NYU - Data Science Project Report

Fall 2015

Professor Foster Provost

Predictive Policing for Robberies in New York City

by

Ali Josue Limon (ajl649)

Kavitha Vishwanathan (kv668)

Xuan Zhou (xz694)

Business Understanding

Can we predict robberies in New York City? How likely for a location, or an area, to see robberies?

The City of New York provides to the public a crime map, a heat map where different precincts are colored differently based on crimes per 1,000 residents. While this information is helpful in telling where a certain type of crime usually happens by precinct, it does not seem enough. Will a precinct, essentially also an area, always see that many crimes? What more may we know about the occurrence of crimes?

A tool named PredPol has been said to be effective in guiding police officers where and when to patrol, by making crime predictions using three data points “past type, place and time of crime” and an “algorithm based on criminal behavior patterns”. In line with PredPol’s approach, while dropping the element of time, we want to concentrate on robberies, and we want to dive in deeper and understand more about the relations between socio-economic factors and the occurrence of robberies. It will not be a causal relationship, given the complexity of criminal activities, but the factors taken into consideration should be relevant enough and potentially contribute to the occurrence of robberies.

Our study will be a supervised learning process, specifically classification. The objective is, when given relevant features of an area in the future, we would be able to tell how much robbery that area is likely to see.

When brainstorming, here are some of the factors that we think are relevant, for instance:

Is the area residential or commercial--commercial areas could be more at risk, or maybe the opposite;

Population density--more people, more potential victims, or maybe the opposite;

Ethnic composition;

Number and quality of street lighting;

Number of surveillance;

Police presence in the area--if quantifiable;

Income level;

Number of public transportation stations--locations near a transportation station could be hotspots for robberies, or maybe the opposite;

And so on.

Data Understanding

It turns out that not all data that we desire are within reach. For example, for the number of surveillance, we were only able to find data for the borough of Manhattan, and the data was in the format of either a static map or a textual listing. As a result, we decided to drop that factor. Similarly, data for a number of other factors are not accessible or available, either. But still, we are able to find the following:

1.Crime data:

Source: NYC Crime Map by The City of New York (http://maps.nyc.gov/crime/)

Description: Data is in the format of an interactive heat map grouped by police precinct. Crime count is available for six major types of crime, including murder, robbery, felony assault, burglary, grand larceny, and grand larceny for motor vehicle. At the time of writing this report, monthly data is available for the time frame of January 2014--November 2015.

2. Socio-economic data:

Source: American Community Survey by the U.S. Census Bureau, 5-year estimate, 2013. (http://www.socialexplorer.com/tables/ACS2013_5yr/)

Description: The source allows us to navigate through the dataset by different geographical type. Once the geographic type and areas are chosen, data for a variety of factors (more than 160) are available, including total population, population density, unemployment rate, household income level, and so forth.

3. Subway data:

Source: Subway Stations from NYC OpenData by The City of New York (https://data.cityofnewyork.us/Transportation/Subway-Stations/arq3-7z49)

Description: Data is in the format of an interactive map, but it is also downloadable in the format of a CSV or JSON file. Names of all subway stations within New York City are provided.

4. Census tract coordinates:

Source: Boundary Location API by Investigative Reporters and Editors (http://census.ire.org/docs/boundary.html

http://census.ire.org/geo/1.0/boundary-set/tracts/36005000100)

Description: Data for coordinates that define each census tract (polygon) geographically. This will be useful for our data preparation, which will be explained further later in the report.

Data Preparation

Ever since the very beginning of our project, we faced the problem of deciding on our unit of interest. As seen from the American Community Survey dataset as well, there are many different ways of segmenting and studying a wider area, even within just a city. Should we study by census tract, by block group, by community district, by neighborhood, by zip code, or in this case in particular, by police precinct?

Initially and very intuitively, we preferred to use neighborhood, since neighborhoods, the way they were formed, were based on different characteristics over the years among the general public to begin with and should provide ideal segmentation. However, for the exact same reason, there is not an official definition on the boundaries of different neighborhoods. For example, people may have different perceptions of what “Harlem” covers. Plus, there are few datasets with information of neighborhood available. So we cannot proceed with “neighborhood”.

For police precinct, while it makes perfect sense to observe the crimes alone by police precinct, unfortunately we don’t have demographic information by police precinct available, which will make it difficult to combine data. For other candidates of units, a

community district seems to cover too big an area, while the level of block group may be too fine-grained.

Eventually we decided to go with census tract. We can have socio-economic data by census tract easily, and with the census tract polygon data available, we will be able to organize crime data and subway data by census tract as well. Also, per the definition by U.S. Census Bureau, census tracts are "designed to be relatively homogeneous units with respect to population characteristics, economic status, and living conditions", making the census tract (shortened to just "tract" from now on for brevity) an ideal choice of unit for our study. Before combining the data, however, individual datasets need to be processed, too.

1. Original data processing:

a. Crime dataset:

As our crime dataset is in the form of a map, we wrote an API to get the data in CSV format. In the API, our code calls the *Table.Features* endpoint of the Google Maps Engine API. The output of the code is then stored as a table, which will later be converted into CSV format. The new CSV file includes original data in the map, along with respective latitude and longitude information.

b. Socio-economic data:

This dataset includes some redundant columns inherently. For instance, the column of "Total Population" appears a number of times. We deleted all the redundant ones. We also applied some domain knowledge, further deleting some columns/factors

that might not be relevant to our target variable, the occurrence of robberies. At the end, we had 90 columns left, down from the original 160.

