New-York-py-v1.0

February 5, 2019

1 INTRODUCTION

A bussinesman desires to open a restaurant he already had a thought in Mind about one place, he is new to the Manhattan NY Area and called me (as a datascient) to do some machine learning and predict if his restaurant will be succesfull or not.

1 - The aim of this project is to respond to a question : Would my restaurant be succesfull? where is the best place to run a restaurant in NY Manhattan Area

2 - Data will be gathered from Foursquare , Rating of existing restaurant and Number of Likes

3 - Methodology that's being used is as follow :

4- Once we get the best ranked restaurants in NY area we will run a model trying to determine the attributes , and determine what make them best ranked 4. We will test and train the model the avaliable data (Surely we will use the SVM algo)

5 . We fix an accuracy of 80% for the model to be validated ; Refine the model untill we get the desired accuracy

6 . Once the model is validated , we will get attributes of the best ranked restaurant

7. Data can be easily gotten from Foursquare (Ranking, Number of likes) , neighborhoud can be gotten from Yelp

8. Now can proceed with a businees model for determining if a restaurant Will be sucessfull or not

9. We will apply this business model to the restaurant wanted , and predict what if would be successfull or not

1.0.1 Here are information about the Bussinesman new restaruant

Categorie of Food = Pizza

Neghborhood = Chinatown

Coordiates(40.715618, -73.994279)

2 The Aim of This project is to Analyse restaurants Manhatan Area

```
In [4]: import numpy as np # library to handle data in a vectorized manner

        import pandas as pd # library for data analsysis
        pd.set_option('display.max_columns', None)
```

```
        pd.set_option('display.max_rows', None)

        import json # library to handle JSON files
        #!conda install -c conda-forge geopy --yes # uncomment this line if you haven't complete
        from geopy.geocoders import Nominatim # convert an address into latitude and longitude v

        import requests # library to handle requests
        from pandas.io.json import json_normalize # tranform JSON file into a pandas dataframe

        # Matplotlib and associated plotting modules
        import matplotlib.cm as cm
        import matplotlib.colors as colors

        # import k-means from clustering stage
        from sklearn.cluster import KMeans

        #!conda install -c conda-forge folium=0.5.0 --yes # uncomment this line if you haven't c
        import folium # map rendering library

        print('Libraries imported.')
```

```
Libraries imported.
```

For your convenience, I downloaded the files and placed it on the server, so you can simply run a `wget` command and access the data. So let's go ahead and do that.

```
In [2]: !wget -q -O 'newyork_data.json' https://ibm.box.com/shared/static/fbpwbovar7lf8p5sgddm06
        print('Data downloaded!')
```

```
Data downloaded!
```

Next, let's load the data.

```
In [3]: import json
        with open('newyork_data.json') as json_data:
            newyork_data = json.load(json_data)

In [4]: neighborhoods_data = newyork_data['features']

In [5]: # define the dataframe columns
        column_names = ['Borough', 'Neighborhood', 'Latitude', 'Longitude']

        # instantiate the dataframe
        neighborhoods = pd.DataFrame(columns=column_names)

In [6]: for data in neighborhoods_data:
            borough = neighborhood_name = data['properties']['borough']
```

```
            neighborhood_name = data['properties']['name']

            neighborhood_latlon = data['geometry']['coordinates']
            neighborhood_lat = neighborhood_latlon[1]
            neighborhood_lon = neighborhood_latlon[0]

            neighborhoods = neighborhoods.append({'Borough': borough,
                                                  'Neighborhood': neighborhood_name,
                                                  'Latitude': neighborhood_lat,
                                                  'Longitude': neighborhood_lon}, ignore_index=T
```

```
In [ ]: manhattan_data = neighborhoods[neighborhoods['Borough'] == 'Manhattan'].reset_index(drop
        manhattan_data.head()
```

```
In [9]: address = 'Manhattan, NY'

        geolocator = Nominatim()
        location = geolocator.geocode(address)
        latitude = location.latitude
        longitude = location.longitude
        print('The geograpical coordinate of Manhattan are {}, {}.'.format(latitude, longitude))
```

```
The geograpical coordinate of Manhattan are 40.7900869, -73.9598295.
```

Next, we are going to start utilizing the Foursquare API to explore the neighborhoods and segment them.

