Facebook-Project

October 23, 2016

1 Facebook Check-Ins

FINAL LEADERBOARD SCORE: 0.53454, Team Name: Jong Lee

```
In [21]: import numpy as np
         import pandas as pd
         from pandas import DataFrame, Series
         from sklearn import neighbors, datasets
         from sklearn.preprocessing import LabelEncoder
         from sklearn.neighbors import KNeighborsClassifier as KNN
```

```
In [34]: #Read data
         train = pd.read_csv('./data/train.csv')
         test = pd.read_csv('./data/test.csv')
```

```
In [23]: #Take a look at the datasets
         print("Training data size is" + str(train.shape))
         print("Testing data size is" + str(test.shape))

         print(train['accuracy'].describe())
         print(train.head(3))
         print(test.head(3))
```

```
Training data size is(29118021, 6)
Testing data size is(8607230, 5)
count    2.911802e+07
mean     8.284912e+01
std      1.147518e+02
min      1.000000e+00
25%      2.700000e+01
50%      6.200000e+01
75%      7.500000e+01
max      1.033000e+03
Name: accuracy, dtype: float64
   row_id       x       y  accuracy    time    place_id
0       0  0.7941  9.0809        54  470702  8523065625
1       1  5.9567  4.7968        13  186555  1757726713
```

```
2        2  8.3078  7.0407        74  322648  1137537235
   row_id       x       y  accuracy    time
0       0  0.1675  1.3608       107  930883
1       1  7.3909  2.5301        35  893017
2       2  8.0978  2.3473        62  976933
```

```
In [24]: train.describe()
```

```
Out[24]:              row_id             x             y      accuracy           tim
         count  2.911802e+07  2.911802e+07  2.911802e+07  2.911802e+07  2.911802e+0
         mean   1.455901e+07  4.999770e+00  5.001814e+00  8.284912e+01  4.170104e+0
         std    8.405649e+06  2.857601e+00  2.887505e+00  1.147518e+02  2.311761e+0
         min    0.000000e+00  0.000000e+00  0.000000e+00  1.000000e+00  1.000000e+0
         25%    7.279505e+06  2.534700e+00  2.496700e+00  2.700000e+01  2.030570e+0
         50%    1.455901e+07  5.009100e+00  4.988300e+00  6.200000e+01  4.339220e+0
         75%    2.183852e+07  7.461400e+00  7.510300e+00  7.500000e+01  6.204910e+0
         max    2.911802e+07  1.000000e+01  1.000000e+01  1.033000e+03  7.862390e+0

                   place_id
         count  2.911802e+07
         mean   5.493787e+09
         std    2.611088e+09
         min    1.000016e+09
         25%    3.222911e+09
         50%    5.518573e+09
         75%    7.764307e+09
         max    9.999932e+09
```

```
In [25]: #Separate spatial data into grid for computation in smaller bits
         #Code mostly taken from 'Sandro' on Kaggle Kernels

         def prepare_data(df, n_cell_x, n_cell_y):
         #     """
         #     Feature engineering and computation of the grid.
         #     """
             #Creating the grid
             size_x = 10. / n_cell_x #divide 10 (x values in [0,10]) by number of c
             size_y = 10. / n_cell_y #divide by 10 (y values in [0, 10]) by number
             eps = 0.00001 #why do eps? Just so for values less than this, just do
             xs = np.where(df.x.values < eps, 0, df.x.values - eps)
             ys = np.where(df.y.values < eps, 0, df.y.values - eps)
             pos_x = (xs / size_x).astype(np.int64) #changing position for x into r
             pos_y = (ys / size_y).astype(np.int64) #changing position for y into r
             df['grid_cell'] = (pos_y * n_cell_x + pos_x) #the grid # (how many y's

             #Feature engineering
             fw = [500, 500, 4, 3, 1./22., 2, 10] #feature weights (black magic her
                 #each fw represents x, y, hour, weekday, day, month, year changing
                 #Note: Changed y fw from 1000 to 750 b/c exceeds int64 and produce
                 #thing to keep in mind: n_cell_x can't be too big, then will have

             df.x = df.x.values * fw[0]
             df.y = df.y.values * fw[1]
             initial_date = np.datetime64('2014-01-01T01:01', dtype='datetime64[m]'
             d_times = pd.DatetimeIndex(initial_date + np.timedelta64(int(mn), 'm')
                                        for mn in df.time.values)
             df['hour'] = d_times.hour * fw[2]
             df['weekday'] = d_times.weekday * fw[3]
             df['day'] = (d_times.dayofyear * fw[4]).astype(int)
             df['month'] = d_times.month * fw[5]
             df['year'] = (d_times.year - 2013) * fw[6]

             df = df.drop(['time'], axis=1) #drop time b/c converted into time
             return df
```

