San Franciso Crime Classification

Crime Fighters
Gopalakrishnan V (IMT2016001)
Vibhav Agarwal (IMT2016003)
Varada Desikan P S (IMT2016115)

Abstract

Given the address, locality, district, date and time of crimes that occurred in San Francisco bay area, we predict the category of a crime. The inferences drawn from the visualizations and the data analysis have played an important role while building the results.

I. Introduction

Crime is a horrific act. What's more horrifying is to note that people willingly commit crimes everyday. Would it be possible to categorize crimes? Will it be helpful to categorize crime just based on the environment? Assuming that we have been given the details of the crime that has happened, can we predict what it can be without knowing the description? How accurate would such a prediction be?

II. Problem Statement

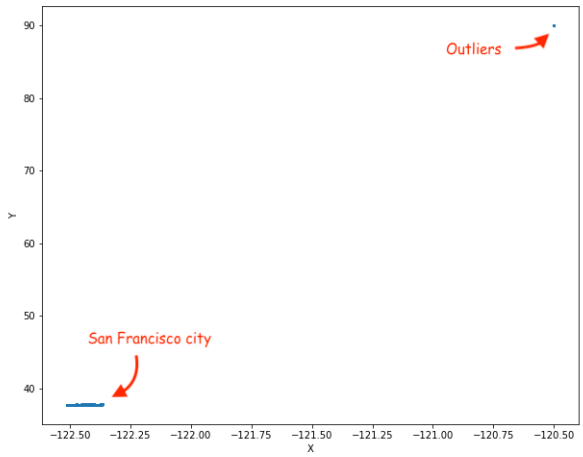
Predict the category of a crime given its address, district, date and time of the crime depending on past occurrences of crimes around the same area in the span of nearly 12 years.

III. Dataset

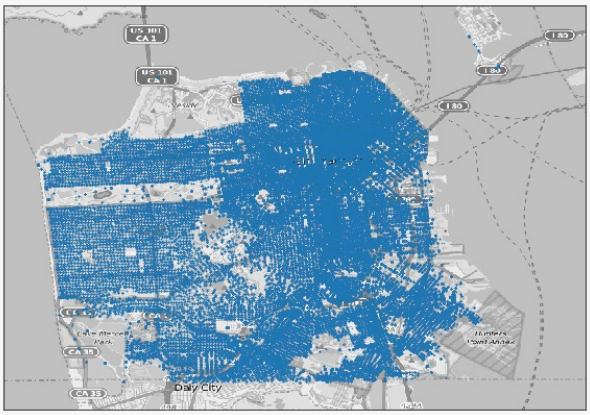
The dataset used is the public dataset from the *Kaggle problem's data* section. The dataset contains the information of crimes occurred in San Francisco Bay area like category, time, date, day, PdDistrict, Address, X and Y coordinates.
The dataset consists of around 868,000 crimes with 9 columns consisting of 36 different categories. The dates mentioned in the dataset ranges from 01-01-2003 to 13-05-2013

IV. Visualization

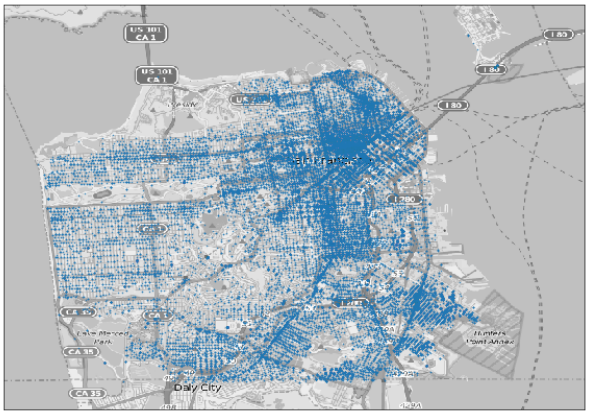
1) On analyzing the X, Y distribution of the crimes across the city, we observed that there were some outliers as seen in this figure.



