Assignment 1

Part 1. Clustering: the baseline

Prepare data

Original data set is 1 million. The following code is used to extract 100K from the total sample.

```
with open('data/tweets_1M.json','r') as f:
    tweets = json.load(f)

X = np.array([[tweets[x]['lat'],tweets[x]['lng']] for x in range(0, len(tweets))])
sample = 100000
total = len(X)
X = X[0::int(total/sample)]
```

Reference time of clustering of 100K samples into k=100 clusters using k-means

```
n = 100
k_means = KMeans(init='k-means++', n_clusters=n, n_init=10)
t_km = time.time()
k_means.fit(X)
t_fin_km = time.time() - t_km
print (t_fin_km)
```

Reference time is 141 seconds.

Reference time of clustering of 100K samples into k=100 clusters with mini-batch k-means.

To select a proprapriate batch_size, the following code is used to roughly look at the relationship between `batch size` and processing time.

```
def frange(start, stop, step):
    i = start
    while i < stop:
        yield i
        i += step

for perc in frange(0.01,0.11,0.01):
    batch_size=int(len(X)*perc)
    mbk = MiniBatchKMeans(init='k-means++', n_clusters=100, batch_size=batch_size,
                          n_init=10, max_no_improvement=10, verbose=0)
    t0 = time.time()
    mbk.fit(X)
    t_mini_batch = time.time() - t0
```

Simplified results follow:

Percentage	`batch_size`	Seconds to run 100 clusters
1%	1000	0.60
5%	5000	1.65
10%	10000	3.04

Maximum number of clusters `k_max` that the implementation of k-means can handle

Via some tests, n = 370 seems to be a reasonable starting number. The following code is implemented to get `k_max`. Processing time threshold is set to approximately 60 seconds.

```
n = 35
t_fin_km = 0
while t_fin_km <= 60:
    print ('testing n equal to ' + str(n))
    ## initialize with K-means++, a good way of speeding up convergence
    k_means = KMeans(init='k-means++', n_clusters=n, n_init=10)
    ## record the current time
    t_km = time.time()
    # start clustering!
    k_means.fit(X)
    ## get the time to finish clustering
    t_fin_km = time.time() - t_km
    print (t_fin_km)
    n += 1
```

`k_max` = 36

Maximum number of clusters k_max that the implementation of minibatch k-means can handle

Three `batch_size` (1000, 5000, 10000) were run to see how `k_max` changes. `perc` and `n` in the following code can vary as needed.

```
perc = 0.01
batch_size=int(len(X)*perc)
n = 2500
t_mini_batch = 0
while t_mini_batch <= 60:
    print ('testing n equal to ' + str(n))
    mbk = MiniBatchKMeans(init='k-means++', n_clusters=n, batch_size=batch_size,
                          n_init=10, max_no_improvement=10, verbose=0)
    t0 = time.time()
    mbk.fit(X)
    t_mini_batch = time.time() - t0
    n += 50
    print (t_mini_batch)
```

Results follow:

Percentage	batch_size	k_max
1%	1000	3950
5%	5000	365
10%	10000	175

Find eps__100 that resultes in 100 clusters with MinPts =100 as well as the corresponding processing time.

```
eps = 0.05
n_clusters_ = 0
while n_clusters_ <= 100:

    print (eps)

    t_db = time.time()
    db = DBSCAN(eps=eps, min_samples=100).fit(X)
    t_fin_db = time.time() - t_db

    #array of numbers, one number represents one cluster
    db_labels = db.labels_
    # minus if there are unclustered noises
    n_clusters_ = len(set(db_labels)) - (1 if -1 in db_labels else 0)

    eps += 0.01
    print (t_fin_db, n_clusters_)
```

eps__100 = 0.001 (0.002 and 0.003 also produces 100 clusters)

Part 2. Clustering: scalability

KMeans Scalability and Estimation

Essentially we are looking at how processing time increases as 1) number of clusters increases and 2) sample size increases.

The following experiments with the relationship between number of clusters and processing time.

Given the 100K sample, computational time is estimated to be 160 seconds when number of clusters is 100, which is proved to be relatively accruate given the previous test (reference time around 141 seconds when number of clusters is 100).