Define Foursquare Credentials and Version We are using 2 CLIENT IDS SINCE FORSQUARE HAS A LIMIT OF QUOTA

```
In [34]: CLIENT_ID = 'RLWSL4OHIHSS2MWCUYLGNIVY24TRWXQCGTKDHMJKNIYUDP1U' # your Foursquare ID
         CLIENT_SECRET = '2NOS2HIYCUL32QEH4QM4Y3ZYUR1CEH350LDZPWQDLLKGRQYM' # your Foursquare Se
         VERSION = '20180605' # Foursquare API version

         print('Your credentails:')
         print('CLIENT_ID: ' + CLIENT_ID)
         print('CLIENT_SECRET:' + CLIENT_SECRET)

         CLIENT_ID1 = 'OOBFNNVFQFDRMNADZTQQVDP4YWGCNBAWL355VFMAFAOIRBVZ' # your Foursquare ID
         CLIENT_SECRET1 = 'E4R5DUJNTOES3SX5TMVZAKRKBYOYXDVSHIPUJQEX1YNY5OET' # your Foursquare S
         VERSION = '20180605' # Foursquare API version
```

```
Your credentails:
CLIENT_ID: RLWSL4OHIHSS2MWCUYLGNIVY24TRWXQCGTKDHMJKNIYUDP1U
CLIENT_SECRET:2NOS2HIYCUL32QEH4QM4Y3ZYUR1CEH350LDZPWQDLLKGRQYM
```

Let's explore the first neighborhood in our dataframe. Get the neighborhood's name.

Now, let's get the top 50 venues that are in Marble Hill within a radius of 200 meters. Since we dont have a premium Foursquare Account , Lets limit the data

```
In [12]: # type your answer here

         LIMIT = 50 # limit of number of venues returned by Foursquare API


         radius = 200 # define radius
         section='food'

          # create URL
         url = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}
             CLIENT_ID,
             CLIENT_SECRET,
             VERSION,
             neighborhood_latitude,
             neighborhood_longitude,
             radius,
             section,
             LIMIT)
         url # display URL
```

```
Out[12]: 'https://api.foursquare.com/v2/venues/explore?&client_id=RLWSL4OHIHSS2MWCUYLGNIVY24TRWX
```

```
In [13]: results = requests.get(url).json()
         #results
```

```
In [ ]: venues = results['response']['groups'][0]['items']
        venues
```

```
In [15]: # function that extracts the category of the venue
         def get_category_type(row):
             try:
                 categories_list = row['categories']
             except:
                 categories_list = row['venue.categories']

             if len(categories_list) == 0:
                 return None
             else:
                 return categories_list[0]['name']
```

Now we are ready to clean the json and structure it into a *pandas* dataframe.

```
In [16]: venues = results['response']['groups'][0]['items']
         venues
         nearby_venues = json_normalize(venues)
```

2.1 2. Explore Neighborhoods in Manhattan

Let's create a function to repeat the same process to all the neighborhoods in Manhattan

```
In [20]: def getNearbyVenues(names, latitudes, longitudes, radius=100):

            venues_list=[]
            result_Like=[]
            for name, lat, lng in zip(names, latitudes, longitudes):
                print(name)

                # create the API request URL
                url = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret
                    CLIENT_ID,
                    CLIENT_SECRET,
                    VERSION,
                    lat,
                    lng,
                    radius,
                    section,
                    LIMIT)

                # make the GET request
                results = requests.get(url).json()["response"]['groups'][0]['items']

                # return only relevant information for each nearby venue
                venues_list.append([(
                    name,
                    lat,
                    lng,
                    v['venue']['name'],
                    v['venue']['location']['lat'],
                    v['venue']['location']['lng'],
                    v['venue']['categories'][0]['name'],
                    v['venue']['id']) for v in results])


            nearby_venues = pd.DataFrame([item for venue_list in venues_list for item in venue_
            nearby_venues.columns = ['Neighborhood',
                          'Neighborhood Latitude',
                          'Neighborhood Longitude',
                          'Venue',
                          'Venue Latitude',
                          'Venue Longitude',
                          'Venue Category','Venue_id']

            return(nearby_venues)
```

Now write the code to run the above function on each neighborhood and create a new dataframe called *manhattan_venues*.

```
In [ ]: # type your answer here
        manhattan_venues = getNearbyVenues(names=manhattan_data['Neighborhood'],
                                           latitudes=manhattan_data['Latitude'],
                                           longitudes=manhattan_data['Longitude']
                                          )
```