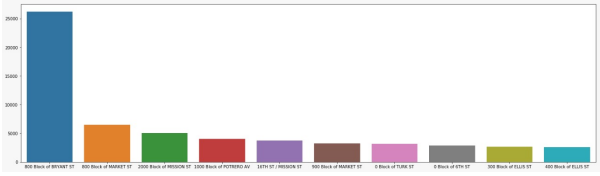
After preprocessing these outliers, we plotted all the data points on the city of San Francisco and got the following distribution.



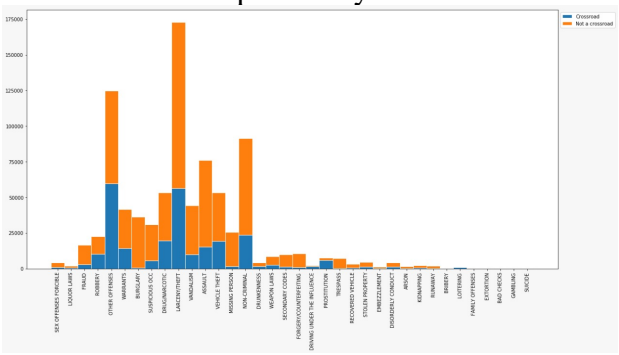
2) We took one particular category "Assault". As it is visibly noticeable, the North Eastern part of San Francisco is the most prone area. Similar visualization for other categories can also be computed.



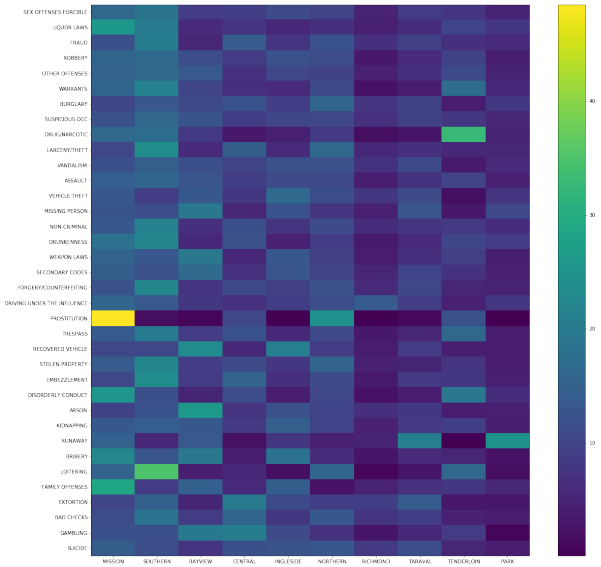
3) We tried to figure out what were the most commonly affected streets and how frequently crimes happened there. ”Bryant Street” was found to be the most prone one



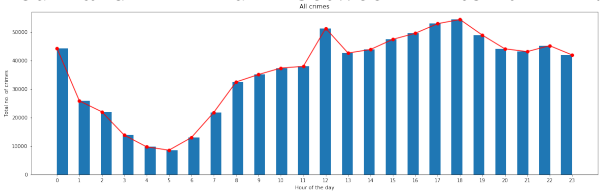
4) Moreover we wanted to know how much the crossroads proved to be a tool for the crimes and specifically for which ones.



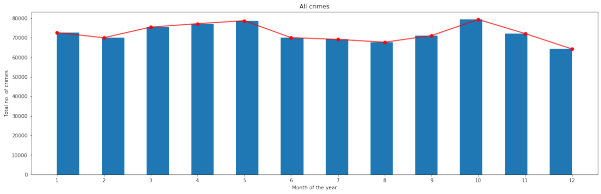
5) To generalize these, we then decided to plot a heat map against the district for all categories thus providing an overall image and a distribution.