The first figure in the four images below shows the relationship between processing time and sample size using k_means. Since `n_clusters` is fixed to 100 in this experiment. We don't have to scale up along this dimension. We only consider how sample size increases processing time. In order to roughly estimate how long 100 million samples will take to generate 100 clusters, three fitted lines were added shown in the rest three figures.

The first one assumes the simplest linear relation ship with polynomial degree of 1 and the computational time is estimated to be 143 seconds as sample size approaches 100 million. It is underestimated because, as shown in previously, 100K samples with 100 clusters require 141 seconds processing time already.

The bottom left figure uses linear relationship with polynomial degree of 2 and the predicted processing time is 446 seconds, which seems more reasonable.

The last figure shows how exponential relationship fits with the data. It looks similar on the graph but the estimate time is close to 3018 seconds.

A table is built to summarize the information:

Sample Size	`n_clusters`	Computational Time (seconds)
100	2	3950
	36 (`k_max`)	
	100	
100000	2	365
	36 (`k_max`)	
	100	
1000000	2	175
	36 (`k_max`)	175
	100	

MiniBatch KMeans Scalability and Estimation

2.1.b

Computational time as a function of `n_clusters` (consider the range of 2 to the `k_max`) with MiniBatchKMeans

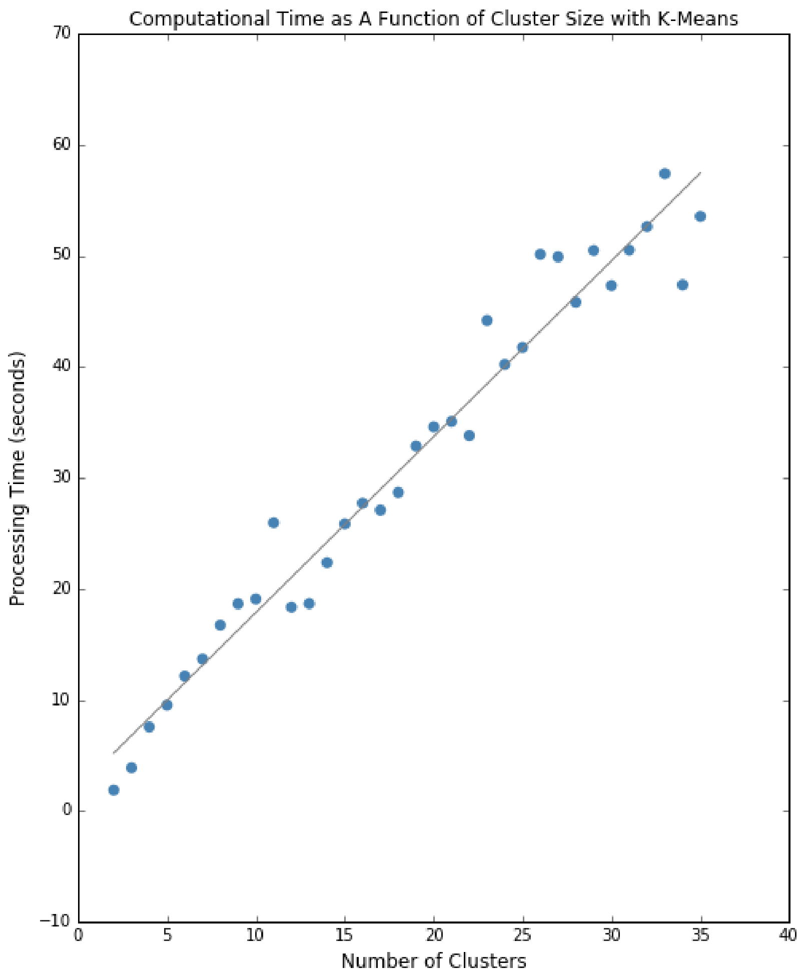

Figure 1: Computational time as a function of `n_clusters` (consider the range of 2 to the `k_max`) with k_means

