FINAL

December 5, 2023

1 Abstract:

We investigate ensemble learning methods, specifically random forests and boosted trees, to classify crime types utilizing data on incidents reported in Los Angeles County during the year 2020. The primary objective is to develop machine learning models capable of accurately predicting crime classifications based on various features associated with the reported incidents. Using data encompassing different attributes such as location, time, and other relevant factors, we aim to find relationships between these variables to enhance the classification accuracy of criminal activities. We seek to understand the ideal location for police stations in the LA country based on the occurence and frequency of crime. We then compare this to the actual location of LA police stations to understand where the city could place future stations. We also seek to achieve an enhanced understanding of crime patterns, enabling law enforcement agencies to improve resource allocation and planning responses. The findings and insights derived from paper hold the potential to contribute to the field of crime analysis and facilitate more efficient measures for aiding victims within Los Angeles County and beyond.

2 1. Introduction:

Los Angeles County has long been grappling with concerns surrounding its high crime rates. There is a wide variety of forms of criminal activities ranging from property crimes to violent offenses. Combatting this has been a consistent effort within the community and among law enforcement. While existing programs and resources offer support and aid to victims of many crimes, the efficiency of these could be greatly enhanced. Nationwide, many crime resources are in high demand, while others are going unused. The current system, though commendable, faces limitations in its ability to cater to the complex needs of crime victims. There exists a gap in the optimization of these programs, suggesting opportunities for improvement in their responses to distress calls and the following services. Enhancing these systems could better address the diverse needs and circumstances of victims, ensuring more effective aid and facilitating their path towards recovery and rehabilitation.

One way to improve these support systems is by analyzing crime data more systematically. By identifying trends and patterns in this data, it becomes possible to pinpoint areas with higher risks of certain crimes. This information can help in onsite assistance and emergency responses. It can also be used directing the county's resources, money, and services more strategically, making sure they're better suited to the specific needs of people and communities most impacted by crime. Using data from the crimes in LA county that were reported in 2020, we can create machine learning models to predict which areas of the county correlate to which crimes, assisting in the distribution of help resources. This also will help us understand where future police stations could be located.

To analyze and understand these questions and potential we will first clean the data. We will also then look at some algorithms like kmeans, random forests, boosted forests and logistic regression to help us further understand crime classification.

3 2. Data Explanation and Cleaning

In order to do data cleaning we needed to important several packages. These were all imported in the usual way (numpy as np, pandas as pd, etc.)

To begin, we had a dataset of observations on criminal reports from LA county that from 2020. This data set was rather large with over 82k data points. The data has 28 columns as well ranging from information about the time and location of the crime, a crime code (unique identifier of what the culprit was charged with), information about the victim, weapon used and more. Overall, the 28 columns represented different attributes that related to the crime that was reported. In order to avoid ethical issues and ensure that our models were not biased we first removed column attribute that had to with the culprit and defendant that could be unethical to use in classification predictions (race, gender and other similar columns).

Following this, due to the breadth of our data, we were able to drop rows with missing entries in critical columns that would help with classification. We took this approach as much of the data is nominal (like the description of the crime or weapon) making it difficult to replace missing values with the mean or other comparable approaches. After doing this we still had over 67k rows. In addition to entries with missing data, there were several crime types with no more than a few instances. With little data about such rarer offenses, we dropped any report involving a crime that happened less than 1000 times.

Additional data cleaning methods were used and are further mentioned depending on the needs of individual algorithms.

4 3. Ideal Police Station/Resource Locations

Much of our analysis revolves around being able to use components of a crime to predict what kind of a crime occurred. However, before we get to that stage, we first look at crime as a whole in LA. One question that we wanted to answer is to be able to understand the ideal locations for police stations in LA based on the frequency of crimes. This would help ensure that a police station was within a close distance to the areas with the most crime in LA. This would also help reduce the response time for first responders and could help improve safety, medical attention to victimes, and increase the odds of finding the culprit.

In order to do this we wrote a kmeans class to help us identify the *k* ideal locations for a police station. The class is initialized with the *k* number of clusters, a maximum number of iterations, a norm, a tolerance check and normalization boolean.