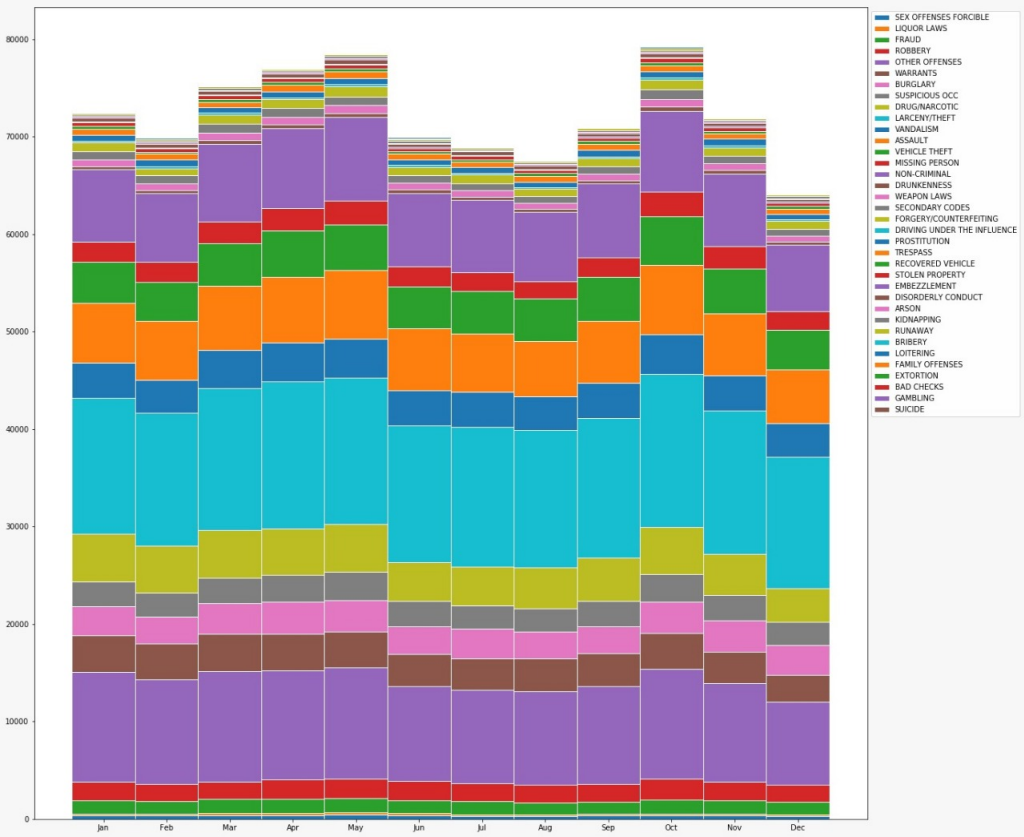
6) The histogram explains the frequency of the crimes happening during each hour of the day. We can observe that the frequency of the crimes is maximum during the rush hour and minimum between 2 to 7 AM.



7) The histogram explains the frequency of the crimes happening during each month of the year. We can observe that there is no prominent high frequency during any month. Particularly, we can observe that October has the highest crime rate.



8) The following histogram is an improvement over the prior where the distribution of each crime is also shown. We can observe that ”Larceny/Theft”, ”Assault” and ”Other offences” are the major crimes.



V. Pre-processing

The dataset provided did not have any NULL values. The date-time information provided through the *Dates* column had a seconds attribute. But on analysis we came to the conclusion that this will have minimum impact on the final result as it always had the value *zero*. Hence the seconds information was not extracted from the *Dates* column.

While analysing the crimes based on the provided X, Y coordinates, it was noted that a few points where outside the city of San Francisco. During this stage such outliers where removed.

VI. Feature Engineering

The given dataset had the columns [*Dates*, *Category*, *Descript*, *DayOfWeek*, *PdDistrict*, *Resolution*, *Address*, *X*, *Y*, *Id*] with the target variable being *Category*. As per the rules of the competition, the attributes *Descript* and *Resolution* were not used.

The *Dates* column was parsed and the *Hour*, *Minutes*, *Year*, *Month*, *Day* features were extracted. The feature *Day* was a string attribute and to handle it we encoded it to a corresponding number. The number 31 in the column *Day* was noted to be less and this was attributed to the lesser number of months with 31 days. The distribution of minutes was skewed; the number of entries at 0 and 30 were abnormally high, hence to mend this we decided to the move the intermediate values to their corresponding quartiles. We bucketed the minutes into 4 sections of 15 minutes interval.

