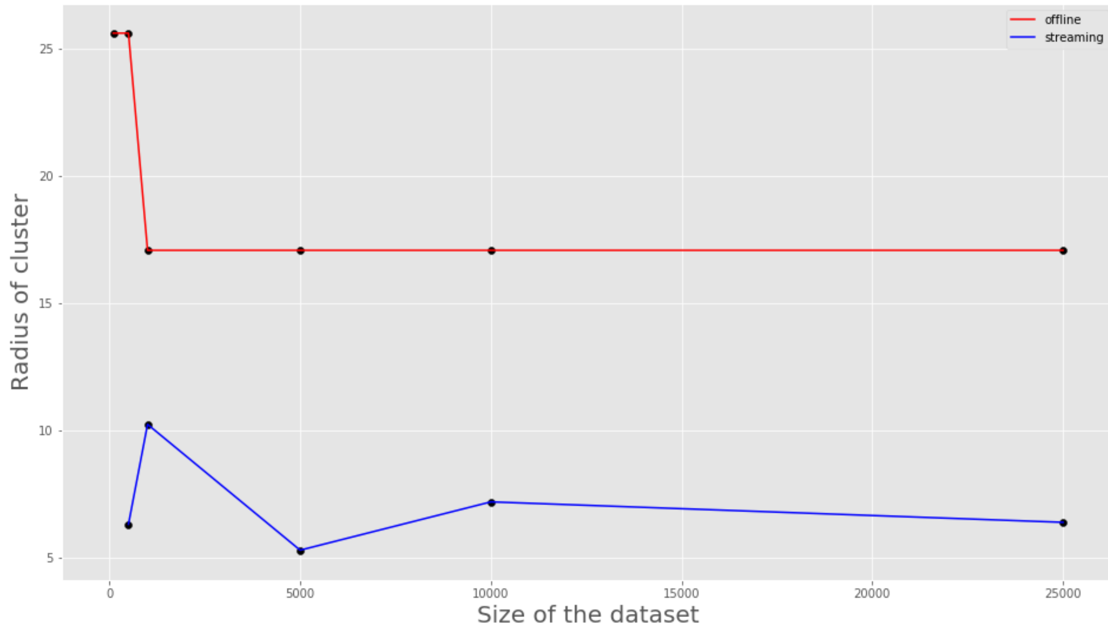
Out[11]: <matplotlib.legend.Legend at 0x11a891f10>



As we see below, the radius in the streaming case is much smaller, because the approximation is a lot worse. In other words, we get a radius which is 4 times less accurate than the offline case on an average.

In [12]:
```
line_3, =plt.plot(w1, w2, c='red', label='offline')
line_4, =plt.plot(w1_s, w2_s, c='blue', label='streaming')
plt.scatter(w1, w2, c='black')
plt.scatter(w1_s, w2_s, c='black')
plt.xlabel('Size of the dataset', fontsize=20)
plt.ylabel('Radius of cluster', fontsize=20)
plt.legend(handles=[line_3, line_4])
```

Out[12]: <matplotlib.legend.Legend at 0x11a65b950>



However, it is worthwhile to note that the entire dataset can be efficiently perused in case of the streaming case PROVIDED the number of outliers are few (because each window is of size k*outliers). NOTE THAT the size of the dataset in the next two graphs is 1 million, i.e the entire data.

```
In [20]: data_outlier_stream = pd.read_csv('benchmarking/outlier_benchmarking_stream.csv')
         print("Input Data and Shape")
         print(data_outlier_stream.shape)
         data_outlier_stream.head()
         data_outlier_stream.tail()
```