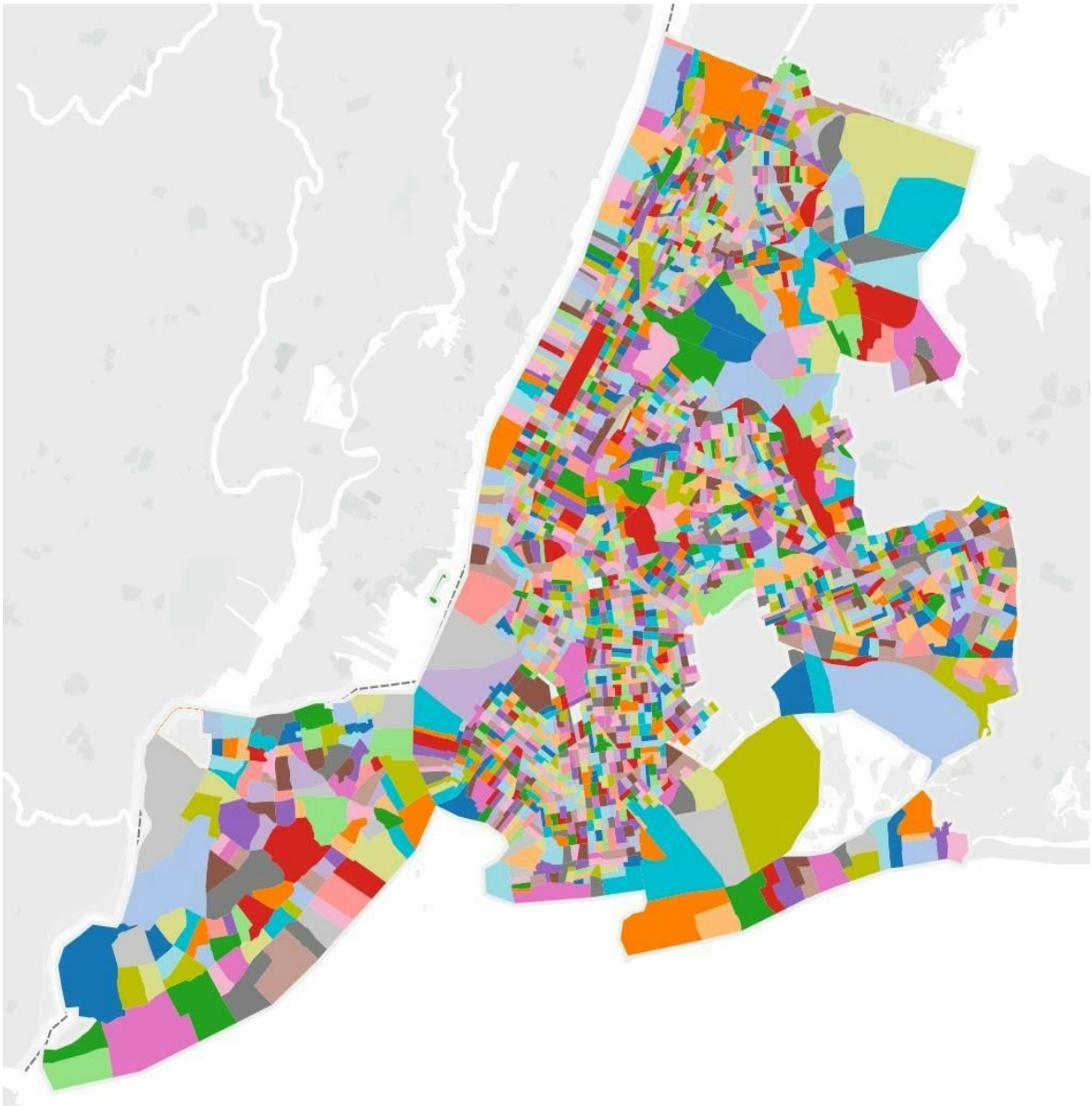
c. Subway stations:

To make the data suitable for our purpose, we need the latitude and longitude of the stations. We wrote another set of code in Java and used the Google Map API to find the coordinates.

2. Preparing to combine the original datasets, with tract polygon coordinates:

We have decided to use tract as our studying unit, based on which we will have to combine our multiple datasets. Crime data and subway data do not come directly with tract information, but we will be able to “translate” their location information into coordinates, and then map the coordinates to tracts to which they belong.

We found a shape file (polygon coordinates) for tracts in New York City**.** Each tract has a separate web page with its relevant information. We used RegEx to identify the specific information of the polygons and worked with them. For every tract, there is a set of associated coordinates and a corresponding polygon. We recorded all the polygons (tracts) in one dataset. Below is the visualization for the polygon-shaped tracts done with Tableau.



Visualization of New York Tracts

3. Merging all datasets via the tract polygons:

We wanted to use the polygon dataset to identify the number of robberies in each tract. The crime dataset was huge, and we tried two different methods before achieving our goal.

The first method was using the *shapely.geometry* package of Python, specifically, using the function *shapely.geometry.Polygon* to create the shape of the polygons and the function *shape.contains* to identify if a polygon contains occurrence of robberies. However, we found this method to be very slow when computing, and it was not realistic to continue with it.

We turned to a second method, working with dictionaries and creating a function of our own named point_in_polygon. The function determines if a robbery happens within a certain polygon by comparing against all coordinates, without actually creating the shapes of the polygons first. This method is much more efficient, and we found it optimal to work with. There were very few robberies that were not identified by our function. Most of the unidentified seemed to have happened in locations outside of New York (We are not sure about the cause. It could be errors from the source). We deleted them.

For the subway dataset, the combining process is similar. In the end, we have a new master file with everything together, organized by tracts.