Analysis through the data showed that the number of crimes happen to increase during the winter months. Hence to reap this information out from the data, we decided to build the feature *seasons*. The information about the seasonal patterns of San Francisco was obtained from https://www.studentflights.com.au/destinations/san-francisco/weather.

The type of data in the attribute *PdDistrict* was a string. Hence to handle the same without increasing

the dimension of the problem, we encoded the string using a Label Encoder.

A large percentage of the population tend to roam the city during weekends and hence it is safe to believe that a lot of the crimes occur on weekends. To extract this information from the data we built a *is_weekend* feature that would help us in understanding the crime scenarios on weekends.

Crimes tend happen in the dark i.e. night time in the after hours. We engineered a new feature called *night_time* to test our assumption of crimes happening in the after hours of 10 P.M. to 6 A.M. But during the course of analysis through visualization, we noted that the significance of this feature could be minimal. We came to this conclusion as, the distribution was fairly spread out and hence didn't seem to align along our belief.

We had an inherent feeling that the *Address* could provide a deep insight into our problem statement. We started off by computing the addresses which signal a cross-road. We computed a Boolean variable corresponding to this by checking if the string contained a "/". The number of unique addresses were large in number, and an analysis over it was difficult. Hence we considered the subset of addresses where more than 100 crimes have occurred. We kept track of these *important* addresses in the attribute *Address Clean*.

From the *Address Clean* attribute, we evaluated the number of cross roads and as mentioned such locations were identified by "/". On analysis through such data points, it was identified that a lot of roads mentioned across the "/" were flipped (ie. it did not have an inherent ordering). To resolve this issue, we parsed the list of all important cross roads and introduced an ordering in them. After introducing an ordering, the list of addresses were label encoded.

Prior to all the above feature extraction from the *Address* column, we tried just extracting the last two words of the address. Ex: from the address '1800 Block of Martin ST', we tried extracting out the 'Martin ST'. These text address attributes were later label encoded. We later realized that this wasn't a clean feature as it wouldn't work out for the addresses containing crossroads i.e., having '/'. During our experiments with this feature, the built models were not up to the mark.

Another thing that we observed from all these feature engineered Address attributes was that there are a lot of 2 character address attributes like 'ST' for Street, 'AV' for Avenue, 'BL' for Boulevard etc. We thought that these 2 character attributes might be useful for better prediction, therefore we extracted all of these from the address. To encode such information we decided to use One Hot Encoding. As crossroads had two 2 letter annotations, the one hot representation of the same was helpful. It allowed us to put 1s at both those columns. The One Hot encoding allowed us to evaluate the correlation of these features against the target variable. The down side of this representation was the increased dimensionality of the problem space.

We did test this feature set but there was only a marginal increase in the score. From the analysis of the correlation matrix it was noted that only annotations like 'ST', 'Block', 'AV' had high correlation compared to others. This forced us to build new Boolean features like is_avenue, is_street, is_block. Using these 3 features against all the previously generated 2 letter attributes was better as it allowed us to reduce the dimensionality of the problem space and at the same time gave a good picture.

Holidays are also the time when a certain section of the crime increases a lot. Therefore we thought that incorporating this feature would help us a lot. We used a python package for this called **Holidays** which returned a list of US holidays. By taking help of this library, we cross-checked if the mentioned date was a US holiday or not.

Through the visualizations of the crimes across the city, it was noticed that most of the crimes happened in the north-east corner of the city. To incorporate this notion explicitly into our model, we introduced the features radial distance and

the product *XY*. Before computing these features, we normalized the *X* and *Y* values by fitting a *Standard Scalar*. The radial distance was evaluated as $r = \sqrt{X^2 + Y^2}$.

The polar coordinates, radial distance r and angle θ, were used to extract the spatial information of the crimes across the city. It is reasonable to consider that a given crime could have happened in the neighbourhood of the given *X* and *Y*. Hence to bring in the idea of neighbourhood, we took help of the rotation matrix. We considered three variants of rotated Cartesian coordinates (rotated by 30, 45, 60 degree each).