The class also consists of four main functions that are used to run the kmeans algorithm. This includes a fit, predict, fit_predict and plot function. The fit function computes the cluster centers from random intiial conditions.

```
[ ]: def fit(self, X, y=None):
             """Compute the cluster centers from random initial conditions.
```

```
        Parameters:
            X ((n_samples, n_classes) ndarray): the data to be clustered.
        """
        #set our centers and then normalize if specificed
        self.centers = X[np.random.choice(X.shape[0],self.
↪n_clusters,replace=False)]
        if self.normalize == True:
            self.centers = np.reshape(self.centers/np.linalg.norm(self.
↪centers,axis=1),(-1,1))
        for i in range(0,self.max_iter): #iterate thorugh max iter and create␣
↪the label and new center
                label = np.argmin(np.linalg.norm(X[:,np.newaxis]-self.
↪centers,ord=self.p,axis=2),axis=1)
                new_c = np.array([X[label==z].mean(axis=0) for z in range(self.
↪n_clusters)])
                if np.linalg.norm(new_c-self.centers,ord=self.p) <self.tol: #if␣
↪error is less than tol break
                     break
                self.centers = new_c #set the new center and normalize if␣
↪specified
                if self.normalize == True:
                    self.centers = np.reshape(self.centers/np.linalg.norm(self.
↪centers,axis=1),(-1,1))
        return self #return our object
```

This block is a little dense so we will explain a little more beyond just the code comments. Essentially, we choose random center locations for each of our k clusters. Then the code looks at the distance (based on the specific norm) from each data point to our randomized centers. Each data point is labeled based on the smallest distance from it to all clusters. With each data point labeled we can now shift our centers to better fit the data. We also iterate through this process the number of times we set our max_iter attribute to.

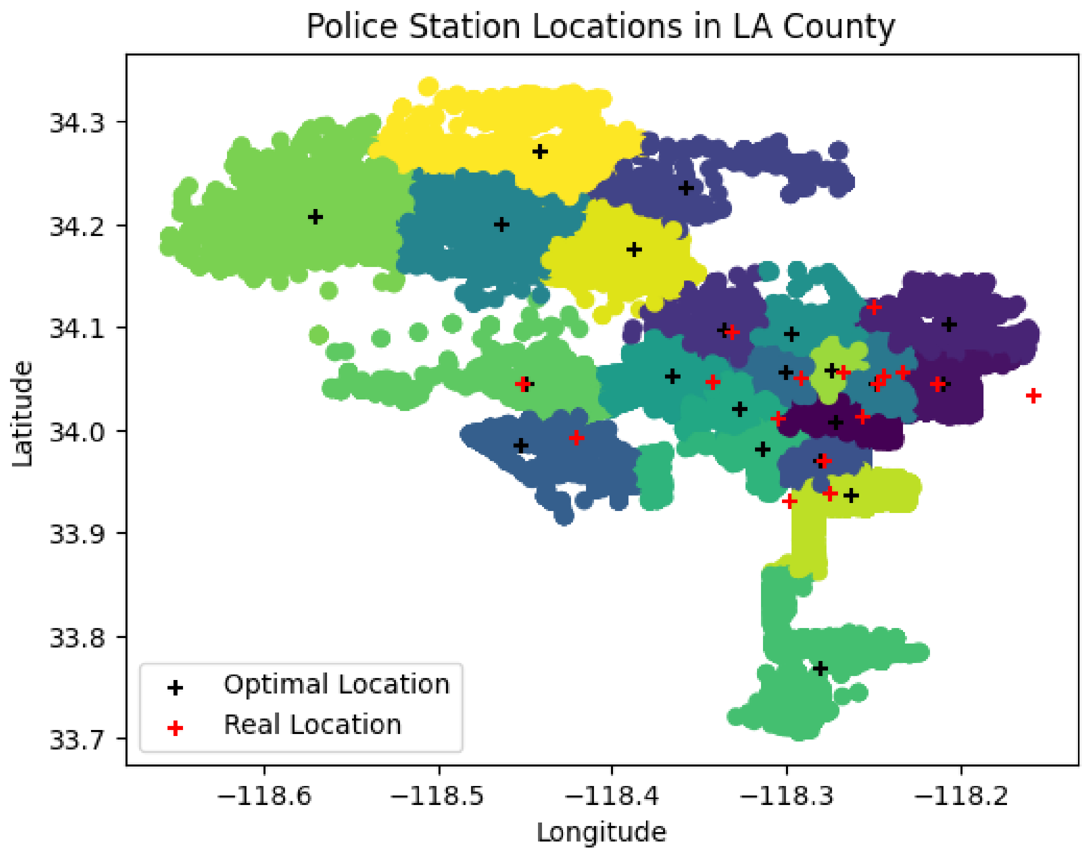
We will not go over the code for predict, fit_predict and plot for brevity's sake. The functions classify each entry of the data on which cluster center it is closest to, return the labels and then plots the data and centers respectively.