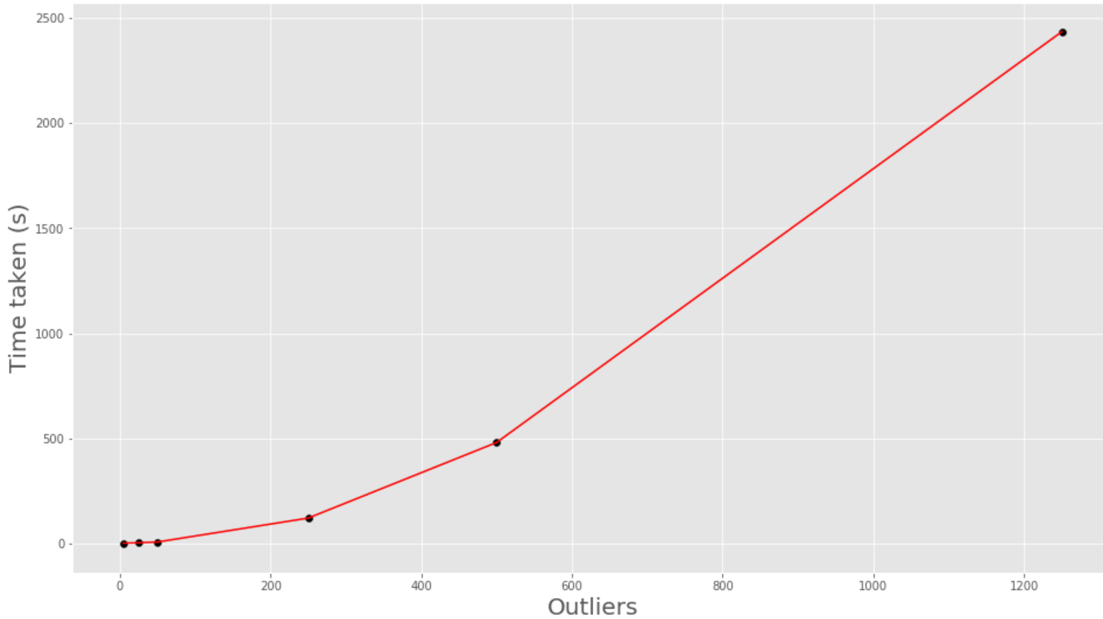
```
Input Data and Shape
(5, 7)
```

Out[20]:

	k	z	alpha	beta	n	r	t
0	4	1	4.0	8	16	22.555461	0.067423
1	4	50	4.0	8	16	5.566289	4.246688
2	4	100	4.0	8	16	11.174795	7.153670
3	4	200	4.0	8	16	6.469594	10.554042
4	4	500	4.0	8	16	7.244098	102.378632

```
In [21]: o1 = data_size['z'].values
         o2 = data_size['r'].values
         o3 = data_size['t'].values
         plt.plot(o1, o3, c='red')
         plt.scatter(o1, o3, c='black')
         plt.xlabel('Outliers', fontsize=20)
         plt.ylabel('Time taken (s)', fontsize=20)
```

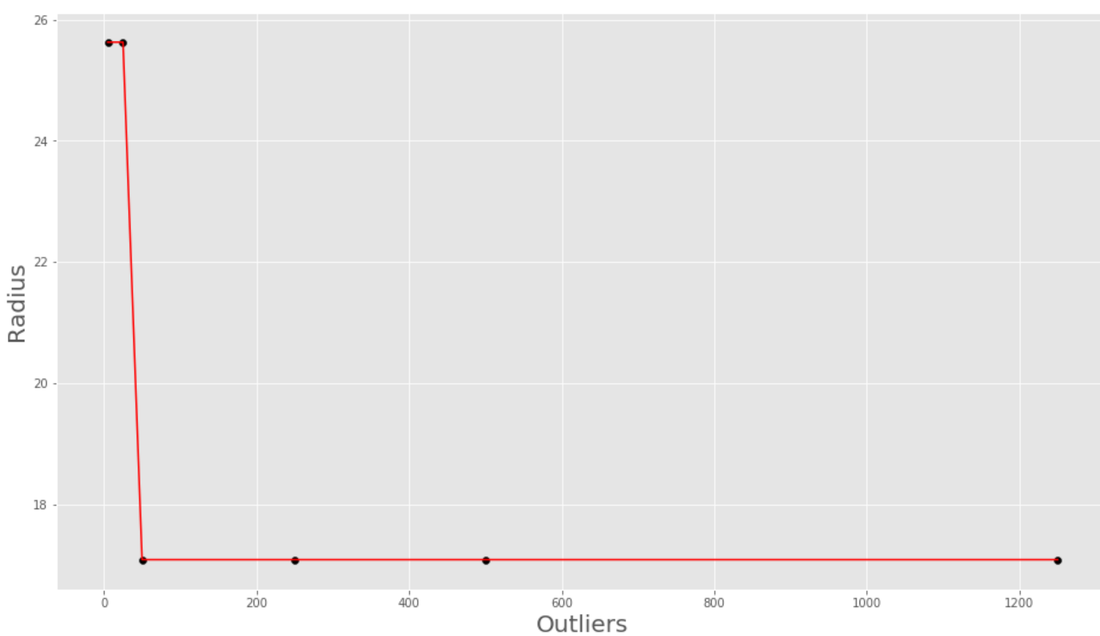
Out[21]: Text(0,0.5,u'Time taken (s)')



