bolliger_CE263N_assignment_1

September 24, 2015

Initial Setup

In [1]:
```
# set random seed
seed = 1234

# imports
from os import chdir as cd
from IPython.display import HTML,Latex
import pandas as pd
from sklearn.cluster import KMeans, MiniBatchKMeans, DBSCAN
import time
from pyproj import Proj
import matplotlib.pyplot as plt
import numpy as np

# set filepaths
data_loc = u"""/Users/ianbolliger/Box Sync/grad_school/courses/2015_fall\
/spatial_analytics/assignments/assignment_1/tweets_1M.json"""

# set display properties
float_frmt = lambda x: '{:,.2f}'.format(x)
plt.style.use('ggplot')
plt.rcParams.update({'font.size': 12})
plt.rcParams.update({'figure.autolayout':True})
```

In [2]:
```
# load data
tweets = pd.read_json(data_loc).set_index('id')

# rescale lat/long into meters
myProj = Proj("+proj=utm +zone=10 +ellps=WGS84 +datum=WGS84")
UTMx,UTMy = myProj(tweets['lng'].values,tweets['lat'].values)
min_x = min(UTMx)
m_E = [j-min_x for j in UTMx]
min_y = min(UTMy)
m_N = [j-min_y for j in UTMy]
tweets['m_N'] = m_N
tweets['m_E'] = m_E

# drop outliers from the other side of the world...
tweets = tweets[tweets['m_N'] < 1E29]

# # convert timeStamp to pd.datetime dtypes
# tweets_pd['timeStamp'] = pd.to_datetime(tweets_pd['timeStamp'])
```

```
# select only coordinates
coords = tweets[['m_N','m_E']]
n_max = coords.shape[0]

# select 100k subset of tweets
n_subset = 100000
subset = coords.sample(n=n_subset, random_state=seed)
```

Part 1. Clustering: the baseline

1.1 & 1.2: k-means and MiniBatch k-means

In [ ]:
```
batch_size = 10000
# k_values = [1, 2, 4, 8, 16]
k_values = [int(i) for i in logspace(log10(2),log10(n_subset))]
# k_values = [1, 2, 4, 8, 16, 32, 64, 100, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768,

times = pd.DataFrame(index=[],columns=['k-means','MiniBatch k-means'])
times.index.name = 'k'

for k in k_values:

    # vanilla k-means
    k_means = KMeans(init='k-means++', n_clusters=k, n_init=1,n_jobs=1,random_state=seed)
    t0 = time.time()
    k_means.fit(subset)
    t = time.time() - t0
    times.loc[k,'k-means'] = t

    # MB k-menas
    MBk_means = MiniBatchKMeans(init='k-means++', n_clusters=k, n_init=1,random_state=seed,batc
    t0 = time.time()
    MBk_means.fit(subset)
    t = time.time() - t0
    times.loc[k,'MiniBatch k-means'] = t

    print k

times.to_pickle('times')
```

In [ ]:
```
# print latex table for including in HW writeup
Latex(times.loc[[1,100,1024, 10539,89794],:].to_latex(formatters={col:float_frmt for col in time
```

NOTE: 68697 was the highest number of clusters at which all algorithms converged within a reasonable timeframe (under 1.5 hours)

In [ ]:
```
# combining new and old timing tables when I ran again with different list of "k" values
# times_old = pd.read_pickle('times_2')
# times_n = times_2.join(times_old,how='outer',rsuffix='o')
# for i in ['k-means','MiniBatch k-means','DBSCAN','DBSCAN_n_clusters']:
#     times_n[i].fillna(times_n[i+'o'],inplace=True)
# times_n = times_n[['k-means','MiniBatch k-means','DBSCAN','DBSCAN_n_clusters']]
# times_2 = times_n
# times_2 = times_2.iloc[:-1,:]
# times_2.to_pickle('times_2')
```