Now that we have our kmeans class we can start to examine the ideal locations for police stations. Based on data from LA county they would ideally have 21 police stations in the county. This gives us our ideal k for our kmeans algorithm. The following code shows what we did to get our kmeans graphs:

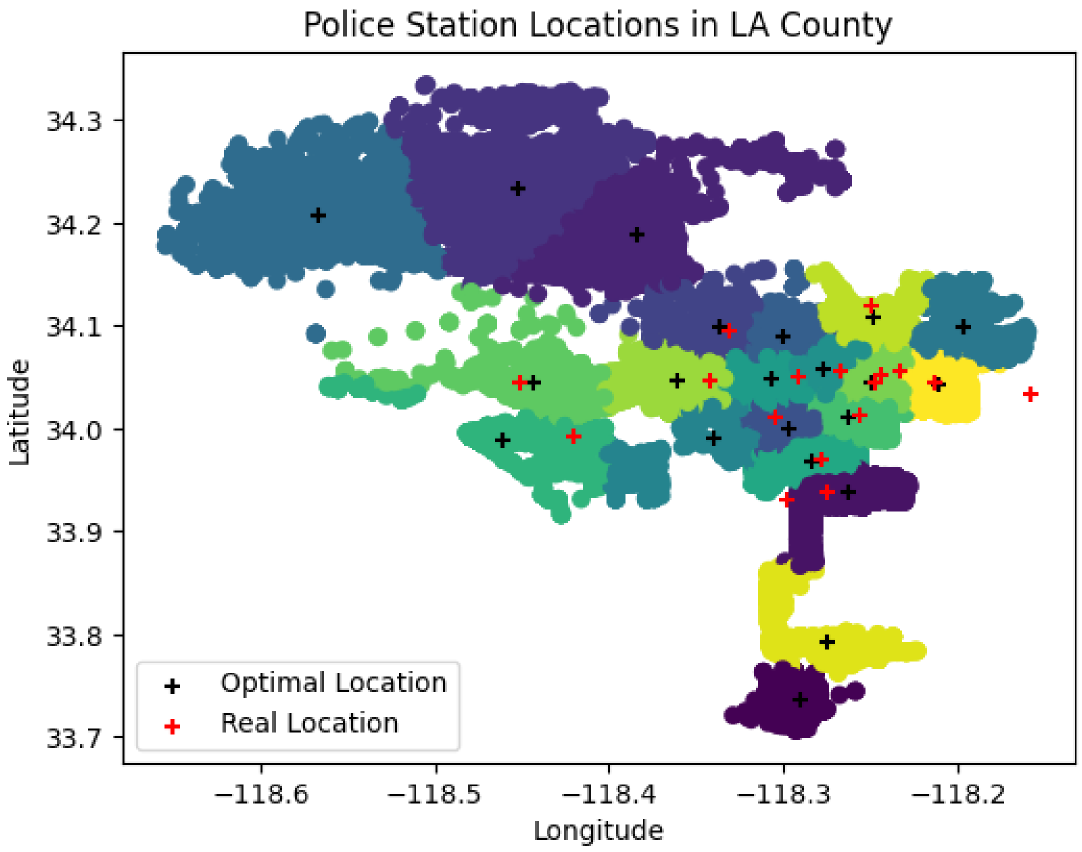
```
[ ]: location = my_data[['LON','LAT']] #get a dataframe with only longitude and␣
      ↪latitude columns
     new_loc = location[~(location == 0).all(axis=1)] #ensure that all entries are␣
      ↪nonzero for both columns
     new_data = new_loc[['LON','LAT']].values #create array with the longitude and␣
      ↪latitude values
```

```
for x in [1,2,np.inf]: #iterate through our different norms
    km = KMeans(n_clusters=21,p=x) #initialize our class, fit it with our data
    km.fit(new_data)
    y = km.predict(new_data) #predict our labels for the data and then use the␣
 ↪plot function
    km.plot(new_data,y)
```