Again, from the data we see that the radius doesn't change much with the increase in outliers

```
In [22]: plt.plot(o1, o2, c='red')
         plt.scatter(o1, o2, c='black')
         plt.xlabel('Outliers', fontsize=20)
         plt.ylabel('Radius', fontsize=20)
```

Out[22]: Text(0,0.5,u'Radius')

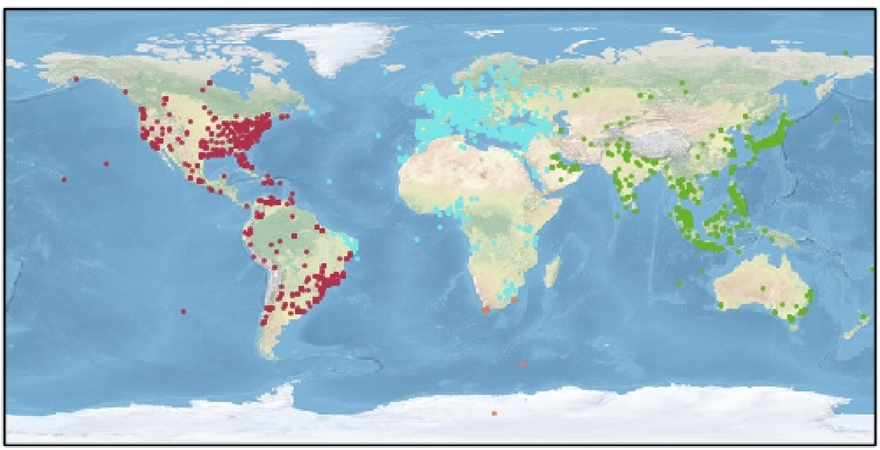


QUALITY OF THE SOLUTION (data size 5000) k = 4

Following is the offline clustering for values k=4,e=0.5,dataset=5000,outlier_count(z)=100. The value we obtain are radius = 25.62890625 and the time taken was 157.971886566

```
In [14]: from IPython.display import Image
         Image(filename='img/report_img.jpg')
```

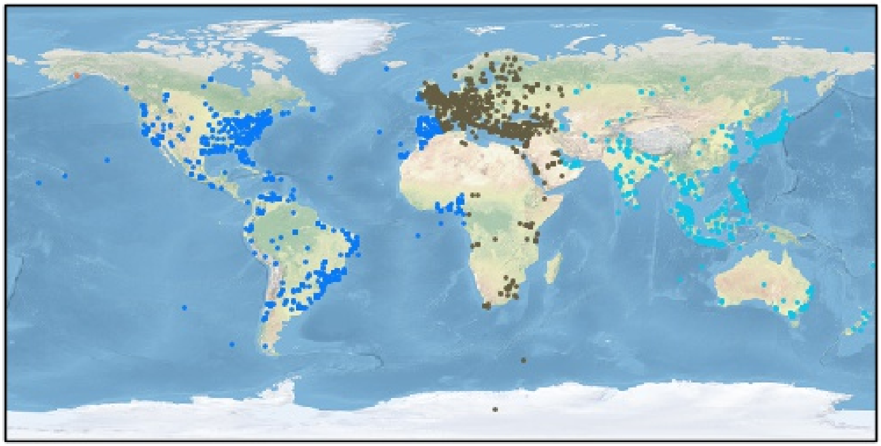
Out[14]:



Following is the offline clustering for values k=4,dataset=5000,outlier_count(z)=100 (alpha, beta, n values as 4,8,16). The value we obtain are radius = 5.6466356 and the time taken was 3.4634563199999997

```
In [23]: Image(filename='figs/report_img_streaming.jpg')
```

Out[23]:



As we see here, the clustering in case of streaming is much more coarse and some tweets from Africa/Europe are clustered along with tweets from the Americas. In case of the other image, it is fairly well separated.

In conclusion, while the streaming algorithm scales better, the clustering is more fine in case of the offline algorithm. Like the paper on streaming suggests, it is possible to run the streaming algorithm in parallel multiple times in order to achieve an approximation factor of almost the same, however, that has not been studied in this project.