

Machine Learning Engineer Nanodegree

Capstone Project

Levent Peker
December 1st, 2016

I. Definition

San Francisco Crime Classification: Predict the category of crimes that occurred in the city by the bay

Project Overview

From 1934 to 1963, San Francisco was infamous for housing some of the world's most notorious criminals on the inescapable island of Alcatraz.

Today, the city is known more for its tech scene than its criminal past. But, with rising wealth inequality, housing shortages, and a proliferation of expensive digital toys riding BART to work, there is no scarcity of crime in the city by the bay.

Kaggle (<https://www.kaggle.com/>) is hosting this competition for the machine learning community to use for fun and practice. This dataset is brought to you by SF OpenData (<https://data.sfgov.org/>), the central clearinghouse for data published by the City and County of San Francisco.

Solution of the problem will lead us to know which areas of the city at when what category of crimes happens. Depends on the crime category, with help of police departments we can solve issues and protect people in neighborhoods before the crimes will happen. With time and category of the crime information; results of the study will be helpful to organize police department teams on field, allocate the police teams in specific areas. We will have better idea and understanding that in what time, what kind of crimes are happening in what kind of neighborhoods. We can get better cautions or we can focus on some areas with more security to protect people.

Problem Statement

Given time and location, we have to predict the category of crime that occurred.

I will also explore the dataset visually. What can we learn about the city through visualizations like this Top Crimes Map?

There are 39 types of crimes in total in the dataset. LARCENY/THEFT, OTHER OFFENSES, NON-CRIMINAL, ASSAULT and DRUG/NARCOTIC are the top 5 category of the crimes. For the output of the problem, I will give probabilities for each category of crime. For example: As a prediction, after machine learning applied to the problem we may say in a given time and location assault will happen 0.9 and drug/narcotic will happen 0.1 percentage.

In general, first I will investigate features in the dataset. If any missing points, outliers to be solved or features that have to be converted to other types of data I will do it in preprocessing stage. Then I will split data and create validation set for the problem. I will choose classification models which fit for the problem and pick the best one according to performance. And finally I will apply chosen model to test set and get the final score.

For model selection; I will plan to choose Naïve Bayes, Logistic Regression, Random Forest and XGboost. Naïve Bayes is classification method based on Bayes Theorem and I think will be helpful for this dataset. Logistic Regression is also another classification method which makes predictions on the probability of occurrence of an event by fitting data to a logit function. Random Forest is an ensemble model which creates multiple decision trees and picks the best one depends on voting. And finally I will apply XGBoost which is the best winning algorithm in Kaggle competitions based on gradient boosted decision trees designed for speed and performance.

Metrics

Submissions are evaluated using the multi-class logarithmic loss. Each incident has been labeled with one true class. For each incident, you must submit a set of predicted probabilities (one for every class). The formula is then,

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

where N is the number of cases in the test set, M is the number of class labels, \log is the natural logarithm, y_{ij} is 1 if observation i is in class j and 0 otherwise, and p_{ij} is the predicted probability that observation i belongs to class j .

The submitted probabilities for a given incident are not required to sum to one because they are rescaled prior to being scored (each row is divided by the row sum). In order to avoid the

Format of the output file looks like this:

II. Analysis

I use dataset from Kaggle, which is from SF OpenData. This dataset contains incidents derived from SFPD Crime Incident Reporting system. The data ranges from 1/1/2003 to 5/13/2015.

- **Dates** - timestamp of the crime incident
- **Category** - category of the crime incident (only in train.csv). This is the target variable you are going to predict.
- **Descript** - detailed description of the crime incident (only in train.csv)
- **DayOfWeek** - the day of the week
- **PdDistrict** - name of the Police Department District
- **Resolution** - how the crime incident was resolved (only in train.csv)
- **Address** - the approximate street address of the crime incident
- **X** - Longitude
- **Y** - Latitude

Figure 1: Train and test dataset

There are 39 types of crimes in total in the dataset. In training set we have total 878049, in test set we have total 884262 incidents. In training set we have total 9 features (1 date, 2 numeric and 6 categorical) and in test set we have total 7 features (1 date, 2 numeric and 3 categorical).