In linear algebra, a rotation matrix is a matrix that is used to perform a rotation in Euclidean space.

$$R = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix}$$

The above matrix rotated the points in the xy-plane counterclockwise through an angle θ about the origin of the Cartesian coordinate system.

VII. Evaluation Criteria

Evaluation criteria for this problem is the **Multi-class log loss**.

$$\text{logloss} = -\frac{1}{\text{N}} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} log(p_{ij})$$

where N is the number of data points in the test set and M is the number of class labels. y_{ij} is 1 if observation i is in class j and 0 otherwise. p_{ij} is the predicted probability that observation i belongs to class j.

VIII. Feature sets used

Different feature sets generated and used in model evaluation:

1) **Standard features (SF):** X, Y, Hour, Minutes, Year, Month, Day, DayOfWeekNum, PdDistrictNum, Address_Cross_Road, Address_clean_encode

2) **Feature set 1 (FS1):** is_weekend, is_night_time, is_holiday

3) **Feature set 2 (FS2):** All 2 letter address attributes that we generated in the feature engineering step like ST, AV, BL etc.

4) **Feature set 3 (FS3):** is_avenue, is_street, is_block

5) **Feature set 4 (FS4):** All features related to the X, Y coordinates like X_reduced, Y_reduced, rot_45_X, rot_45_Y, rot_30_X, rot_30_Y, rot_60_X, rot_60_Y, radial_r, XY_reduced.

6) **Feature set 5 (FS5):** bucketed minutes, seasons

IX. Model Experiments

To iterate over various possible combinations of hyper-parameters, GridSearch was deployed (provided by *sklearn*).

1) **Logistic Regression**
   The algorithm used in the optimization problem has been set to 'sag' as it is fast for large datasets. The 'sag' algorithm is capable of handling nulticlass problems with multinomial loss.

2) **Naive Bayes**
   Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variables. Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods.

3) **Decision Tree Classifier**
   Decision trees are a supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
   Advantage
   - Requires less data preparation. However this module does not support missing values.
   - Cost of using this tree (ie. predicting data) is logarithmic in the number of data points

TABLE I
MODELS EVALUATED AGAINST DIFFERENT FEATURE SETS

Model	**Model Parameters**	**Feature Set**	**Validation Score**
Logistic Regression	solver: sag, max_iter: 100	SF	2.582
Gaussian Naive Bayes		SF	2.611
Decision Tree	depth: 6	SF	2.479
Decision Tree	depth: 7	SF	2.478
Decision Tree	depth: 8	SF	2.486
Random Forest Classifier	depth: 6	SF	2.450
XGBoost	learning_rate: 0.1, max_depth: 8	SF	2.252
Random Forest Classifier	depth: 7	SF	2.433
Random Forest Classifier	depth: 8	SF	2.413
Random Forest Classifier	depth: 12	SF	2.325
Random Forest Classifier	depth: 16	SF	2.290
XGBoost	**learning_rate: 0.2, max_depth: 8**	**SF**	**2.247**
XGBoost	learning_rate: 0.2, max_depth: 9, min_child_weight: 1.5	SF	2.238
Random Forest Classifier	depth: 11, n_estimators: 300	SF + FS2	2.390
Random Forest Classifier	depth: 13, n_estimators: 300	SF + FS2	2.350
Random Forest Classifier	depth: 18	SF + FS2	2.304
Random Forest Classifier	depth: 19	SF + FS2	2.307
Random Forest Classifier	depth: 17	SF + FS2	2.308
Random Forest Classifier	depth: 10	SF + FS1	2.371
XGBoost	learning_rate: 0.2, max_depth: 8	SF + FS5	2.251
Random Forest Classifier	depth: 13	SF + FS1	2.319
Random Forest Classifier	depth: 16	SF + FS1	2.294
XGBoost	depth: 6, n_estimators: 40, learning_rate: 0.2	SF + FS2	2.289
Random Forest Classifier	depth: 18	SF + FS1 + FS2	2.304
Random Forest Classifier	depth: 17	SF + FS1 + FS2	2.303
LightGBM	learning_rate: 0.05, is_unbalance: true	SF + FS1 + FS2	2.326
Logistic Regression	solver: sag, max_iter: 70	SF + FS1 + FS2	2.592
Gaussian Naive Bayes		SF + FS1 + FS2	2.589
PCA with XGBoost	n_components for PCA: 17, max_depth: 8, n_estimators: 100	SF + FS1 + FS2	2.509
XGBoost	**learning_rate: 0.2, max_depth: 8**	**SF + FS1 + FS3**	**2.246**
XGBoost	learning_rate: 0.2, max_depth: 8	SF + FS1 + FS3 + FS4	2.243