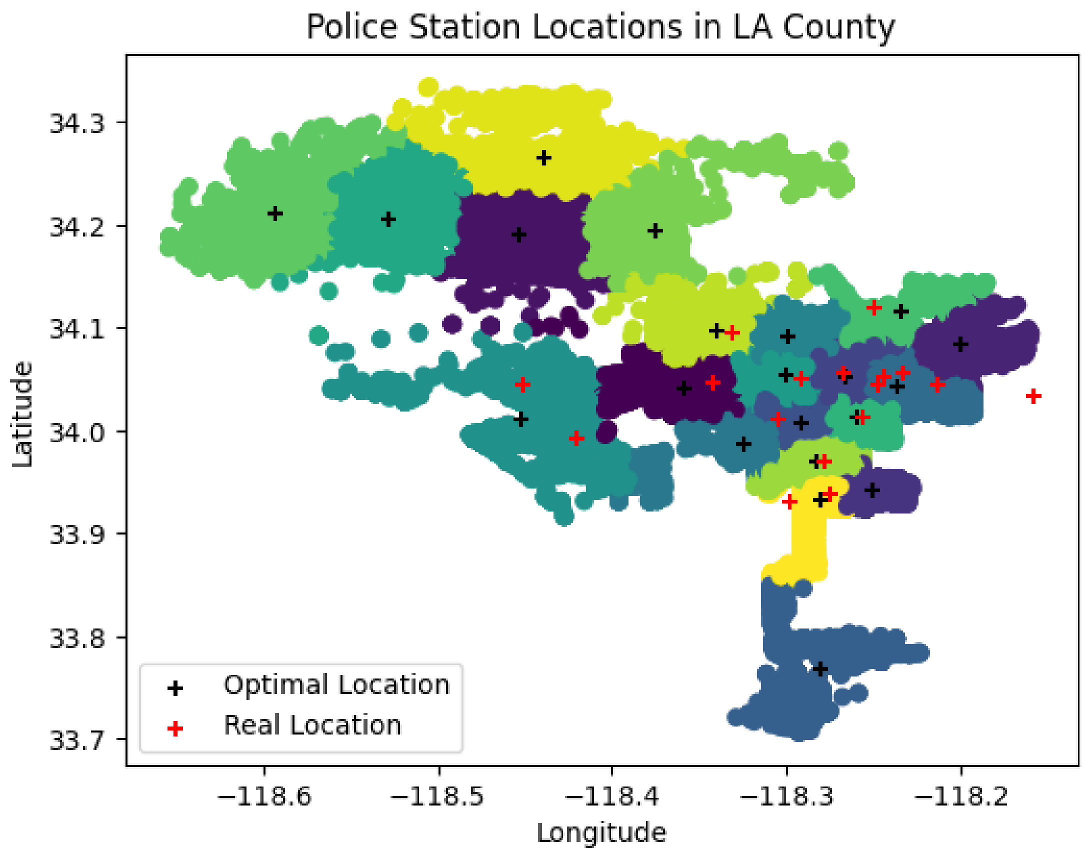
As a part of the plot function, it will not only plot the ideal clusters for police stations but we also read in data to include the actual location of the current 17 police stations in LA county. We also utalized three different norms, the 1-norm, 2-norm and infinity norm to get three different possible locations. This all yielded the following three graphs.



This is the graph that corresponds to the 1-norm.



This is the graph that corresponds to the 2-norm.



This is the graph that corresponds to the infinity norm.

Overall, each of these graphs have slightly different ideal locations for police stations. The black crosses correspond to the optimal location of police stations based on our crime data and the red crosses correspond to the actual location of police stations. We will not analyze the specific differences of the different results from each norm but wanted to provide potential ideal locations based on varying norms. However, there is much that we can learn from these three graphs overall. Each graph has at least three police stations in the north/west part of LA county (the range is from 3 to 5). Also, each graph shows that there would ideally be 1-2 police stations in the furthest south portion of LA county. Interestingly, the red crosses showcase that nearly all of the police stations in LA county are located centerally in LA county. This showcases that LA could look into investing and placing more police stations in the north/west and south parts of its county. Overall, these kmeans maps help us to understand the location of crime and how to better place police stations to minimize the distance from crime to police stations. This has many practical applications as it could decrease response time and increase overall safety.

Now that we have examined ideal locations, we will look into further classifying crime based on attributes of the crime.