Dataset link: <https://www.kaggle.com/c/sf-crime/data>

There is no missing values in both train and test datasets. But when we analyze further steps we will see there are some outliers in X, Y coordinates which belong to outside of SF Bay Area geographically. I will do necessary modifications on these features before we step into model selection part.

Exploratory Visualization

Let's analyze features in the dataset. We will predict the crime category for this project, so the most important feature is **Category**. From training set, first let's see the distribution of category of crime

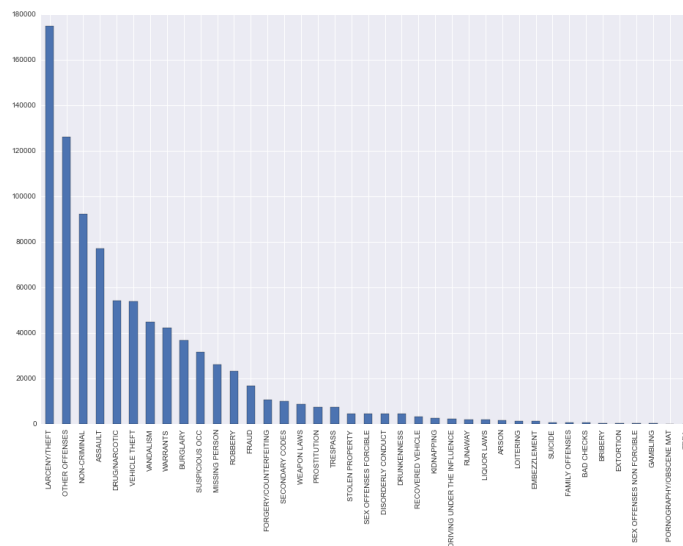


Figure 2: Category of Crimes

There are total 39 category of crimes. From the distribution, we can see most of the crimes are belong to LARCENY/THEFT category. Following of that one OTHER OFFENSES, NON-CRIMINAL, ASSAULT, DRUG/NARCOTIC are the other top ones. Also from the figure, we can say that more than 50% of the crime categories are listed in top 10 category. Most of the categories has very few number of incidents.

Let's check how the crimes are distributed in the day of the week: Friday is the day when most of the crimes occurring. Wednesday and Saturday are the top crime days following Friday. From

the figure, we can say that there is not much difference between days in the week. Beginning of the week like Monday and end of the week Sunday has the lowest number of incidents. Most of the crimes are happening in the middle of the week or the weekend.

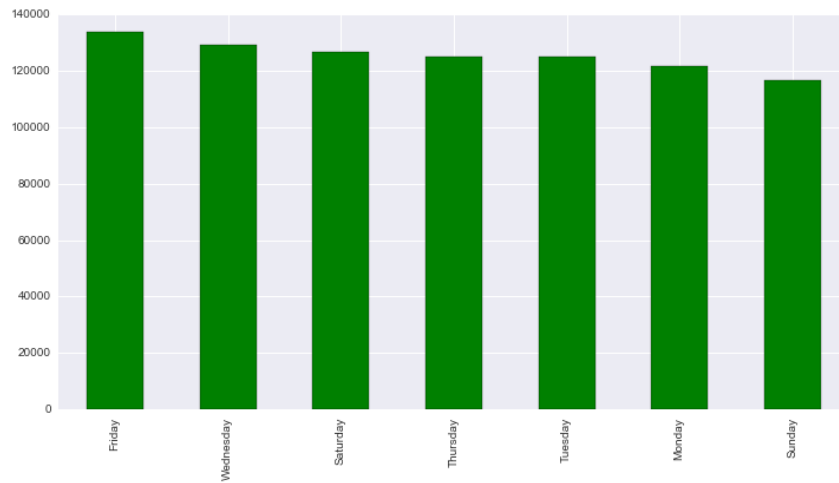


Figure 3: Crimes - Day of the Week

Another useful plot is distribution of Police Department Districts. As we can see; Southern Police Department is the top one where dealing with most of the crimes. Mission and Northern is top ones following that. Richmond has the lowest number of crimes.