Modeling

Step 1: Classification (K-Means Clustering)

The first step, since the number of robberies was numeric and consecutive, was to classify the number of robberies into categories through meaningful partitioning. We employed k-means clustering to the two columns in our data, number of robberies and

population density, and after a trial-and-error process of choosing different number of clusters, we decided on having 3 clusters.

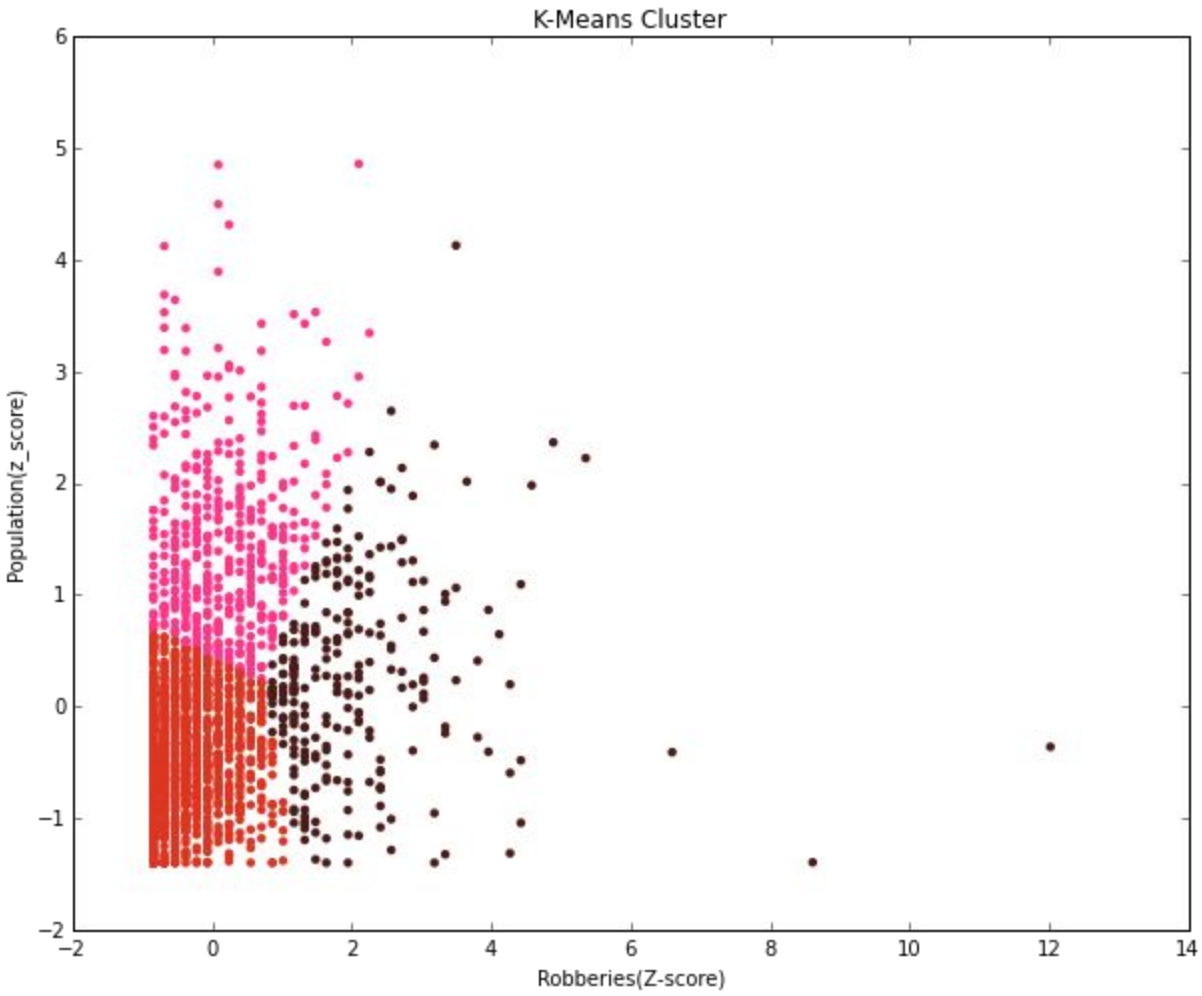
As our dataset is a numeric one, we normalized the dataset using *Z_Score* before running the model. Below are the various statistics observed in each cluster.

Low	count	mean	std	min	25%	50%	75%	max
ROBBERIES	1461	-0.41	0.45	-0.84	-0.84	-0.53	-0.22	1.02
Population Density	1461	-0.49	0.50	-1.40	-0.86	-0.51	-0.11	0.69

Medium	count	mean	std	min	25%	50%	75%	max
ROBBERIES	442	0.17	0.65	-0.84	-0.38	0.09	0.55	2.26
Population Density	442	1.45	0.82	0.25	0.82	1.27	1.87	4.86

High	count	mean	std	min	25%	50%	75%	max
ROBBERIES	257	2.04	1.19	0.86	1.33	1.79	2.41	12.02
Population Density	257	0.26	0.88	-1.40	-0.34	0.19	0.75	4.13