5 4. Feature Engineering

When deciding which features to include in our models, we first dropped those that had potential ethical issues, as mentioned above. After that, we removed the columns corresponding to secondary crimes, as we decided that those may too directly relate to the primary crime. Also, we decided it to be an unlikely scenario in which one would be aware of secondary crimes but not the primary. We also decided to drop the status of the case because that is irrelevant to when the crime actually occurred. Finally, we removed any columns with duplicate information (i.e.) linearly dependent columns, such as the crime description and weapon description. With each model, there were additional measures taken to select which features to use, such as built in feature selection and PCA.

6 5. Visualizations and Model Analysis

Random Forest Model

Grid Search of HyperParameters:

Based off the data cleaning, we perform a grid search on the dataframe after dropping crime codes with less than 600 instances. This helps us to be able to better classify crimes that frequently happen instead of trying to classify less relevant crimes. RandomForest is a good classifier when dealing with binary or multi-class classification as it is composed of many decision trees. Our goal was to use different features, such as weapon description, location, and time of day to identify the likely crime being committed which is recordered in the crime code.

The data provided had multiple crimes with only a few instances, so by cleaning it, we've reduced the amount of crime codes to 16. We can now attempt to classify the 16 most frequent crimes in our data set.

We first performed a grid search to look for the optimal hyperparameters of our random forest model on the cleaned data.

```
[ ]: #Define your training and test data
     X = new_df
     le = LabelEncoder()
     y = le.fit_transform(df['Crm Cd'])
     X_train,X_test,y_train, y_test = train_test_split(X,y, test_size = 0.3)

     #define a Parameter Grid
     rf = RandomForestClassifier()
     param_grid = {'n_estimators': [25,50,100], "criterion": ['gini','entropy'],␣
      ↪"max_features": [None,'sqrt','log2'], 'max_depth': [5,10]}

     #Perform a Grid search with 3-fold cross validation
     rf_gs = GridSearchCV(rf,param_grid,cv = 3, n_jobs = -1)
     rf_gs.fit(X_train, y_train)

     #display the best parameters and your score
     print(f'Best Parameters: {rf_gs.best_params_}')
     print(f'Best Score: {rf_gs.best_score_}')
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': 10, 'max_features': None, 'n_estimators': 100}
Best Score: 0.5290793275184281
```

Above were the results of the grid search with the provided hyperparameters and the cleaned data. We then performed post-classification feature selection to get the optimal features for our columns to see if feature importance would allow us to increase the score of our model.By decreaseing the amount of features that the data is trained on we also decrease the complexity and time to train our model.

```
[ ]: #define random forest with optimal parameters
     X = new_df

     #provide labels to each crime code from the cleaned data
     le = LabelEncoder()
     y = le.fit_transform(df['Crm Cd'])
     rf = RandomForestClassifier(n_estimators=100,criterion='gini', max_depth=10,␣
      ↪max_features=None)
     rf.fit(X_train, y_train)

     #get your predictions, accuracy and f1 score
     y_pred = rf.predict(X_test)
     accuracy_s = accuracy_score(y_test, y_pred)
     f1_score(y_test, y_pred, average = 'micro')
```

Here are the accuracy and F__1 scores that the model received after training on the cleaned data.

F1: 0.513062603937557

Accuracy: 0.513062603937557

```
[ ]: #get the Optimal Features based off of feature importance
     sel = SelectFromModel(RandomForestClassifier(criterion = 'gini', max_depth =␣
      ↪10, n_estimators = 100))
     sel.fit(X_train, y_train)

     #select the features to retrain the model on
     X_selected = sel.fit_transform(X_train, y_train)
     X_selected = X_selected.astype(int)
```

Here are the accuracy and F__1 scores after picking out features based off feature importance determined by the SelectFromModel package of Sklearn.