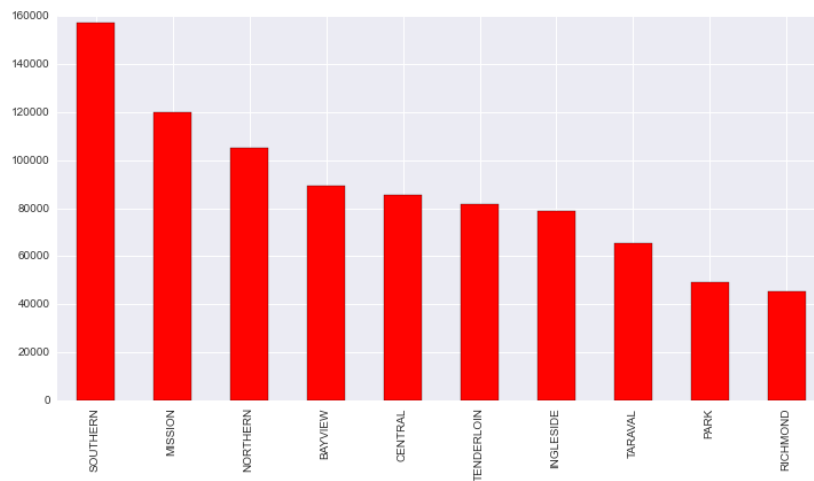


Figure 4: Crimes - PdDistrict

We know top category of crimes and we know which Police Department Districts has high number of crimes. What about the distribution of top 5 crimes for each department? Let's plot Category vs. PdDistrict plot to see the answer:

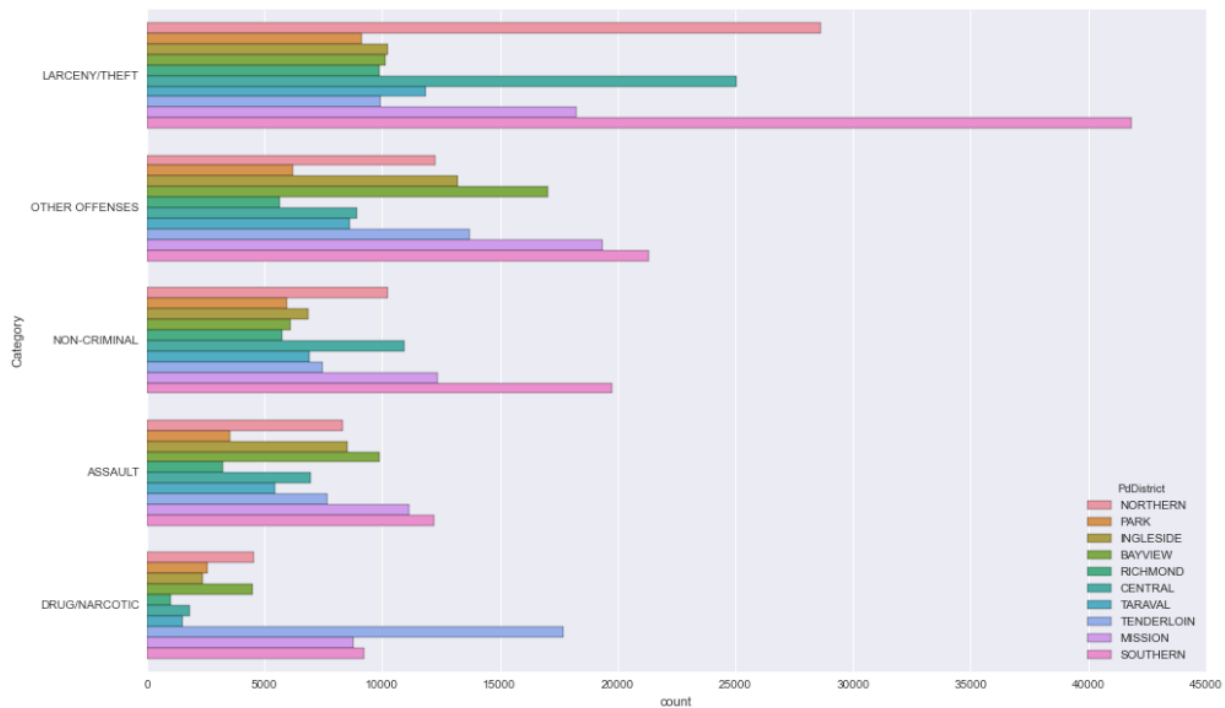


Figure 5: Crimes - Category and PdDistrict

As we can see Top 3 Category of crimes are happening in Southern Police Department District. In LARCENY/THEFT category, surprisingly NORTHERN district counts are also pretty high.