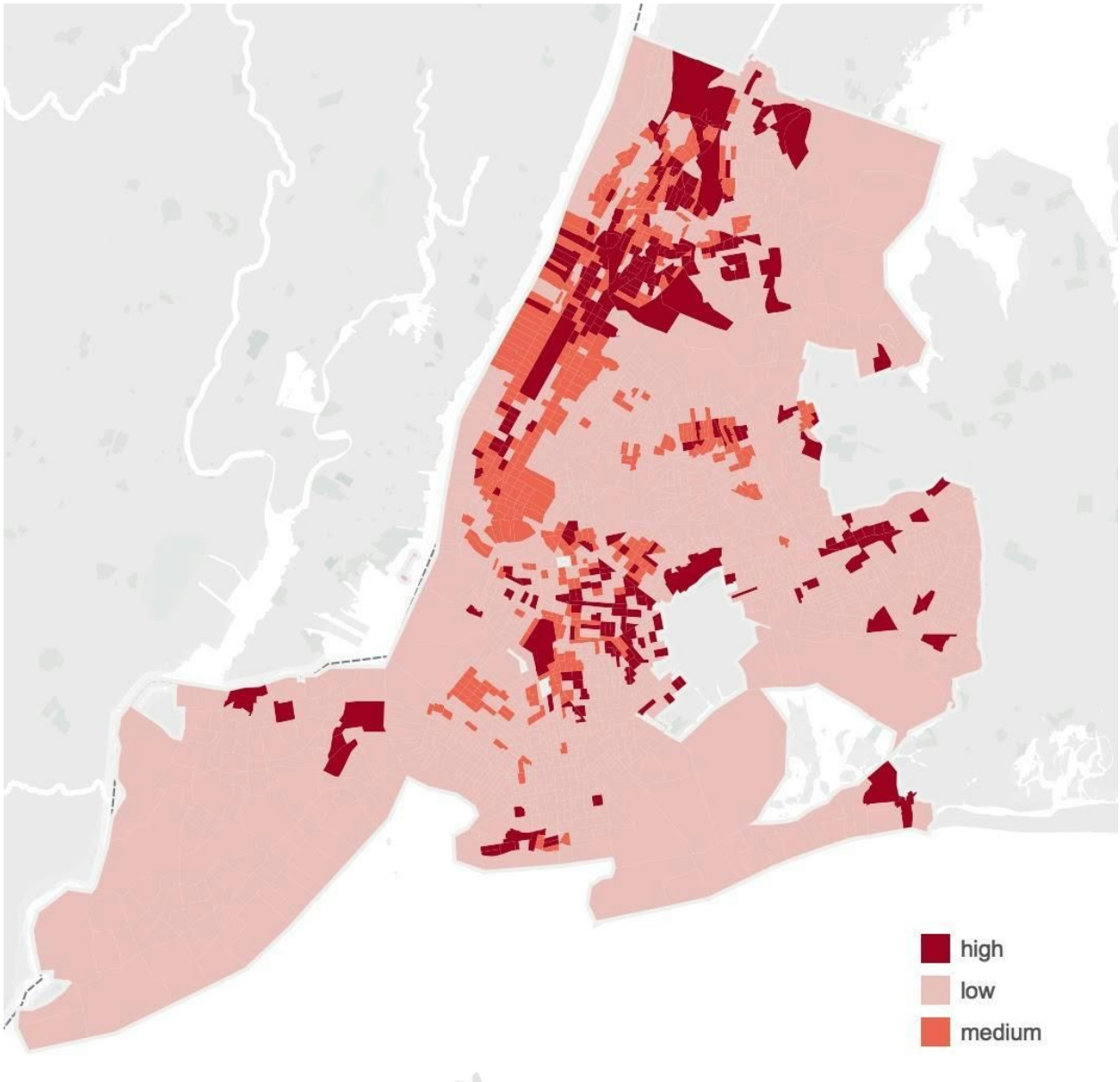
By observing the above clusters, we could find that they are each distinctive with the number of robberies. Based on this, we identified the three clusters as low, medium, and high robbery rate areas.



Visualization of K-Means Cluster

We added a column called “crime-classification” in our master file. Now each tract would be given a value (low/medium/high) based on the clustering result. This column would be our target variable, and we now had a complete dataset with labels and we could proceed with further modelling.

Below is our heat map of New York City. It reflects our findings from the clustering, showing tracts with three different robbery rate across New York City. This is again done with Tableau.



Visualization of NYC based on Robberies

Step 2: Feature Selection (Decision Tree)

After a second round of manual feature selection, our master dataset (along with the newly added column ‘crime_classification’) now has around 90 features. To avoid overfitting, we chose the Decision Tree model to help us further select the features that

will provide us with the biggest information gain. We built a decision tree model and fit our data. Using one of the attributes of sklearn 'decision tree classifier' known as "feature_importances_", we found the important features for each class. Here, the importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature, which is also known as the Gini importance.

Using the "feature_importances_" attribute, we printed all the features along with their values and selected those with a value (indicating their importance) above 0.02 for each class. We ended up having 30 important features (see the image below).

```
['Households: $25,000 to $29,999_zscore', 'Households: $35,000 to $39,999_zscore', 'Number of
Subway Stations_zscore', 'Total Population: Male_zscore', 'Households: $10,000 to $14,999_zsc
ore', 'Total Population: Female: 75 to 84 Years_zscore', 'Workers 16 Years and over: Public t
ransportation (Includes Taxicab)_zscore', 'Households: $200,000 or More_zscore', 'Population
Age 18 to 64 for whom poverty status  is determined: Living in Poverty_zscore', 'Asian 16 Yea
rs Old In Civilian Labor Force: Employed_zscore', 'Total Population: Female: 45 to 54 Years_z
score', 'Households: $20,000 to $24,999_zscore', 'Households: $45,000 to $49,999_zscore', 'Ho
useholds:_zscore', 'Households: $125,000 to $149,999_zscore', 'Total Population: Female: 65 t
o 74 Years_zscore', 'Households: $100,000 to $124,999_zscore', 'Households: $15,000 to $19,99
9_zscore', 'Total Population: Female_zscore', 'Workers 16 Years and over: Walked_zscore', 'Ar
ea (Land)_zscore', 'Population Age 18 to 64 for whom poverty status  is determined:_zscore',
'Black or African American 16 Years Old In  Civilian Labor Force: Unemployed_zscore', 'Total
Population: Male: 55 to 64 Years_zscore', 'Average household income (In 2013 Inflation Adjust
ed Dollars)_zscore', 'White 16 Years Old In Civilian Labor Force: Unemployed_zscore', 'Total
Population: Female: 35 to 44 Years_zscore', 'Households: Less than $10,000_zscore', 'Total Po
pulation: Male: 45 to 54 Years_zscore', 'Some Other Race 16 Years Old In Civilian Labor Force
: Employed_zscore', 'Households: $50,000 to $59,999_zscore', 'Area Total:_zscore', 'Median Ag
e:_zscore']
```