F1: 0.5093074405879513

Accuracy: 0.5093074405879513

A classification report allows us to visualize how each label was classified, and the amount of data that existed for each label. Precision being the amount of true positives out of all the data classified as positive, recall being the true positive rate, f1 being the combination of those, and support being the amount of true positives and false negatives in the cleaned dataset.

```
          precision    recall  f1-score   support


     330       0.86      0.03      0.05       224
```

```
         210       0.51      0.09      0.15      3293
         230       0.00      0.00      0.00       583
         930       0.55      0.84      0.67      5260
         350       0.33      0.00      0.00       531
         624       0.67      0.02      0.04       190
         626       0.00      0.00      0.00       206
         740       0.61      0.64      0.62       227
         860       0.48      0.75      0.58      3957
         220       0.38      0.01      0.02       242
         625       0.38      0.58      0.46      1593
         121       0.47      0.36      0.41       328
         236       0.67      0.63      0.65       256
         761       0.17      0.00      0.00       965
         623       1.00      0.00      0.01       251
         753       0.69      0.85      0.76       535

    accuracy                           0.51     18641
   macro avg       0.48      0.30      0.28     18641
weighted avg       0.48      0.51      0.42     18641
```

Gradient Boosted Model

We decided the next step was to create a boosted model. This could possibly help us increase the accuracy score that we had in the previous model. To do this we ran a new parameter search using a gradient boosted classifier. For this we used the same data cleaning techniques. The following codes showcases how we implemented the classifier and what grid search we ran on it.

```
[ ]: le = LabelEncoder()
     #Get crime data
     y = le.fit_transform(df['Crm Cd'])
     X_train,X_test,y_train, y_test = train_test_split(X,y, test_size = 0.3)

     #initialize classifier and set parameters
     rf = GradientBoostingClassifier()
     param_grid = {'n_estimators': [25,50,100], "loss": ['log_loss','exponential'],␣
      ↪"max_features": [None,'sqrt','log2'], 'max_depth': [5,10],␣
      ↪'min_samples_leaf': [1,4,8]}

     #run the grid search
     rf_gs = GridSearchCV(rf,param_grid,cv = 3, n_jobs = -1)
     rf_gs.fit(X_train, y_train)
     #display the best parameters and your score
     print(f'Best Parameters: {rf_gs.best_params_}')
     print(f'Best Score: {rf_gs.best_score_}')
```

Best Parameters: {'loss': 'log_loss', 'max_depth': 5, 'max_features': None, 'min_samples_leaf': 4, 'n_estimators': 25}

Best Score: 0.3777531474692358

As before we will now run a post-classification feature selection using sklearn to try and improve the score of our model.

```
[ ]: #get the Optimal Features based off of feature importance
     sel = SelectFromModel(GradientBoostingClassifier(loss = 'log_loss', max_depth =␣
      ↪5, n_estimators = 25,min_samples_leaf=4,max_features=None))
     sel.fit(X_train, y_train)

     #select the features to retrain the model on
     X_selected = sel.fit_transform(X_train, y_train)
     X_selected = X_selected.astype(int)
```

From this we get an accuracy and F_1 score of 0.31574318381706246 and 0.31574318381706246 respectively.

Here is our classification report for this model helping us understand more of our statistical specifics for our model. precision recall f1-score support

```
     210       0.23      0.00      0.01      3233
     230       0.00      0.00      0.00       549
     930       0.32      0.93      0.47      5359
     624       0.12      0.00      0.00       482
     626       0.32      0.10      0.15      3926
     740       0.04      0.00      0.00      1650
     220       0.00      0.00      0.00       373
     236       0.00      0.00      0.00       953
     761       0.00      0.00      0.00       530

accuracy                             0.32     17055
macro avg        0.12      0.11      0.07     17055
weighted avg        0.22      0.32      0.18     17055
```

We expected the boosted model to be an improvement to the random forest model, however our results suggest that this is not the case. Our accuracy has consistently been worse for this model. We suspect this may be due to a less thorough grid search. There is likly a better combination of hyperparameters for this model that would greatly improve the scores that we are seeing. One improvement that we could make to help this model perform better is to extend our parameter grid to include more options and run a more indepth search to achieve the best model. Another improvement that could be made is to adjust the features that we are using to train our model. There is likely a more optimal combination of features that would improve the predictive power of our model, thus increasing scores.

Multiclass Logistic Regression Classifiers

We will no look into our last model. We decided to also examine how a multiclass logistic regression classifier would work.