1.3: DBSCAN

```
In [ ]: min_samples=100
        max_dist_poss = sqrt(subset['m_N'].max()**2 + subset['m_E'].max()**2)
        eps = range(489,494)+range(1002,1022)
        times_db = pd.DataFrame(index=[],columns=['time','n_clusters'])
        times_db.index.name = 'epsilon'
        for e in eps:
            db = DBSCAN(eps=e, min_samples=min_samples)
            t0 = time.time()
            db.fit(subset)
            t = time.time() - t0

            # Number of clusters in labels, ignoring noise if present.
            n_clusters = len(set(db.labels_)) - (1 if -1 in db.labels_ else 0)

            times_db.loc[str(e),'time'] = t
            times_db.loc[str(e),'n_clusters'] = n_clusters
            print e,t,n_clusters
```

Part 2. Clustering: scalability

2.1 & 2.2

```
In [ ]: sample_size = [int(i) for i in logspace(2,log10(n_max))]
        # sample_size = [100, 200, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000, coords.shape[0]]
        e = 500
        min_samples=100

        times_2 = pd.DataFrame(index=[],columns=['k-means','MiniBatch k-means','DBSCAN'])
        times_2.index.name = 'n'

        for n in sample_size:

            # choose subset with different random state for each iteration (but same across runs of the
            data = coords.sample(n=n,random_state=n)

            # vanilla k-means
            k_means = KMeans(init='k-means++', n_clusters=100, n_init=1,n_jobs=1,random_state=seed)
            t0 = time.time()
            k_means.fit(data)
            t = time.time() - t0
            times_2.loc[n,'k-means'] = t

            # MB k-means (batch size = 10% of sample size)
            MBk_means = MiniBatchKMeans(init='k-means++', n_clusters=100, n_init=1,random_state=seed,ba
            t0 = time.time()
            MBk_means.fit(data)
            t = time.time() - t0
            times_2.loc[n,'MiniBatch k-means'] = t

            # DBSCAN
            db = DBSCAN(eps=e, min_samples=min_samples)
            t0 = time.time()
            db.fit(data)
```

```
        t = time.time() - t0

        # Number of clusters in labels, ignoring noise if present.
        n_clusters = len(set(db.labels_)) - (1 if -1 in db.labels_ else 0)

        times_2.loc[n,'DBSCAN'] = t
        times_2.loc[n,'DBSCAN_n_clusters'] = n_clusters

        print n

    times_2.to_pickle('times_2')
```

In [ ]:
```
# print latex table for including in HW writeup
Latex(times_2.iloc[-1:,:].to_latex(formatters={col:float_frmt for col in times_2.columns}))
```

In [ ]:
```
# plot time as func of sample size
fig,axes = subplots(1,2,figsize=(16,8))
times_only = times_2[['k-means','MiniBatch k-means','DBSCAN']]
times_only.loc[:100000,:].plot(ax=axes[0])
times_only.plot(logy=False,ax=axes[1])
suptitle('Computational Time for each of the 3 Algorithms',fontsize=18)
for i in axes:
    i.set_ylabel('Time (s)')
    i.set_xlabel('Sample Size')
axes[0].set_title('Sample Size 100:100,000')
axes[0].legend(['k-means (k=100)','MiniBatch k-means (k=100)','DBSCAN ($\epsilon$ = 500m)'],loc=
axes[1].set_title('Sample Size 100:1M')
axes[1].legend(['k-means (k=100)','MiniBatch k-means (k=100)','DBSCAN ($\epsilon$ = 500m)'],loc=
axes[0].text(0.05, 0.95, 'A', transform=axes[0].transAxes, fontsize=16, fontweight='bold', va='
axes[1].text(0.05, 0.95, 'B', transform=axes[1].transAxes, fontsize=16, fontweight='bold', va='
fig.savefig('comptime.pdf')
```

