Tweet Clustering

Exploratory data analysis of one million tweets using clustering techniques in scikit-learn.
Civil & Environmental Engineering 263n: Scalable Spatial Analytics at UC-Berkeley, Fall 2016
By Paul Sohn, September 14, 2016

Part 1: Baseline Results for Different Clustering Algorithms

We will test three different clustering algorithms through the scikit-learn packing in Python:

- K-means
- MiniBatch K-means
- DBSCAN

The first step is to understand the processing limits of each algorithm. The dataset we are using includes 1 million tweets (mostly) from the Bay Area. We will start with a random subset of 100,000 tweets to test the algorithms.

K-means

We are trying to find the reference time of clustering of 100K samples into k=100 clusters with k-means. The basic python code snippet involves instatiating a KMeans object, fitting to a numpy array (`data`), and printing the time taken:

```
k_means = KMeans(n_clusters=100, init='k-means++', n_init=10)

t0 = time.time()
k_means.fit(data)
print time.time() - t0
```

Time to cluster 100,000 tweets into 100 clusters using K-means: **20.7 seconds**

MiniBatch K-means

As above, we are trying to find the reference time of clustering of 100K samples into k=100 clusters with MiniBatch k-means. The python code is very similar, except one has to select a batch size. We can simply test several arbitraty batch size values as below:

```
for batch_size in [5, 10, 20, 50, 100, 500, 1000]:

    mb = MiniBatchKMeans(n_clusters=100, init='k-means++', n_init=10, batch_size=batch_size)

    t0 = time.time()
    mb.fit(data)
    print time.time() - t0
```

Batch Size	Time to generate 100 clusters (seconds)
5	5.29
10	2.84
20	2.69
50	1.48
100	0.74

Batch Size	Time to generate 100 clusters (seconds)
500	0.63
1000	0.70