Next step, let's dive into Dates column. To analyze better, I split the Dates as – Year, Month, Days, Hours and Minutes. Since we know exact date information for each crime sample, best question came up to my mind is: In the day, when crimes are happening the most? If we know that answer, we can allocate police sources or we can get more people to work in that specific time frame. And how is top category of crimes are distributed in that? I pick Hour data and let's see how top 5 crimes are distributed in the day:

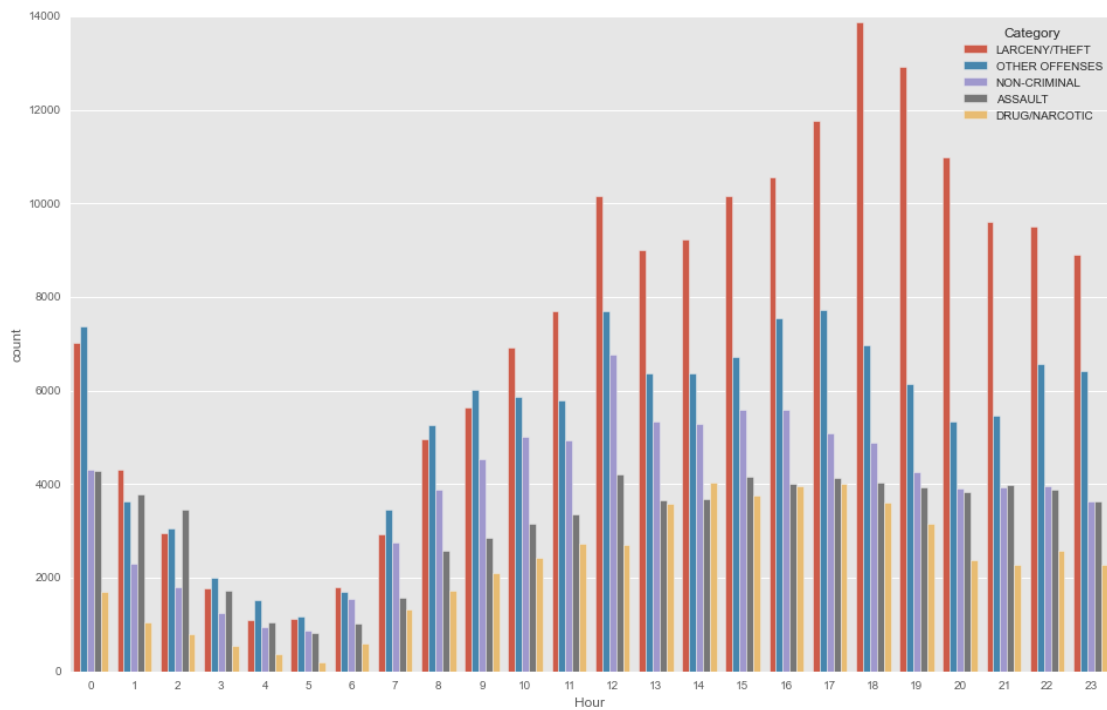
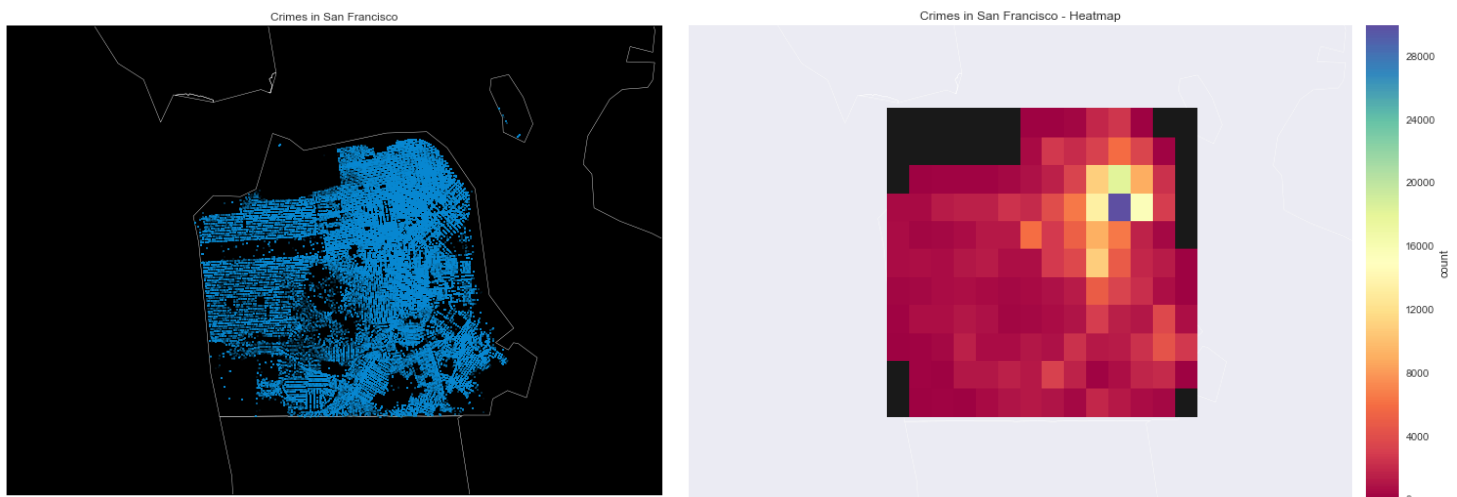