In [26]:

```
#Also code adapted from 'Sandro'
#jk most of this stuff is mine now

def process_one_cell(df_train, df_test, grid_id, threshold, n_cell_x, n_ce
    """
    Throw in a training dataset and it will split it into local training a
    do a KNN classification inside one grid cell.
    """

    #Working on df_train to train ONE CELL (from grid_cell column)
    df_cell_train = df_train.loc[df_train.grid_cell == grid_id] #gets all
    place_counts = df_cell_train.place_id.value_counts()
        #getting counts of places in that cell

    mask = (place_counts[df_cell_train.place_id.values] >= threshold).valu
    df_cell_train = df_cell_train.loc[mask] #weeds out ID's with less than

    #Working on df_test
    df_cell_test = df_test.loc[df_test.grid_cell == grid_id]

    #Saving row ids of test for our output later, which will be along with
    row_ids = df_cell_test.index


    features = ['x','y','hour','day','weekday','month','year','accuracy']

    train_y = df_cell_train['place_id']
    train_x = df_cell_train[features]

    test_x = df_cell_test[features]
```

```
             # KNN algorithm and test accuracy
             knn = KNN(15) #15 nearest neighbors
             knn.fit(train_x, train_y) #classifying data based on 15-nearest neighb
             all_preds = knn.predict_proba(test_x)

             #Saving predictions into preds_per_cell
             preds_per_cell = np.zeros((test_x.shape[0], 3), dtype=int)
             for record in range(len(all_preds)):
                 top3_idx = all_preds[record].argsort()[-3:][::-1]
                 preds = knn.classes_[top3_idx]
                 preds_per_cell[record] = preds

             train_acc = knn.score(train_x, train_y) # score KNN on train set

             return preds_per_cell, row_ids, train_acc

In [27]: def process_grid(df_train, df_test, threshold, n_cells, n_cell_x, n_cell_y
             """
             Iterates over all grid cells, return average training and testing accu
             """
             preds = np.zeros((df_test.shape[0], 3), dtype=int)
             small_train_acc_sum = 0
             #small_test_acc_sum = 0

             for grid_id in range(n_cells):
                 if grid_id % 100 == 0:
                     print('iter: %s' %(grid_id)) #Print iteration per 100 grids
                     print(small_train_acc_sum / (grid_id - 1))

                 #Applying classifier to one grid cell
                 pred_labels, row_ids, small_train_acc = process_one_cell(df_train,
                                                                          grid_id,
                                                                          n_cell_x,

                 small_train_acc_sum += small_train_acc #add up each training accu
                 #small_test_acc_sum += small_test_acc #add up each testing accurac

                 #Updating predictions
                 preds[row_ids] = pred_labels

             train_acc_avg = small_train_acc_sum/n_cells
             #test_acc_avg = small_test_acc_sum/n_cells

             print('Generating submission file ...')

             #Auxiliary dataframe with the 3 best predictions for each sample
```

```
    df_aux = pd.DataFrame(preds, dtype=str, columns = ['l1', 'l2', 'l3'])

    #Concatenating the 3 predictions for each sample
    ds_sub = df_aux.l1.str.cat([df_aux.l2, df_aux.l3], sep=' ')

    #Writting to csv
    ds_sub.name = 'place_id'
    ds_sub.to_csv('sub_knn.csv', index=True, header=True, index_label='row

    return train_acc_avg#, test_acc_avg
```

1.1 Notes

All you need to call is the process_grid() method with appropriate parameters

- Inside process_grid(), the process_one_cell() method is used on each cell, and
- Inside each process_one_cell() call, prepare_data() is used

1.2 Implementation

```
In [30]: #Required Variables
         n_cell_x = 30
         n_cell_y = 30
         threshold = 3
```

```
In [35]: #First, feature engineering + separating training data into amplified gri
         df_train = prepare_data(train, n_cell_x, n_cell_y)
         df_test = prepare_data(test, n_cell_x, n_cell_y)
```

```
In [36]: # #Then running classification model through process_grid
         process_grid(df_train, df_test, threshold, n_cell_x * n_cell_y, n_cell_x,
```

```
iter: 0
-0.0
iter: 100
0.589275893058
iter: 200
0.583516438543
iter: 300
0.583023747064
iter: 400
0.583135357303
iter: 500
0.582176202611
iter: 600
0.582177203954
iter: 700
0.581937037505
```

```
iter: 800
0.581684408644
Generating submission file ...
```

```
Out[36]: 0.58086004301081262
```

```
In [ ]:
```

```
In [ ]:
```