used to train the tree
- Able to handle both numerical and categorical data
- Able to handle multi-output problems

Disadvantage

- Decision tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms like setting the max-depth of the tree are required to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.

4) **Random Forest Classifier**
A random forest classifier is a meta estimator that fits a number of decision tree classifiers on a various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

5) **XGBoost**
Implements machine learning algorithms under the Gradient Boosting framework. It provides a parallel tree boosting algorithm (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.
- Default booster *gbtree* uses tree based models
  - Learning Rate : Step size shrinkage used in update prevents overfitting.
  - Max Depth : Increasing this value will make the model more complex and more likely to overfit.
- Learning task parameters
  - Objective - mutli:softmax : Set XGBoost to do multiclass classification. The result contains predicted probability of each of data point belonging to each class.

6) **LightGBM**
LightGBM is gradient boosting framework that uses tree based learning algorithms.
Advantage
- Faster training speed and higher efficiency.
- Better accuracy.

Parameters
- Application : Specifies the application of the model. *multiclass* : For multiclass classification problem.
- Boosting : Defines the type of algorithm to be run. Default = *gbdt*
- Learning Rate : Determines the impact of each tree on the final outcome. GBM works by starting with an initial estimate which is updated using the output of each tree. This parameter controls the magnitude of this change in the estimates.

7) **Principal Component Analysis (PCA) with Random Forest (RF) and XGBoost**
Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.

8) **Stacking Models**
Stacking is an ensemble learning technique that uses predictions from multiple models to build a new model.

X. Directory Structure

- pickle/
  - XGBoost Model 1
  - XGBoost Model 2
- dataset/
  - train.csv
  - test.csv
  - fe_train.csv [*Generated from the associated source code*]
  - fe_test.csv [*Generated from the associated source code*]
- feature_engineering.py
- model.py
- visualization.py
- requirements.txt
- sf_map_copyright_openstreetmap_contributors.txt [*Map file for the visualization*]

Install the required packages by running *pip3 install -r requirements.txt*. The feature_engineering.py has to be run as *python3 feature_engineering.py*. This

will generate the prepared train and test dataset (referred as fe_train.csv, fe_test.csv respectively). Now run the file model.py as *python3 model.py* to build the respective model. To build the associated visualizations, please run the visualization.py as *python3 visualization.py*.
The required directory structure is maintained at **Drive Link**. The pickle models are maintained the 'pickle' directory **Pickle directory** and the associated dataset (along with the cleaned data) has been maintained in the 'dataset' folder **Dataset directory**.

XI. Conclusion

The model successfully predicts the category of a crime occurring in San-Francisco with an overall validation score of 2.246. We had many hypotheses before we began our analysis. The visualization gave an insight and the predictions gave us valediction of each result. We were proven to be wrong at a few cases. The models that we build proved to be a really good learning factor for us to see how the score changes with respect to each change in the way we predict.

XII. Future Work

Ensemble modelling can exponentially boost the performance of your existing models. Another thing that seems worthwhile is to split the examples into subsets and build multiple models on them and then average them out. This way we can tune the hyperparameters on a smaller training set and even if we overfit a bit, it will be relatively okay because of the bagging effect.

XIII. References

- Logistic Regression sklearn documentation
- Naive Bayes sklearn documentation
- Decision Tree Classifier sklearn documentation
- Random Forest Classifier sklearn documentation
- XGBoost documentation
- LightGBM tutorial
- XGBoost hyper parameter tuning
- ScatterPlot