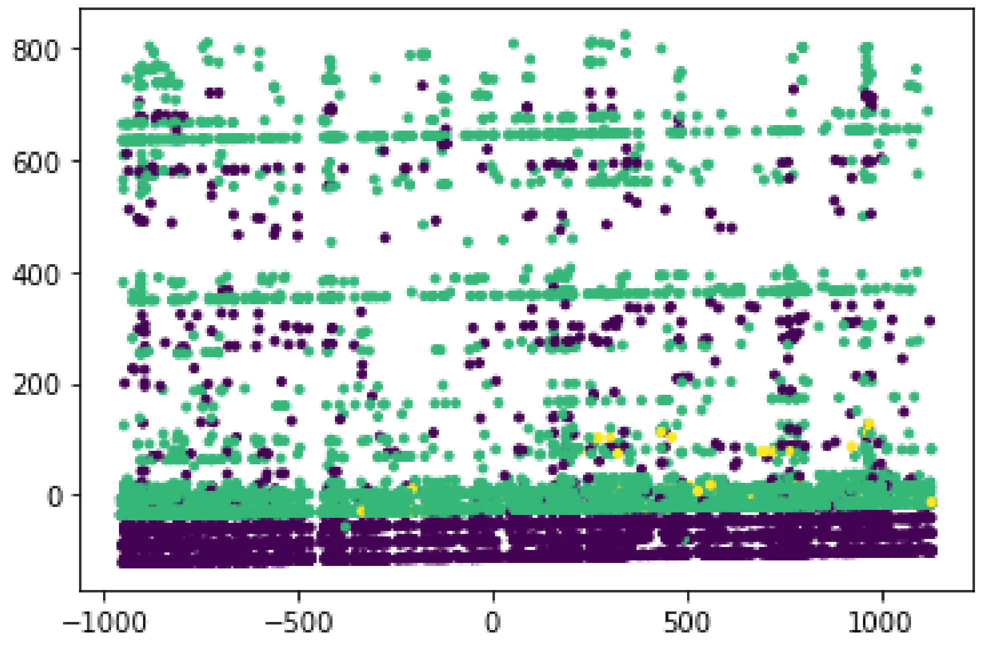
```
[3]: le = LabelEncoder()
     #get crime data X and labels y
     y = le.fit_transform(crime_data['Crm Cd'])
```

```
#split test data and initialize model
X_train,X_test,y_train, y_test = train_test_split(X,y, test_size = 0.3,␣
 ↪random_state=3)
logreg = LogisticRegression(multi_class='multinomial', solver='lbfgs', C=1,␣
 ↪max_iter=500)
#train and run test data
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
#get accuracy score
accuracy = logreg.score(X_test, y_test)
print(f'Accuracy of the Logistic Regression Model: {accuracy}')
```

```
Accuracy of the Logistic Regression Model:  0.4540603928466725
```

Multiclass logistic regression is a valuable tool for predicting crime types, particularly in scenarios where the dataset involves nonbinary labeled data. Using this model allows for the classification of multiple crime categories based on associated features. In running logistic regression, we use identical datasets as the ones previously cleaned. After a grid search to find the optimal parameters, we settled on a multinomial class with regularization strength C of 1, while doing lbfgs as the solving method. When applied to crime data, our multiclass logistic regression model yielded an accuracy score of 0.4541, signifying its moderate predictive capability in discerning between different crime types. Considering there are 16 types of crime within the dataset, this is a noteworthy score. Despite the modest accuracy, a visualization of the graph shows that the majority of crimes are classified under the two highest crime categories in green and purple (corresponding to assualt and battery, respectively). Each dot represents a crime reported and the color is the type of crime predicted by the model. The axes are arbitrary components to assist in two dimensional visualization of the classifier.

Visualization of Multiclass Classifier



6.1 6. Results and Conclusion

When examining the ideal location of resources and police stations based on crime in LA using kmeans, we found that the given locations may not be best suited for the crime data we were working with. We understand that there is much more nuance to choosing where to place these resources than we included, and recognize that more work is required to validate our results. Though none of our models were particularly accurate in predicting types of crime, they weren't entirely unsuccessful. Our random forest classifier works about 50% of the time, which is not a dismissible score, given that we included 16 different classifications for crime. We found that simply using location is not enough to determine the type of crime, and including information about the victim and weapon used greatly improved the accuracy of our models. We believe in the likely possibility that classification of crime is much more complicated than we expected, and this data simply may not include the necessary information to do so. There is room for more complicated modeling to address the provided issues, as well as consideration of tracking other features related to each crime.

REFERENCES

Source for crime rates in LA over time

https://lasd.org/transparency/statistics/

Sources about need for victim resources. TO DO: do we want to include these?

https://time.com/5886815/crime-survivors-funding/

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9837801/