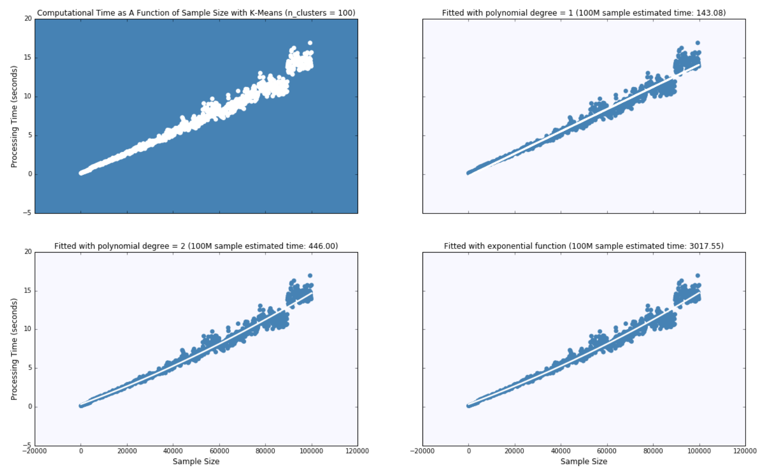

Figure 2: Computational time as a function of sample size for a fixed k=100 with k_means

`batch_size` here is 1% of 100K sample, which is 1000. If sample size increases 100 million, computational time is estimated to be about 3.6 seconds. Given the previous experiment, we know that as batch_size increases, the computational time will increase towards the time using k-means.

[Computational time as a function of `n_clusters` (consider the range of 2 to the `k_max`) with MiniBatchKMeans][Figure4]

[Figure4]

Part 3. Clustering: 1 million samples problem

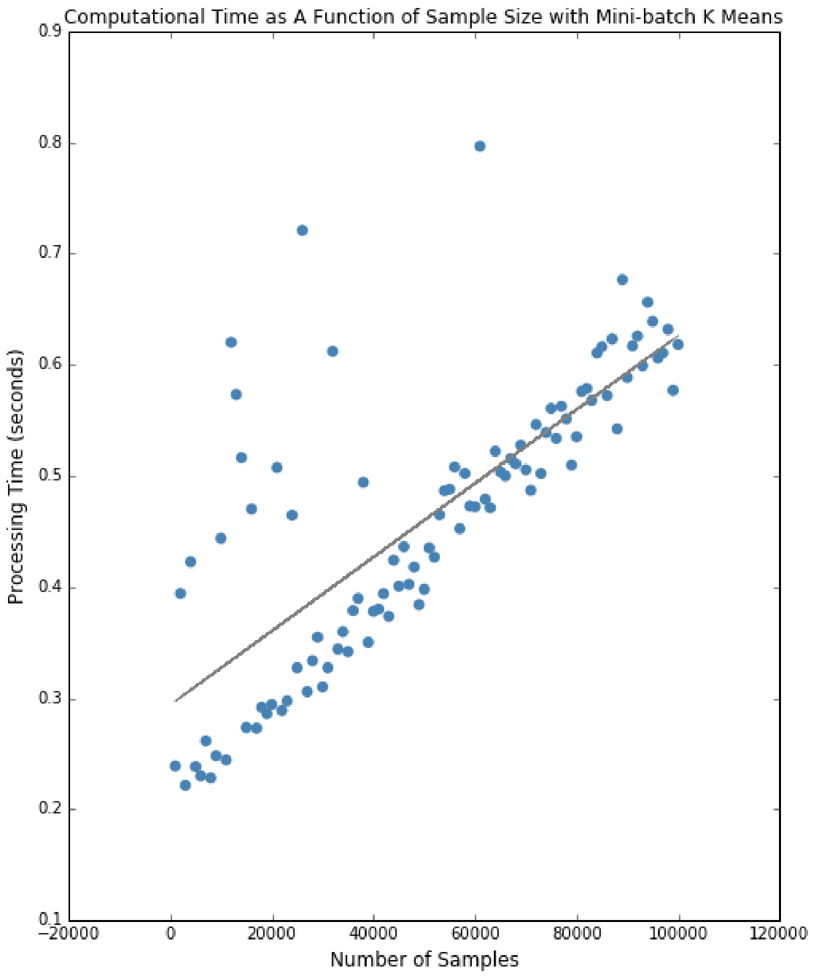

Figure 3: Computational time as a function of sample size for a fixed k=100 with MiniBatchKMeans