In [ ]:
```
# plot time as func of requested clusters
plt.figure(figsize=(8,8))
ax = times.plot()
ax.set_title('Computational Time for the 2 k-means Algorithms',fontsize=14)
ax.set_ylabel('Time (s)')
ax.legend(loc='center left')
for tick in ax.get_xticklabels():
    tick.set_rotation(45)
ax.text(0.05, 0.95, 'C', transform=ax.transAxes, fontsize=16, fontweight='bold', va='top')
savefig('comptime_k.pdf', bbox_inches='tight')
```

Part 3. Clustering

In [ ]:
```
### 2-layer clustering

# MB k-means parameters
batch_size = 10000

# DBSCAN parameters
e = 100 # 100 meter epsilon
min_samples = 100

# try analysis for many 1st-stage cluster sizes
```

```
k_step1 = [20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2]

# initialize DBSCAN object
db = DBSCAN(eps=e, min_samples=min_samples)

# copy coordinates so that we can assign clusters in new dataframe
data = coords.copy()
data['cluster2'] = np.nan

# create timing and number of cluster metric
times_3 = pd.DataFrame(index=[],columns=['time','n_clusters'])
times_3.index.name = 'k_step1'

for k in k_step1:
    MBk_means = MiniBatchKMeans(init='k-means++', n_clusters=k , random_state=seed, batch_size=

    t0 = time.time()
    MBk_means.fit(coords)
    data.loc[:,'cluster1'] = pd.Series(MBk_means.labels_, index = data.index)

    # run DBSCAN on each cluster
    for c in range(k):
        ss = coords[data['cluster1']==c]
        db.fit(ss)
        ss = pd.DataFrame({'cluster2':db.labels_}, index = ss.index)
        data = data.combine_first(ss)

    t = time.time() - t0

    # no. of final clusters
    clust_num = data[data['cluster2'] != -1]
    clust_num = clust_num.groupby(['cluster1','cluster2']).count()
    # drop clusters that DBSCAN produced w/ less than min_samples points (due to unfortunate be
    n_clusters = clust_num[clust_num['m_N']>= min_samples].shape[0]


    # add timing and cluster number data to timing dataframe
    times_3.loc[k,'time'] = t
    times_3.loc[k,'n_clusters'] = n_clusters

times_3 = times_3.reindex(times_3.index.sort())
times_3.to_pickle('times_3')
times_3
```

In [ ]:
```
# plot time as func of requested clusters
ax = times_3['time'].plot()
ax.set_ylabel('Time (s)')
ax.set_xlabel('Clusters Requested in Step-1 Algorithm')
savefig('2steptimes.pdf')
```

In [ ]:
```
### 3-layer clustering

# MB k-means parameters
batch_size = 10000
```

```
# DBSCAN parameters
e = 100 # 100 meter epsilon
min_samples = 100

# try analysis for many 1st-stage cluster sizes
k_step1 = [2,3,4,5,6]
k_step2 = [2,3,4,5,6]

# initialize DBSCAN object
db = DBSCAN(eps=e, min_samples=min_samples)

# copy coordinates so that we can assign clusters in new dataframe
data = coords.copy()
data['cluster2'] = np.nan
data['cluster3'] = np.nan

# create timing and number of cluster metric
times_3b = pd.DataFrame(index=pd.MultiIndex(levels=[[],[]],labels=[[],[]],names=['k1','k2']),co

for k in k_step1:
    MBk_means = MiniBatchKMeans(init='k-means++', n_clusters=k , random_state=seed, batch_size=

    t0 = time.time()
    MBk_means.fit(coords)
    data.loc[:,'cluster1'] = pd.Series(MBk_means.labels_, index = data.index)
    t1 = time.time() - t0

    for k2 in k_step2:
        t0 = time.time()
        MBk_means = MiniBatchKMeans(init='k-means++', n_clusters=k2 , random_state=seed, batch_

        for c1_label in range(k):
            ss1 = coords[data['cluster1']==c1_label]
            MBk_means.fit(ss1)
            ss1.loc[:,'cluster2'] = pd.Series(MBk_means.labels_, index = ss1.index)
            data = data.combine_first(ss1)

            # run DBSCAN on each cluster
            for c in range(k2):
                ss2 = ss1[ss1['cluster2']==c]
                db.fit(ss2)
                ss2 = pd.DataFrame({'cluster3':db.labels_}, index = ss2.index)
                data = data.combine_first(ss2)

        t = time.time() - t0

        # no. of final clusters
        clust_num = data[(data['cluster2'] != -1) & (data['cluster3'] != -1)]
        clust_num = clust_num.groupby(['cluster1','cluster2','cluster3']).count()
        # drop clusters that DBSCAN produced w/ less than min_samples points (due to unfortunat
        n_clusters = clust_num[clust_num['m_N']>= min_samples].shape[0]

        # add timing and cluster number data to timing dataframe
        times_3b.loc[(k,k2),'time'] = t + t1
```

```
            times_3b.loc[(k,k2),'n_clusters'] = n_clusters

    times_3b
```