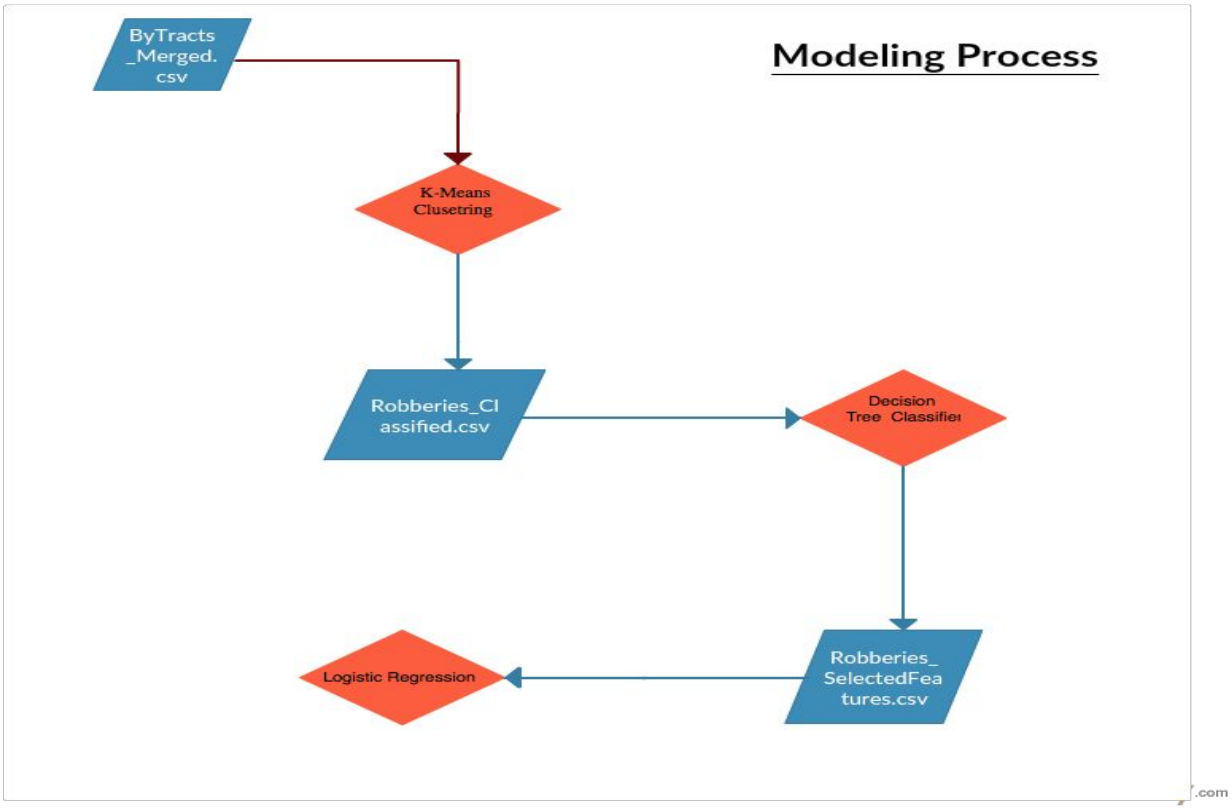
Important Features from Decision Tree Classifier

Step 3: Data Mining (Logistic Regression, Naive Bayes, Decision Tree, SVM)

Now we have our further refined data, which has 30 most important features and a target variable and ready to be mined. We tried to fit our data on four different models. Since this is a multi-classification problem, we chose to use Decision Tree, Logistic Regression, Support Vector Machine, and Naive Bayes. We used sklearn's

label_binarizer to label our target variable as 0,1,2 and also used sklearn's *OneVsRestClassifier* to deal with multi-classification. We found ROC curve and AUC for all the models.

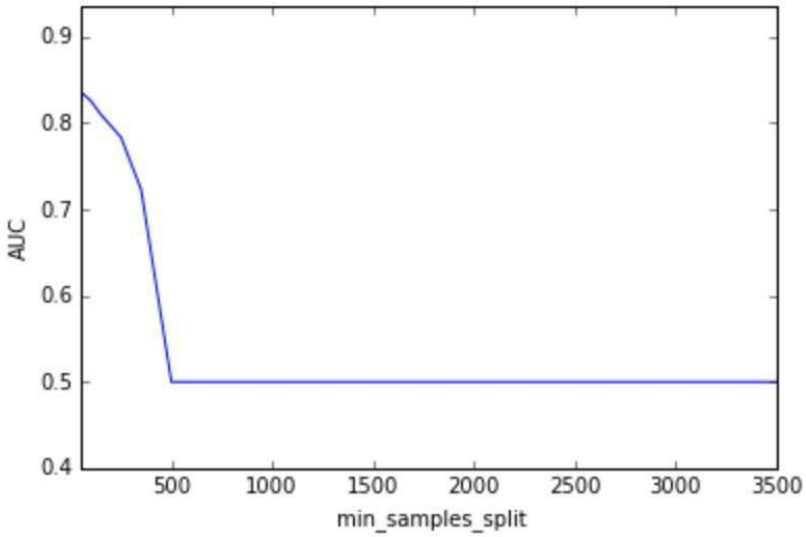
Below is a summary of our data mining process.

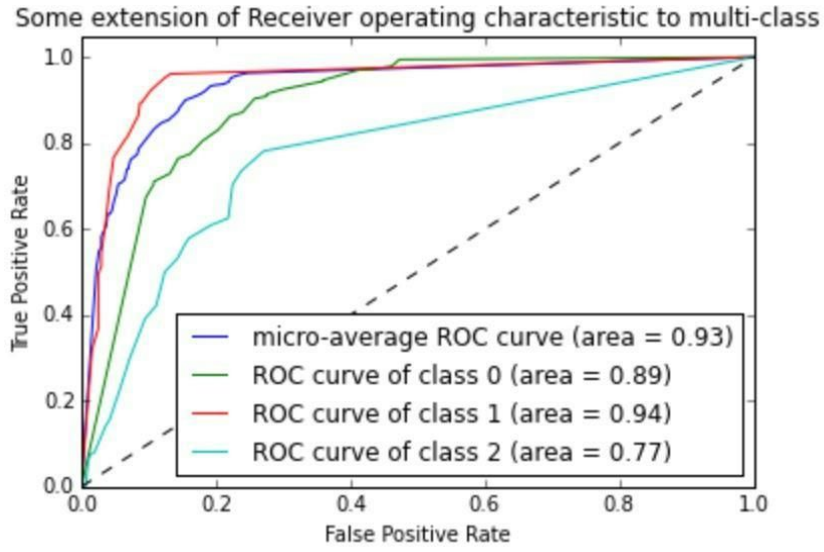


Evaluation:

We used holdout data for training and testing (75% training and 25% testing), and our evaluation for all the above models uses ROC curve and AUC. We hope to find the best model by comparing the ROC curve and the AUC across different models.

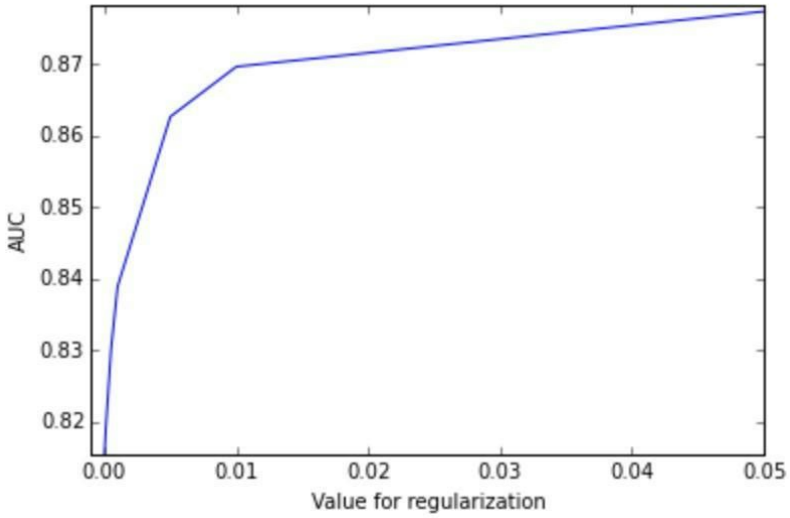
Decision Tree

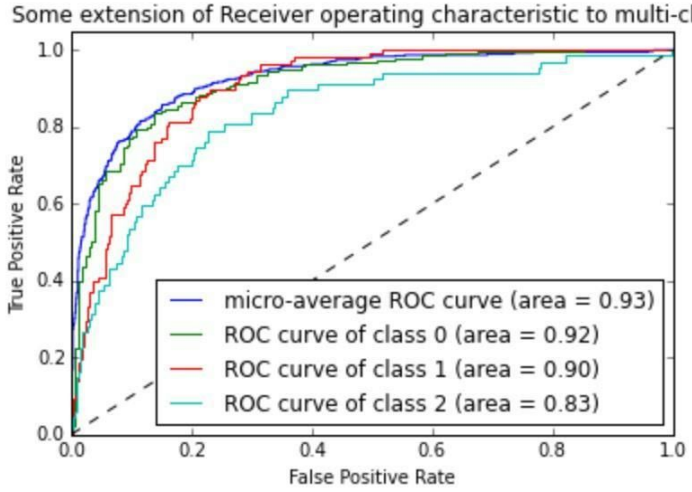




For the Decision Tree model, we found the optimal value for min_samples_split using 3-fold cross-validation and AUC. Based on the first figure, we decided to take min_samples_split = 50. The model has a good predictive performance for the first two classes (low and medium). However, its performance is significantly lower for the third class (high).

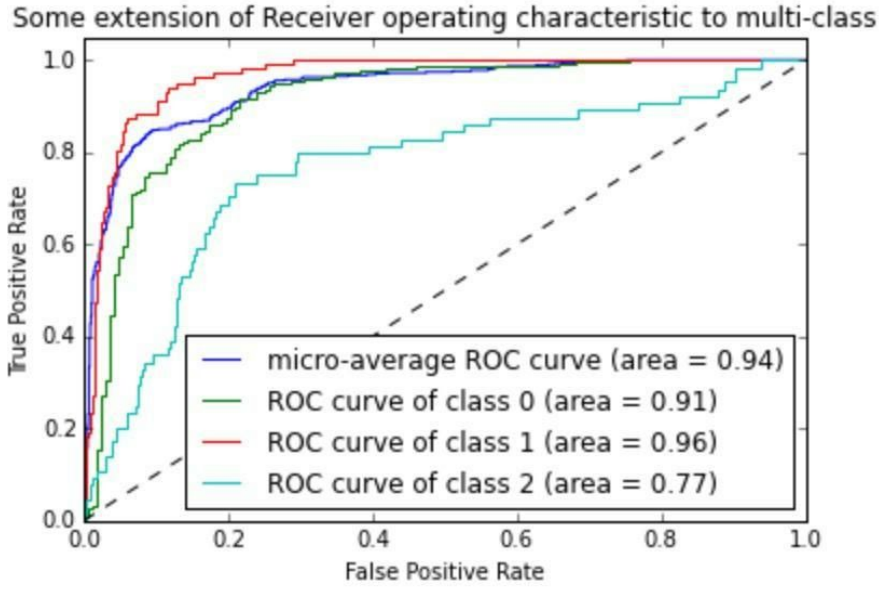
Logistic Regression





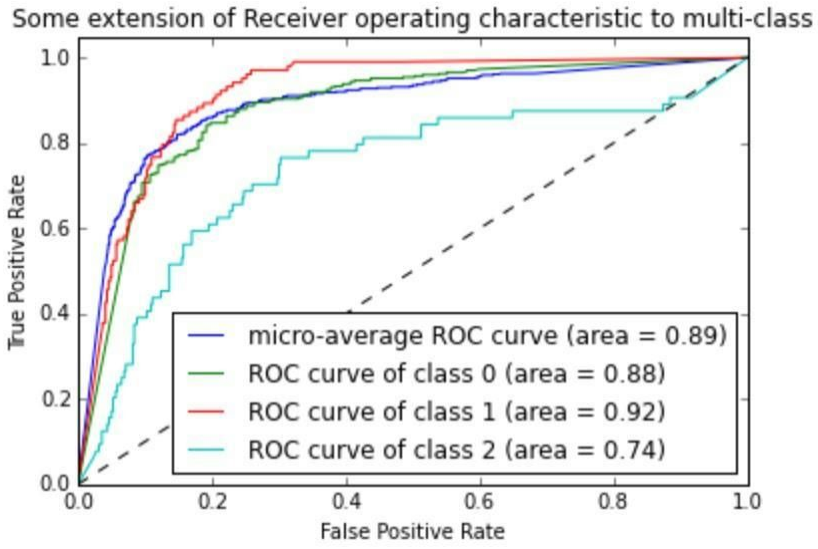
Similarly to the Decision Tree model, we found the optimal value for regularization (C). In this case, we decided to use C= 0.01 because greater values do not have significant improvement on the AUC. The model has a good performance for all three classes. The selection of the threshold would be based on cost-and-benefit matrices.

SVM



The SVM model has a good predictive performance when the threshold gives a fpr around 0.22. After that the model does not have significant improvement in terms of tpr. This model also shows lower predictive performance for the class number 2 (high), and there is overfitting for the class number one (medium). That is probably because of the selection of the important features.

Naive Bayes



We decided to implement the Naive Bayes model in order to have a baseline model. As it turns out, overall, this model has the worst predictive performance.

Deployment:

At any point in future, with the relevant features, one could use the SVM model to find whether their area comes under high, low or medium risk area for robberies. For instance, in a scenario where police officers want to find areas with high risks of robberies, logistic regression which performs well in class 2 (high risk) can be used to identify high risk areas. While this is done with data of New York City, as the features we considered are common across all states and cities, potentially we could also apply the model in places outside New York City, although some extra fine-tuning may be expected.

Review

1. Challenge of having multiple datasets from different sources

We did not have data for the same time range across different datasets. For socio-economic data, we had 5-year average for 2009-2013, and our robbery data used was from 2015 (we could not find crime data for 2013, and there is no American Community Survey data for 2015 available yet). So in this study, we are assuming that the socio-economic factors have not changed much. Of course, for future studies, it would still be more ideal if we could have data for the same time range across all datasets.

2. Challenge of not having enough datasets that we wished for

In a perfect world, if we could have datasets for more factors that we mentioned at the beginning (and also thought of along the way), our models would be even more thoughtful.

3. Dealing with NaN's

Since the amount of data that we have is 160 X 2200, it obviously had some empty cells. The sklearn models couldn't handle the NaN, so we had to skew the data to deal with it.

4. Challenge of multi-class logistic regression and ROC curve

In our class, we mainly dealt with binary classification. In the case of our project, we have a 3-class classification, so it took a while to figure out how to run a multi-class logistic regression. Same with making an ROC curve for a multi-class classification.

5. Challenge of understanding the result of clustering

We experimented a number of times, when initially we had hoped to see what (unexpected and previously unknown) information the unsupervised clustering could provide us in general. At one point, we run clustering with all data (yes, everything), and it was overwhelming and we were at loss in interpreting the clusters generated. In the future, maybe if we know how to generate cluster descriptions with supervised learning, we can make even better use of clustering.

6. Cost and benefit

Had we known the costs associated with robberies, for example, average monetary loss of a robbery and cost of wrongly predicting a robbery for the police officers or for a community, we could have used the probabilities from our model to find a threshold. A specific threshold would be helpful for getting an Expected Value, which could be of great use for people who deploy the model.

Appendix

(These will also be emailed as one zip file along with our report.)

1. Screenshot of the Google API code:

itp_parser.py | GoogleMapsLatLong.java

```java
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;

public class GoogleMapsLatLong {
    public static String API_KEY = "AIzaSyAUHZGuA1L-SE3h3bjAWRobN8DX33Qj-4A";

    public static String[] getLatLong(String address) throws UnsupportedEncodingException, IOException{
        String google = "https://maps.googleapis.com/maps/api/geocode/xml?key="+API_KEY + "&address=";
        String charset = "UTF-8";
        String userAgent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_1) AppleWebKit/601.2.7 (KHTML, like Gecko) Version/9.0.1 Safari/601.2.7"; // Change this to your company

        Document links = Jsoup.connect(google + URLEncoder.encode(address, charset)).userAgent(userAgent).ignoreContentType(true).get();
        String status  = links.getAllElements().select("status").toString();
        if(status.toLowerCase().contains("zero_results"))
            return new String[]{};

        String googleAddress = links.getAllElements().select("formatted_address").first().toString().replace("<formatted_address>","").replace("</formatted_address>", "").replac
        String lat = links.getAllElements().select("lat").first().toString().replace("<lat>","").replace("</lat>", "").replace("\n", "").replace(" ", "");
        String lng = links.getAllElements().select("lng").first().toString().replace("<lng>","").replace("</lng>", "").replace("\n", "").replace(" ", "");

        return new String[]{googleAddress,lat,lng};
    }

    public static void main(String[] args) {
        File subwayFile = new File("/Users/Kavitha/Downloads/DOITT_SUBWAY_ENTRANCE_01_13SEPT2010.csv");
        String subwayStreetName = null;
        BufferedReader brSubway =null;
        FileReader reader = null;
        int count=0;
        PrintWriter writer =null;
        try {
            reader = new FileReader(subwayFile);
            brSubway = new BufferedReader(reader);
            writer= new PrintWriter("/Users/Kavitha/Downloads/Subway_Entrance_LatLong.csv", "UTF-8");
            while((subwayStreetName=brSubway.readLine())!=null)
            {
                Thread.sleep(500);
                String[] values = subwayStreetName.split(",");
                String streetName = values[1];
                String stName = streetName.trim();
                String[] googleLatLong = getLatLong(stName);
                if(googleLatLong.length>0){
                    String write = ++count+ ","+subwayStreetName + ","+"\"" + googleLatLong[0]+ "\""+ "," + googleLatLong[1]+ "," + googleLatLong[2];
                    System.out.println(write);
                    writer.write(write+"\n");
                }
                if(count%100==0){
                    writer.flush();
                }
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
```

Line 44, Column 102 Tab Size: 4 Java

2. Screenshot of the code for clustering:

In [10]:
```
k_clusters = 3

model = KMeans(init='k-means++', n_clusters=k_clusters, n_init=10, max_iter=100)
model.fit(data)
clusters = model.predict(data)

# Do some messy stuff to print a nice table of clusters
cluster_listing = {}
for cluster in range(k_clusters):
    cluster_listing['Cluster ' + str(cluster)] = [''] * 109
    where_in_cluster = np.where(clusters == cluster)[0]
    cluster_listing['Cluster ' + str(cluster)][0:len(where_in_cluster)] = data.index[where_in_cluster]

# Print clusters
#pd.DataFrame(cluster_listing).loc[0:np.max(np.bincount(clusters)) - 1,:]

#to make array size equal
clustered_data = pd.DataFrame({k : pd.Series(v) for k, v in cluster_listing.iteritems()})
```

In [ ]:
```
clustered_data
```

In [11]:
```
#Understanding the cluster
pd.set_option('display.max_columns', None)
cluster0 = clustered_data["Cluster 0"].tolist()
outputcluster0 = data.loc[cluster0]
outputcluster0
output_data = outputcluster0.describe().transpose()
sum_data = outputcluster0.sum()
output_data
#sum_data
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max
ROBBERIES_zscore	444	0.170447	0.644458	-0.841171	-0.376364	0.088442	0.553249	2.257540
Population Density (per sq. mile)_zscore	444	1.450291	0.822224	0.250631	0.818538	1.269794	1.870429	4.860918

3. Screenshot of the code for Decision Tree Feature Analysis:

```
from sklearn.metrics import roc_curve, auc

%matplotlib inline

data = pd.read_csv("Data/Robberies_classified.csv")

Y= data['crime_classification']
data = data.drop(['crime_classification'], axis=1)
cols = list(data.columns)
i = 0
Y = label_binarize(Y, classes=[0, 1, 2])
n_classes = Y.shape[1]
for col in cols:
    col_zscore = col + '_zscore'
    i = i+1
    data[col_zscore] = (data[col] - data[col].mean())/data[col].std(ddof=0)

data = data[data.columns[i:]]
X= data
X=X.fillna(0)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.75)

# Learn to predict each class against the other
#classifier = OneVsRestClassifier(LogisticRegression())
classifier = OneVsRestClassifier(DecisionTreeClassifier(criterion="entropy"))
classifier.fit(X_train, Y_train)
```

In [ ]:
```
taImportance= DataFrame(classifier.estimators_[1].feature_importances_, columns = ["Imp"], index = X.columns).sort(['Imp
= dataImportance[dataImportance.Imp > 0.01]
taImportance= DataFrame(classifier.estimators_[0].feature_importances_, columns = ["Imp"], index = X.columns).sort(['Imp
= dataImportance[dataImportance.Imp > 0.01]
taImportance= DataFrame(classifier.estimators_[0].feature_importances_, columns = ["Imp"], index = X.columns).sort(['Imp
= dataImportance[dataImportance.Imp > 0.01]
y.append(x)
y.append(z)
set = set( y.index.values)
```

In [ ]:
```
X = X[list(myset)]
```

4. Screenshot of all the models:

In [4]:

```
#Logistic Regression multiclass

classifier = OneVsRestClassifier(LogisticRegression(C=0.01))
y_score = classifier.fit(X_train, Y_train).decision_function(X_test)
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(Y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(Y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot ROC curve
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label='ROC curve of class {0} (area = {1:0.2f})'
                                   ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Some extension of Receiver operating characteristic to multi-class')
plt.legend(loc="lower right")
plt.show()
```