Econ 900 - Final Exam - Predicting Burglaries in Chicago - Matt Cronin

May 3, 2019

0.1 Objective:

The objective of this task was to use crime data from Chicago's Data Portal (Crimes - 2001 to present.csv) to train a supervised learning model that could predict the occurrence of a crime in Chicago.

In [1]:
```
from sklearn import linear_model
import pandas as pd
import random

# The data to load
f = "crime_data/Crimes_-_2001_to_present.csv"

# Code to load random sample of .csv
num_lines = sum(1 for l in open(f))
# Sample size - in this case ~20% - Anymore than this I run into memory issues
size = int(num_lines / 5)
skip_idx = random.sample(range(1, num_lines), num_lines - size)

# Read the data
## According to Chicago Data Portal, 'Community Areas' is current.
## I am going to remove 'Community Area'.
drops = ['Case Number', 'Block',  'Description',  'Location Description',
                    'Updated On', 'Community Area']
data = pd.read_csv(f, skiprows=skip_idx).drop(drops, axis=1)
data.columns = ['ID','Date','IUCR',
                     'Primary_Type','Arrest','Domestic', 'Beat', 'District',
                     'Ward', 'FBI_Code', 'X_Coordinate', 'Y_Coordinate',
                     'Year', 'Latitude', 'Longitude', 'Location',
                     'Historical_Wards', 'Zip Codes', 'Community_Areas',
                     'Census_Tracts', 'Wards', 'Boundaries_ZIP',
                     'Police_Dist', 'Police_Beats']
print(data['Primary_Type'].value_counts())
BURGLARY = data[data.Primary_Type == 'BURGLARY']
```

```
THEFT                                289054
BATTERY                              250844
```

```
CRIMINAL DAMAGE                      156058
NARCOTICS                            143367
ASSAULT                               85091
OTHER OFFENSE                         84997
BURGLARY                              78550
MOTOR VEHICLE THEFT                   63719
DECEPTIVE PRACTICE                    54282
ROBBERY                               51717
CRIMINAL TRESPASS                     39092
WEAPONS VIOLATION                     14668
PROSTITUTION                          13758
PUBLIC PEACE VIOLATION                 9574
OFFENSE INVOLVING CHILDREN             9375
CRIM SEXUAL ASSAULT                    5585
SEX OFFENSE                            5216
INTERFERENCE WITH PUBLIC OFFICER       3207
GAMBLING                               2915
LIQUOR LAW VIOLATION                   2788
ARSON                                  2237
HOMICIDE                               1954
KIDNAPPING                             1342
INTIMIDATION                            843
STALKING                                701
OBSCENITY                               108
CONCEALED CARRY LICENSE VIOLATION        80
PUBLIC INDECENCY                         35
NON-CRIMINAL                             32
OTHER NARCOTIC VIOLATION                 23
HUMAN TRAFFICKING                        13
RITUALISM                                11
NON - CRIMINAL                            9
NON-CRIMINAL (SUBJECT SPECIFIED)          1
DOMESTIC VIOLENCE                         1
Name: Primary_Type, dtype: int64
```

The code above is part of the **eda.py** program. This program limits the crime data (Crimes_-_2001_to_present.csv) to just a random 20% sample of the original rows. Working with any larger of a sample resulted in memory issues. The next step was to get a sense of the types of crimes contained in these data. By looking at the "Primary Type" column I could see a standardized description of the type of crime each observation in the data represents. The data contain a wide array of offenses. To build a model that could predict every type of crime would require many different features. Thus, I felt it was appropriate to limit this exercise to just looking at one type of crime. I chose Burglary since it is a common offense in Chicago, and I felt I could develop an appropriate set of model features for predicting this type of crime. These sample data were then limited to just burglaries and output to a csv file.

0.2 Features to Predict Burglary:

The program **burglaries_predictor_variables.py** takes the dataset described in the last step and adds some columns that will be used as features for our model:

0.2.1 1. Time Related Features

- For each observation, how much time has elapsed since the last burglary in the same community area (in days)
- Dummy variable to indicate working hours of the day (8am - 7pm)
- Dummy variable to indicate colder (winter) months (October to March)
- Dummy variable to indicate work days of the week (M - F)

0.2.2 2. Geographical Features (Community Area Features)

- Average amount of time that elapses between burglaries within a community area (in days)
- Dummy variables to indicate each community area
- Total number of burglaries in that community area
- Number of affordable housing units within that community area
  - The data on affordable housing units by community area were obtained from here: https://data.cityofchicago.org/Community-Economic-Development/Affordable-Housing-Units-by-Community-Area/yvj4-y3fb

0.2.3 3. Law Enforcement Features

- Number of police beats (from crimes dataset)
- Haversine distance to nearest police station (in kilometers)
  - The latitude and longitude for police station locations in Chicago were obtained from: https://data.cityofchicago.org/Public-Safety/Police-Stations/z8bn-74gv
  - Haversine distances were calculated from each burglary to each police station. Then for each burglary, the minimum of the distances calculated were used as the distance to the nearest police station.

0.3 Summary Statistics of Training Data:

Summary statistics of the feature variables are described below. For the sake of space, only 2 of the 77 community variable dummies are displayed on the table.

```
In [11]: data = pd.read_csv("training_data_burglary.csv")
         data.iloc[:,np.r_[32:37,111:115]].describe()
```

```
Out[11]:        working_hours        winter      work_day    10.0_area     11.0_area  \
         count   77543.000000  77543.000000  77543.000000  77543.00000  77543.000000
         mean        0.580426      0.479076      0.234283      0.01514      0.004810
         std         0.493492      0.499565      0.423553      0.12211      0.069189
         min         0.000000      0.000000      0.000000      0.00000      0.000000
         25%         0.000000      0.000000      0.000000      0.00000      0.000000
         50%         1.000000      0.000000      0.000000      0.00000      0.000000
```

```
75%         1.000000      1.000000      0.000000      0.00000      0.000000
max         1.000000      1.000000      1.000000      1.00000      1.000000

        CA_AVG_DELTA      CA_COUNT      LI_COUNT        police
count   77543.000000  77543.000000  77543.000000  77543.000000
mean        6.591533   1612.864643      5.528352    157.484712
std         6.602893    901.593887      7.953746     80.783302
min         1.997870     65.000000      0.000000      1.000000
25%         2.946607    808.000000      1.000000     89.000000
50%         4.333036   1541.000000      2.000000    174.000000
75%         8.217308   2267.000000      6.000000    227.000000
max        97.264371   3341.000000     33.000000    277.000000
```

0.4 Random Forest Model:

I wanted to build a model that could predict the occurrence of a burglary within a certain future timeframe. More specifically I wanted to identify whether a burglary will happen in a community area within the next two weeks. The outcome variable is a binary outcome, taking on 1 if a burglary happened within 14 days of the last burglary in the community area, or 0 otherwise. Since the outcome is categorical, I would need to use a classifier algorithm. I chose to use a random forest model for several reasons.

The first reason I chose to use a random forest model was that it is easy to optimize the feature selection of the model. My approach, as shown below, would be to first run the random forest classifier with all the features I described in the previous section, score the algorithm, identify the most important features, then repeat the same classification algorithm using only the most important features, with gini-importance above a certain threshold.

The second reason I chose to use a random forest is that it resolves the issues of using a "greedy algorithm", like with implementing a decision tree, as we are using many decision trees to prevent our results from being biased by one decision tree outcome. In addition to this we are not susceptible to overfitting issues.

0.4.1 Model With All Features:

```
In [1]: from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import confusion_matrix
        from sklearn.feature_selection import SelectFromModel
        import pandas as pd
        import numpy as np

        n = 90
        #
        data = pd.read_csv("training_data_burglary.csv")
        Training_Target = np.where((data['day_delta'] < 14), 1, 0)
        Training_Data = data.iloc[:,np.r_[32:115]]

        data_training, data_test, target_training, target_test = \
```

```
        train_test_split(Training_Data, Training_Target, test_size = 0.25, random_state=1)
    random_forest_machine = RandomForestClassifier(n_estimators=n)
    random_forest_machine.fit(data_training, target_training)
    predictions = random_forest_machine.predict(data_test)
    print(accuracy_score(target_test, predictions))

    cm = confusion_matrix(target_test,predictions)
    confusion_matrix = pd.DataFrame(
            cm,
            columns = ['Predict No', 'Predict Yes'],
            index = ['True No', 'True Yes']
    )
    print(confusion_matrix)
```

```
0.8859889399968991
          Predict No  Predict Yes
True No          354         1841
True Yes         365        16789
```

The first model scored ~0.88598. As we can see with the model with all features, our random forest algorithm correctly predicted the occurrence of a burglary 16,788 times and failed to predict a burglary only 340 times. However, when burglaries didn't happen it tended to make a type I error and predict a burglary would be present, when in fact a burglary did not occur within the 14 day timespan.

0.4.2 Feature Importance:

Checking the feature importance of all the variables used in the first model shows that a lot of the features weren't contributing much to the decision-making process for our algorithm. At first glance only a small number of features such as average time between burglaries (*CA_AVG_DELTA*) have a relatively high importance. The steps below will describe what was done to limit to only important features of the model.

```
In [3]: ### Train Model to Select Most Important Features and Refine The Model
        # Fine Best Features
        select_features = SelectFromModel(random_forest_machine, threshold = 0.1)
        select_features.fit(data_training, target_training)
        x_refined_train = select_features.transform(data_training)
        x_refined_test = select_features.transform(data_test)

        # Print Top Features
        important_features = select_features.get_support()
        feature_name = Training_Data.columns[important_features]
        print(feature_name.value_counts())
```

```
CA_AVG_DELTA    1
CA_COUNT        1
police          1
```

```
dtype: int64
```

The above code does the following: 1) SelectFromModel detects the features of our previous model that have a feature importance above 0.1 2) The training and test data are then transformed to only take the features of the required importance 3) Print a list of the top features * Police Beat Count * Community Burglary Count * Community Average Time Between Burglaries

Below we will re-run the random forest classifier with only these three features.

0.4.3 Refined Model:

```
In [4]: #Train and Test New Model
        refined_forest_machine = RandomForestClassifier(n_estimators=n)
        refined_forest_machine.fit(x_refined_train, target_training)
        refined_predictions = refined_forest_machine.predict(x_refined_test)
        print(accuracy_score(target_test, refined_predictions))

        from sklearn.metrics import confusion_matrix
        cmr = confusion_matrix(target_test,refined_predictions)
        confusion_matrix = pd.DataFrame(
                cmr,
                columns = ['Predict No', 'Predict Yes'],
                index = ['True No', 'True Yes'])

        print(confusion_matrix)
```

```
0.8917256705772908
          Predict No  Predict Yes
True No          399         1796
True Yes         299        16855
```

The accuracy score for the refined model improved marginally to ~0.89172. The confusion matrix however indicates the model became better at predicting burglaries when they actually happened, but would still overpredict a burglary happening when it did not.