In [ ]:
```
# print latex table for including in HW writeup
Latex(times_3b.to_latex())
```

Extra Credit

In [3]:
```
import folium
```

In [ ]:
```
## recluster using optimal 2-step algorithm (with full k-means)

# parameters
k1 = 2
e = 100 # 100 meter epsilon
min_samples = 100

# initialize algorithms object
km = KMeans(init='k-means++', n_clusters=k1, random_state=seed)
db = DBSCAN(eps=e, min_samples=min_samples)

# copy coordinates so that we can assign clusters in new dataframe
data = coords.copy()
data['cluster2'] = np.nan

# run step 1
km.fit(coords)
data.loc[:,'cluster1'] = pd.Series(km.labels_, index = data.index)

# run DBSCAN on each cluster
for c in range(k1):
    ss = coords[data['cluster1']==c]
    db.fit(ss)
    ss = pd.DataFrame({'cluster2':db.labels_}, index = ss.index)
    data = data.combine_first(ss)
```

In [5]:
```
# select points from most populous cluster
clustered_pts = data[data['cluster2'] != -1]
sorted_clusters = clustered_pts.groupby(['cluster1','cluster2']).count().sort('m_N',ascending=Fa
biggest_cluster = sorted_clusters.index[0]
pts_in_cluster = data[(data['cluster1'] == biggest_cluster[0]) & (data['cluster2'] == biggest_cl
all_data = pts_in_cluster.join(tweets[['lat','lng','text','timeStamp']],how='left')[['lat','lng

# get rounded lat/long b/c folium can't handle full length
lat = [float(i.round(2)) for i in all_data['lat']]
lng = [float(i.round(2)) for i in all_data['lng']]

#find cluster center
center = all_data[['lat','lng']].mean()
```

In [6]:
```
# use folium to map

def inline_map(map):
```

```
    """
    Embeds the HTML source of the map directly into the IPython notebook.

    This method will not work if the map depends on any files (json data). Also this uses
    the HTML5 srcdoc attribute, which may not be supported in all browsers.
    """
    map._build_map()
    return HTML('<iframe srcdoc="{srcdoc}" style="width: 100%; height: 510px; border: none"></i

map_osm = folium.Map(location=list(center),zoom_start=12)
for i in range(0,len(lat),100):
    marker_data = all_data.iloc[i,:]
    map_osm.simple_marker([marker_data['lat'],marker_data['lng']])
inline_map(map_osm)
```

Out[6]: <IPython.core.display.HTML object>

It's clear that the cluster with the most tweets belongs to downtown San Francisco. Since that area is so dense, the area still forms a cluster even though people are tweeting about a wide range of topics. A quick visual analysis of the tweet texts did not help identify the cluster other than that many were about San Francisco. To confine the tweets further (to Downtown SF), plotting on the map was necessary.

In [ ]: