In-depth Analysis (Machine Learning)

If a model accuracy is higher than 50%, it means that using ML to predict crime type is better than random chance.
Jupyter notebook:
https://github.com/elizabeamedalla/Capstone-1/blob/master/In-depth%20Analysis%20(Machine%20Learning).ipynb

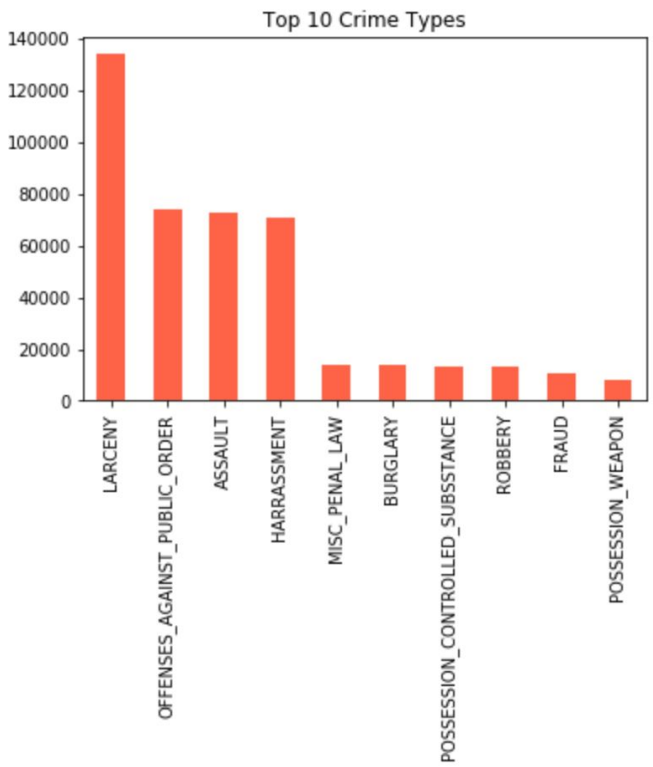
Procedure

What we will do is focus on the top 4 most occurring crimes and try to use machine learning to predict those. We will use 70,000 samples from each of the top 4 crimes then split the data into training and testing data.

Then we will fit a logistic regression and random forest classifier on the data, investigate the accuracy, then perform hyperparameter tuning to compare which classifiers achieve better predictions on what crimes may occur in different areas of NYC.

Inspecting the Data

Since we are trying to use machine learning to predict the crime type, we visualized the number of occurrences of each crime type in the data set to see if it was balanced.



When looking at the top 10 crimes, we can see that there is a heavy skew towards the top 4 crimes - the top 4 crimes have greater than 70,000 occurrences, while the rest of the crimes have less than 15,000 occurrences. We don't have enough uniform data to make a prediction on all the classes.

We can conclude here that we will have to resample our data. We can try undersampling or oversampling, but undersampling would risk losing important data, while oversampling would risk overfitting.

What we decided on doing instead, is to simply try to predict the top 4 most occurring crimes, and resample our dataset so that it only contains 70,000 rows each for the top 4 crimes. This will allow for predicting on a more uniform dataset with less data loss.

Feature Engineering

In this step, we inspect and select different features to use for training the dataset and predicting the outcome with.

At first, I chose a handful of features such as BOROUGH, DISTRICT, CRM_ATPT_CPTD_CD, PREM_TYP_DESC and VIC_SEX but my accuracy of Logistic Regression was only about

20%. I realized I needed to use more features to get greater accuracy. After trial and error, I chose the following features:

features = ['BOROUGH', 'DISTRICT', 'MONTH', 'VIC_AGE_GROUP', 'VIC_RACE','VIC_SEX', 'SUSP_AGE_GROUP', 'SUSP_RACE', 'SUSP_SEX']

These features yielded a testing accuracy of over 50% on each classifier.

One feature I wanted to use was OFFENSE LEVEL, but I chose not to for a few reasons.
1) Because a crime such as theft can be multiple offense levels (Felony or Misdeameanor).
2) Choosing OFFENSE_LEVEL alone gave me 50% accuracy with Logistic Regression, which immediately told me that it is easy to overfit using this feature.

Preparing Data

To prepare my data, first I resampled the data as described above, using only the rows containing the top 4 crimes and pulling only 70,000 rows each. This created a new dataset with 280,000 rows.

I then used the features I defined above to create the X dataframe containing the data I will use to train and predict with. I used the CRIME_TYPE column for my Y dataframe, as the prediction target.

I one-hot encoded all categorical variables in my X dataframe using pd.get_dummies(). I numerically encoded my Y dataframe using pd.factorize(). This is because The Y dataframe needs to be a single column as the target, and one-hot encoding would make it 4 columns.

Finally, I converted my X and Y dataframes into Numpy arrays, and split them into training and testing datasets using train-test-split.

Logistic Regression

Our first classification method was performed with logistic regression. Logistic regression models are able to make predictions through the logistic function.

I wanted to aim for above 50% accuracy scores on both my training and testing data. This means that although it’s not reliable enough, logistic regression is still better than using random chance to predict crimes.

Vanilla Logistic Regression Accuracy

First, I wanted to test out how accurately a basic Linear Regression with default settings would predict on my data. I ended up with a training accuracy of 0.532 and a testing accuracy of 0.537. Since the scores are similar, I can assume it is not overfitting on my training data.

Logistic Regression Hyperparameter Tuning

I wanted to tune the hyperparameters to see if I could get a little better accuracy out of my model. I ran a GridSearchCV on the logistic regression model and found out that using an l1 penalty with a liblinear solver was the best hyperparameters.

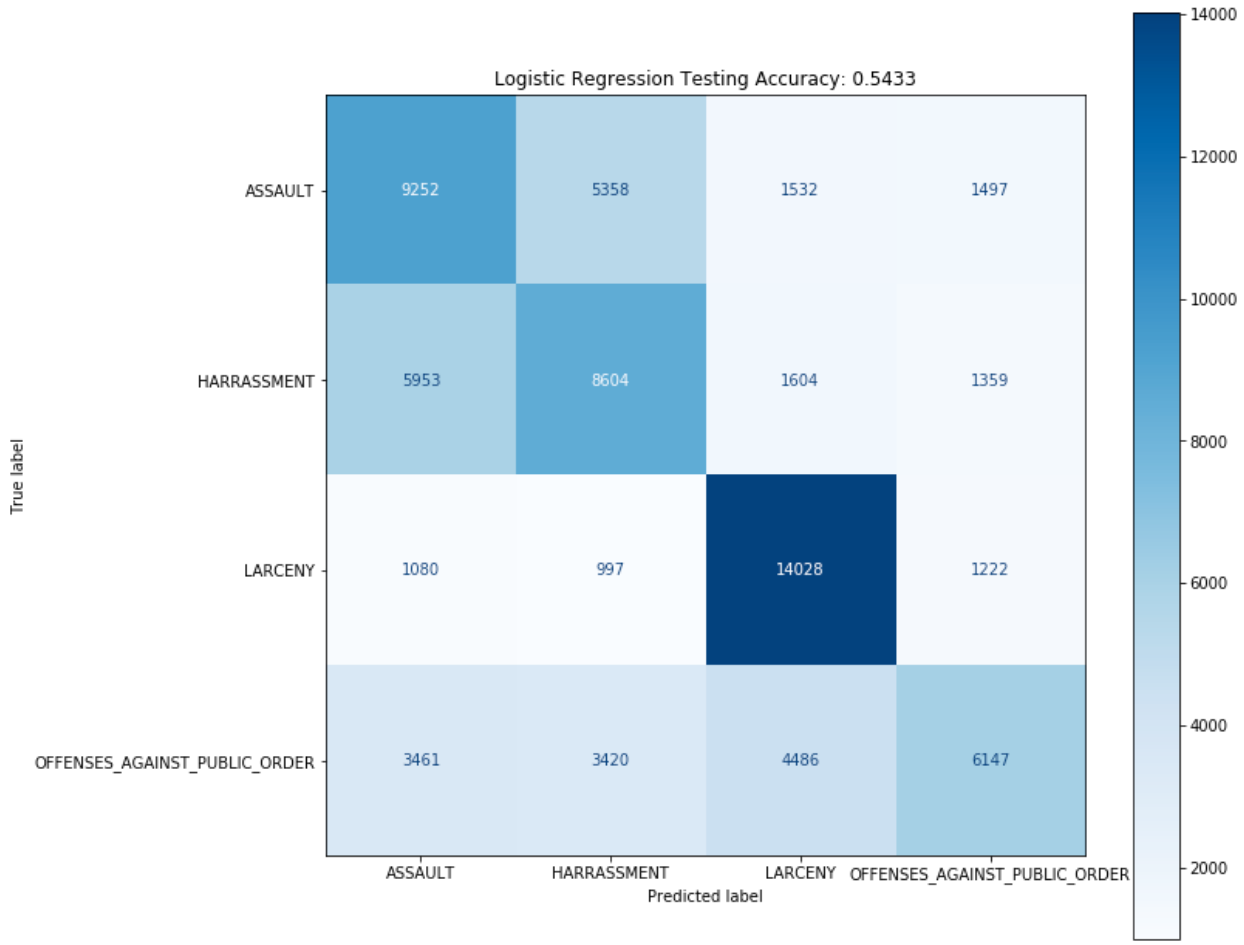
In this analysis, logistic regression was performed with an *l1* penalty in order to include a regularization method that works to prevent overfitting by decreasing the weights of the features to a small number (we initiated 4 features). Additionally, *liblinear* (A Library for Large Linear Classification) can work with *l1* penalties and which is said to minimize the cost function more quickly for large datasets.

Hypertuned Logistic Regression Accuracy

After applying the best estimators using hyperparameter tuning, our scores got higher. I achieved a training accuracy of 0.548 and a testing accuracy of 0.543, which was better than the vanilla Logistic Regression.

Confusion Matrix (Logistic Regression)

Confusion matrix was applied to test the performance of the hypertuned logistic regression model that we made. The confusion matrix shows the True Positives, False Negatives, False Positives, and True Negatives for each class being predicted and its true result.



The confusion matrix showed a diagonal trend, meaning that the features are not biased in classifying crimes. The diagonal trend indicates that in the majority of cases, when crimes are misclassified, they were incorrectly classified into the nearby crime. Since we have 9 features and some are somewhat related, the misclassification was already expected. You can see some misclassifications in ASSAULT and HARASSMENT.

Random Forest

This function includes parameters for stratified cross-fold validation and supports code for hyperparameter tuning for a random forest model. Because we are working with a multi-classification problem, we evaluate our logistic regression models with one-vs-one classification.

Vanilla Random Forest Accuracy

I wanted to test out how accurately a basic Random Forest with default settings would predict on my data.

I ended up with a training accuracy of 0.853 and a testing accuracy of 0.508. It looks like the training accuracy is drastically greater than the testing accuracy, which indicates overfitting. We can fix this using hyperparameter tuning.

Random Forest Hyperparameter Tuning

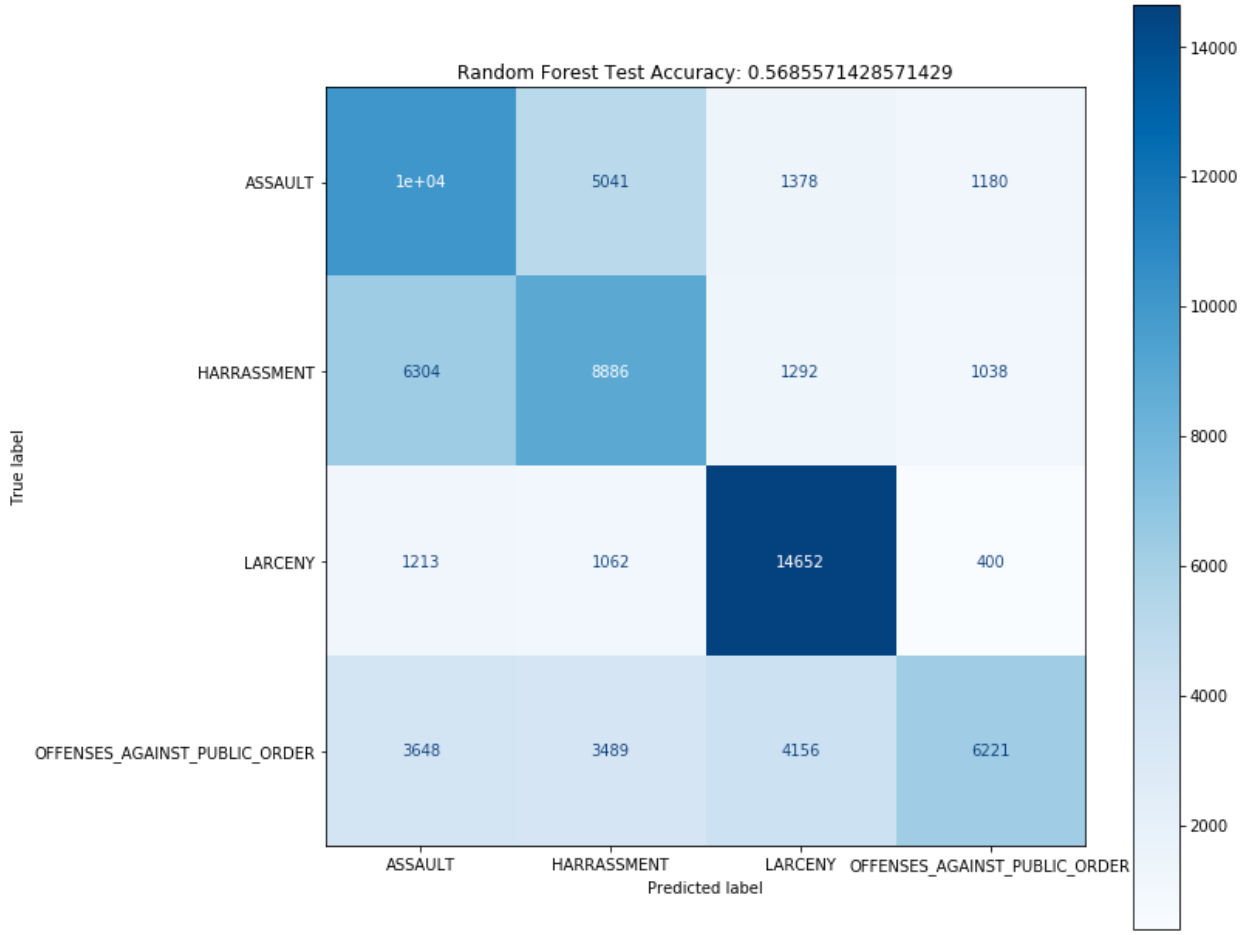
Hyperparameter tuning our Random Forest model will allow us to try to prevent the overfitting happening in the default Random Forest model. I ran a GridSearchCV on the Random Forest model and found out that the best hyperparameters were **criterion='entropy', max_depth=15, min_samples_leaf=10, n_estimators=70, random_state=12345.**

In this analysis, **entropy** calculates the homogeneity of the sample data. If the sample data is completely homogeneous the entropy is zero and if the sample is equally divided it has an entropy of one. The **max_depth**, 15 is the maximum number of features random forest considers to split a node. The opposite does in our min_samples_leaf, the minimum number of samples required to split an internal node. Another significant estimator is **n_estimators** hyperparameter, which is just the number of trees the algorithm builds before taking the maximum voting or taking the averages of predictions, here, we'll choose 70 random trees.

Hypertuned Random Forest Accuracy

After applying the best estimators using hyperparameter tuning, the testing accuracy got higher and the overfitting was fixed. I achieved a training accuracy of 0.5863 and a testing accuracy of 0.5695, which showed that the model was no longer overfitting to the training data.

Confusion Matrix (Random Forest)



The confusion matrix showed a diagonal trend, meaning that the features are not biased in classifying crimes. The diagonal trend indicates that in the majority of cases, when crimes are misclassified, they were incorrectly classified into the nearby crime. Since we have 9 features and some are somewhat related, the misclassification was already expected. You can see some misclassifications in ASSAULT and HARASSMENT.

Random Forest Hyperparameter Tuning

Found best parameters to be:

rf_best = RandomForestClassifier(criterion='entropy', max_depth=15, min_samples_leaf=10, n_estimators=70, random_state=12345)

In this analysis, **entropy** calculates the homogeneity of the sample data. If the sample data is completely homogeneous the entropy is zero and if the sample is equally divided it has an entropy of one. The **max_depth**, 15 is the maximum number of features random forest considers to split a node. The opposite does in our min_samples_leaf, the minimum number of samples required to split an internal node. Another significant estimator is **n_estimators** hyperparameter, which is just the number of trees the algorithm builds before taking the maximum voting or taking the averages of predictions, here, we'll choose 70 random trees.

Hypertuned Random Forest Accuracy

Training Accuracy: 0.5857476190476191
Testing Accuracy: 0.5746428571428571

Looks like the accuracy score got more consistent when we applied the best parameters from our hyperparameter tuning.

After comparing my accuracy scores in both logistics regression and random forest, we have better results if we use random forest than logistic regression. Both models, given the accuracy of more than 50%, will still give us better results in predicting crimes than random choice.

Conclusion

The main goal of the project is to be able to aid the detectives and NYPD to do crime analysis on crime data in NYC in 2019 and to create a model that would help them to predict crimes better.

Both logistic regression and random forest are some of the most popular classification models. But not all classification problems are the same. Applicable parameters must be applied to get accurate scores. This can be done through hyperparameter tuning.