Let's check the size of the resulting dataframe

```
In [22]: print(manhattan_venues.shape)
         manhattan_venues.head()
```

```
(182, 8)
```

```
Out[22]:   Neighborhood  Neighborhood Latitude  Neighborhood Longitude  \
         0    Chinatown              40.715618              -73.994279
         1    Chinatown              40.715618              -73.994279
         2    Chinatown              40.715618              -73.994279
         3    Chinatown              40.715618              -73.994279
         4    Chinatown              40.715618              -73.994279

                                    Venue  Venue Latitude  Venue Longitude  \
         0  Super Taste () (Super Taste)       40.715198       -73.993511
         1                East Dumpling       40.715912       -73.993193
         2              Wong Wah Bakery       40.715335       -73.993365
         3                Highline Deli       40.715345       -73.993329
         4                   Sheng Wang       40.715159       -73.993553

                Venue Category                  Venue_id
         0         Noodle House  49e8ee2ef964a520af651fe3
         1  Dumpling Restaurant  588e11ed0393cc517e783401
         2               Bakery  4b9bd959f964a520da2c36e3
         3        Deli / Bodega  50611575498e8f79e53d0d6a
         4         Noodle House  4a960366f964a520d12520e3
```

2.2 Let's Find the Rating and Number of Likes of each venue

this requires some cleaning, and Foursquare Quota is very limited in this API , we are only using 180 row

```
In [ ]: #Now lets Get the Some other informations bout venues
        d=[]
        for Id in manhattan_venues.Venue_id:
            url = 'https://api.foursquare.com/v2/venues/{}?&client_id={}&client_secret={}&v={}'.
            print(Id)
            result=requests.get(url).json()
```

```
        try:
            rating=result['response']['venue']['rating']
            likes=result['response']['venue']['likes']['count']
        except:
            rating=0
            likes=0
        d.append({'venue_id': Id, 'Rating': rating, 'Number_likes':likes})
```

```
In [66]: d
         venue_rating=pd.DataFrame(d)
         venue_rating.columns.values[2]='Venue_id'

         #pd.merge(df_users, df_valid_users, how='inner', on=['user_id'])

         Data=pd.merge(venue_rating,manhattan_venues,how='inner',on=['Venue_id'])
         Data.to_csv('Newyork_restaurant_rating.csv')
```

3 All Required Data is now loaded , Lets do Some Machine Learning

3.1 Since we dont need to proces data anymore let's recover it from the saved file to avoid exceeding foursquare Quota again

```
In [5]: Data=pd.read_csv('Newyork_restaurant_rating.csv')
```

Now Lets Make an Assumption that a sucessfull restaurant is at least wiht more than 100 Likes or a Rating superior than

```
In [6]: Data.head()
```

```
Out[6]:    Unnamed: 0  Number_likes  Rating                 Venue_id Neighborhood  \
        0           0           185     8.7  49e8ee2ef964a520af651fe3    Chinatown
        1           1            11     7.5  588e11ed0393cc517e783401    Chinatown
        2           2             3     7.0  4b9bd959f964a520da2c36e3    Chinatown
        3           3             8     6.7  50611575498e8f79e53d0d6a    Chinatown
        4           4            23     6.0  4a960366f964a520d12520e3    Chinatown

           Neighborhood Latitude  Neighborhood Longitude  \
        0              40.715618              -73.994279
        1              40.715618              -73.994279
        2              40.715618              -73.994279
        3              40.715618              -73.994279
        4              40.715618              -73.994279

                                    Venue  Venue Latitude  Venue Longitude  \
        0  Super Taste () (Super Taste)       40.715198       -73.993511
```

```
1                      East Dumpling       40.715912       -73.993193
2                    Wong Wah Bakery       40.715335       -73.993365
3                      Highline Deli       40.715345       -73.993329
4                         Sheng Wang       40.715159       -73.993553

        Venue Category
0         Noodle House
1  Dumpling Restaurant
2               Bakery
3        Deli / Bodega
4         Noodle House
```

```
In [ ]: Data['Succesfull'] = Data['Number_likes'].apply(lambda x: '1' if x >= 100 else '0')
        Data['Succesfull'] = Data['Rating'].apply(lambda x: '1' if x >= 7 else '0' )

        Data
```

3.2 Now lets apply SVM algorithm

We will use SVM (Support Vector Machines) to build and train a model using existing restaurant records, and classify cells to whether there are sucessfull or Not.

SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. A separator between the categories is found, then the data is transformed in such a way that the separator could be drawn as a hyperplane. Following this, characteristics of new data can be used to predict the group to which a new record should belong.

```
In [43]: import pandas as pd
         import pylab as pl
         import numpy as np
         import scipy.optimize as opt
         from sklearn import preprocessing
         from sklearn.model_selection import train_test_split
         %matplotlib inline
         import matplotlib.pyplot as plt
```

```
In [53]: feature_df = Data[['Venue Longitude', 'Venue Latitude']]
         X = np.asarray(feature_df)
         X[0:5]
```

```
Out[53]: array([[-73.99,  40.72],
                [-73.99,  40.72],
                [-73.99,  40.72],
                [-73.99,  40.72],
                [-73.99,  40.72]])
```

```
In [54]: y = np.asarray(Data['Succesfull'])
         y [0:5]
```

```
Out[54]: array(['1', '1', '1', '0', '0'], dtype=object)
```

```
In [55]: X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=
         print ('Train set:', X_train.shape,  y_train.shape)
         print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (145, 2) (145,)
Test set: (37, 2) (37,)
```

```
In [56]: from sklearn import svm
         clf = svm.SVC(kernel='rbf')
         clf.fit(X_train, y_train)
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/svm/base.py:196: FutureWarning: The d
  "avoid this warning.", FutureWarning)
```

```
Out[56]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
           kernel='rbf', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
In [57]: yhat = clf.predict(X_test)
         yhat [0:5]
```

```
Out[57]: array(['1', '1', '1', '1', '1'], dtype=object)
```

4 Evaluation

```
In [58]: from sklearn.metrics import classification_report, confusion_matrix
         import itertools
```

```
In [59]: def plot_confusion_matrix(cm, classes,
                                   normalize=False,
                                   title='Confusion matrix',
                                   cmap=plt.cm.Blues):
             """
             This function prints and plots the confusion matrix.
             Normalization can be applied by setting `normalize=True`.
             """
             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
                 print("Normalized confusion matrix")
             else:
                 print('Confusion matrix, without normalization')

             print(cm)
```

```
            plt.imshow(cm, interpolation='nearest', cmap=cmap)
            plt.title(title)
            plt.colorbar()
            tick_marks = np.arange(len(classes))
            plt.xticks(tick_marks, classes, rotation=45)
            plt.yticks(tick_marks, classes)

            fmt = '.2f' if normalize else 'd'
            thresh = cm.max() / 2.
            for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                plt.text(j, i, format(cm[i, j], fmt),
                         horizontalalignment="center",
                         color="white" if cm[i, j] > thresh else "black")

            plt.tight_layout()
            plt.ylabel('True label')
            plt.xlabel('Predicted label')
```

In [60]:
```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=['0','1'])
np.set_printoptions(precision=2)

print (classification_report(y_test, yhat))

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Succesfull','Non_Succesfull'],normalize= Fa
```

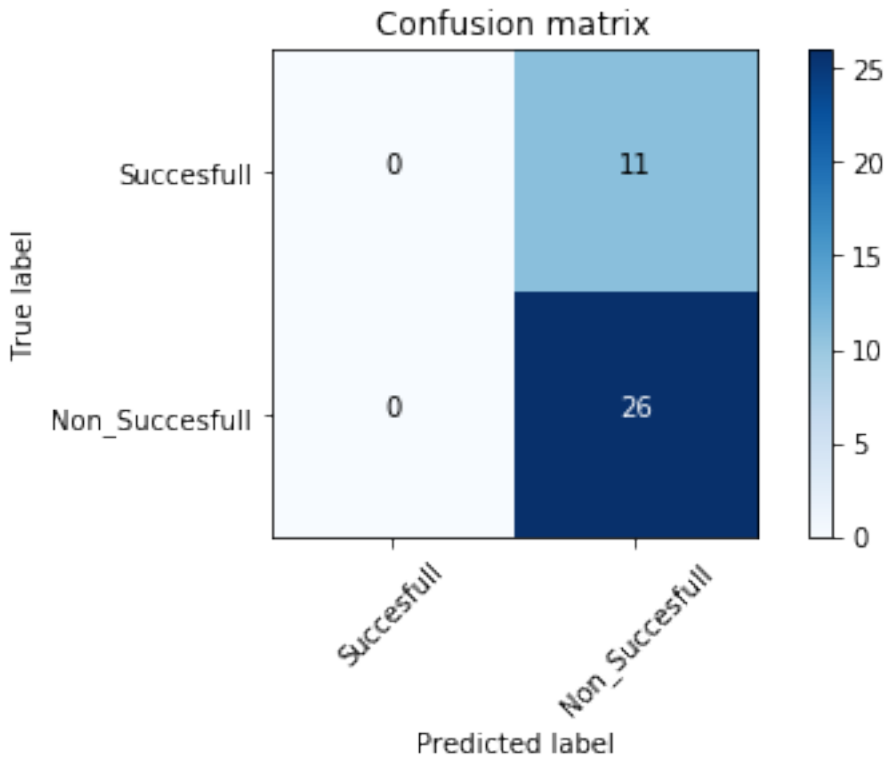
```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: Undef
  'precision', 'predicted', average, warn_for)
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        11
           1       0.70      1.00      0.83        26

   micro avg       0.70      0.70      0.70        37
   macro avg       0.35      0.50      0.41        37
weighted avg       0.49      0.70      0.58        37

Confusion matrix, without normalization
[[ 0 11]
 [ 0 26]]
```



4.0.1 Looks Like Our model is doing Great on Predicting Sucessfull restaurant (Only 2 restuarant were predicted with error)

```
In [61]: from sklearn.metrics import f1_score
         f1_score(y_test, yhat, average='weighted')
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: Undef
  'precision', 'predicted', average, warn_for)
```

```
Out[61]: 0.58000858000858
```

4.1 It looks Like the Accuracy is not good enough , let's make a stronger model using more attributes

4.1.1 we will try to add the 'category' of the restaurant, not only the geographical coordonates

Now Let's Run our model in the Busnessman retaurant to get the result

```
In [ ]: ## Lets normalize the Venue Category to Use in Sikit Learn
```

```
In [ ]: Data
```

```
In [90]: Data['category_id'] = Data['Venue Category'].factorize()[0]
         Data['Neignborhood_id']=Data['Neighborhood'].factorize()[0]

In [48]: Categorie = 'Pizza'
         Neghborhood = 'Chinatown'
         Latitude= 40.715618
         Longitude= -73.994279

In [ ]: Data['Succesfull'] = Data['Number_likes'].apply(lambda x: '1' if x >= 100 else '0')
        Data['Succesfull'] = Data['Rating'].apply(lambda x: '1' if x >= 7 else '0' )

        Data

In [52]: import pandas as pd
         import pylab as pl
         import numpy as np
         import scipy.optimize as opt
         from sklearn import preprocessing
         from sklearn.model_selection import train_test_split
         %matplotlib inline
         import matplotlib.pyplot as plt

In [91]: feature_df = Data[['Venue Longitude', 'Venue Latitude','category_id','Neignborhood_id']]
         X = np.asarray(feature_df)
         X[0:5]

Out[91]: array([[-73.99351119,  40.71519758,   0.        ,   0.        ],
                [-73.99319261,  40.71591184,   1.        ,   0.        ],
                [-73.99336451,  40.7153352 ,   2.        ,   0.        ],
                [-73.99332854,  40.71534522,   3.        ,   0.        ],
                [-73.99355311,  40.71515876,   0.        ,   0.        ]])

In [92]: y = np.asarray(Data['Succesfull'])
         y [0:5]

Out[92]: array(['1', '1', '1', '0', '0'], dtype=object)

In [93]: X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=
         print ('Train set:', X_train.shape,  y_train.shape)
         print ('Test set:', X_test.shape,  y_test.shape)

Train set: (178, 4) (178,)
Test set: (4, 4) (4,)


In [97]: X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=
         print ('Train set:', X_train.shape,  y_train.shape)
         print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (145, 4) (145,)
Test set: (37, 4) (37,)
```

```
In [98]: from sklearn import svm
         clf = svm.SVC(kernel='rbf')
         clf.fit(X_train, y_train)
```

```
/home/jupyterlab/conda/lib/python3.6/site-packages/sklearn/svm/base.py:196: FutureWarning: The d
  "avoid this warning.", FutureWarning)
```

```
Out[98]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
           kernel='rbf', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
In [99]: yhat = clf.predict(X_test)
         yhat [0:20]
```

```
Out[99]: array(['0', '1', '0', '1', '1', '0', '1', '1', '1', '0', '1', '1', '1',
                '1', '1', '1', '1', '1', '1', '0'], dtype=object)
```

```
In [100]: from sklearn.metrics import f1_score
          f1_score(y_test, yhat, average='weighted')
```

```
Out[100]: 0.8256113256113256
```

4.2 Great Results on model Score after adding attributes f Neignborhood and Restaurant Category

5 Lets Get a response for our Restaurant

```
In [ ]: #Data
```

```
In [105]: Categorie = 'Pizza'
          Neghborhood = 'Chinatown'
          Latitude= 40.715618
          Longitude= -73.994279

          # China toum = Neignborhood _Id is 0 and
          # Pizza category = Category_Id is 12
          # Lets Predict

          #X_test
          clf.predict([[-73.994279,40.715618,0.,0.]])
```

```
Out[105]: array(['1'], dtype=object)
```

5.0.1 Great Looks Like our restaurant is a Sucessfull, The Businessman should be happy

```
In [ ]:
```