Figure 6: Crimes - Hourly

As we can see, most crimes are happening in 18 o'clock window. And from the graph, also we can see most of the crimes are happening in late hours in the day rather than morning hours.

In the dataset we have also X, Y coordinates. To see where these crimes happening on the map, I use matplotlib Basemap package. After I limit the area with San Francisco county, then I put every point on map and create heatmap depends on number of crimes. Below is the figure how it looks like:

According to heatmap, most of the crimes are in north-west side on San Francisco county. That will give us better understanding in the future where to allocate police resources before the crimes will happen.



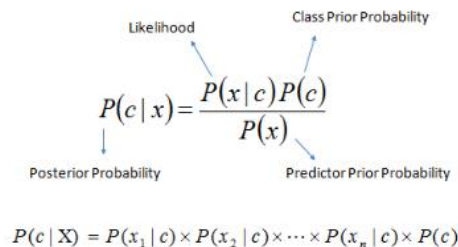
Algorithms and Techniques

Our aim in this project is predict category of the crimes. That means we have classification problem. I use most common classification algorithms for the problem and will pick the one which performs the best.

- Naïve Bayes
- Logistic Regression
- Random Forest
- XGB

Naïve Bayes

Naïve Bayes is a classification technique based on Bayes Theorem. Bayes Theorem assumes features are independent to each other. It describes posterior probability of an event depends on prior probability of class, likelihood and prior probability of predictor.



The diagram shows the formula for Bayes' Theorem: $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$. Arrows point from labels to the terms in the formula: 'Likelihood' points to $P(x|c)$, 'Class Prior Probability' points to $P(c)$, 'Posterior Probability' points to $P(c|x)$, and 'Predictor Prior Probability' points to $P(x)$.

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

- $P(c|x)$ is the posterior probability of class (target) given predictor (attribute).
- $P(c)$ is the prior probability of class.
- $P(x|c)$ is the likelihood which is the probability of predictor given class.
- $P(x)$ is the prior probability of predictor.

I chose Naïve Bayes because it is simple to build and very effective in large datasets like we have. Naïve Bayes is used mostly in text classification and problems with multi classes. It assumes features in the dataset are independent to each other and in our case PdDistrict and X,Y seems like dependent but we will see how it performs in our classification problem.

Logistic Regression

Even it has Regression in its name, Logistic Regression is for classification problems. It estimates discrete values based on dataset which has independent features. It predicts probability of occurrence of an event with using logit_function. It gives output as a probability so results are between 0 and 1. It is useful for categorical variables and for multi classification problems.

Random Forest

Random Forest is an ensemble method of decision trees. In Random Forest algorithm, we are building multiple number of decision trees (that's why its name is 'Forest'). Based on features, each tree gives a classification and then we vote for the all trees. It chooses the classification which has the most votes. They also call Random Forest as 'black-box' model because we give inputs to the model and it gives us the predictions. We don't know what is going on, which calculations are made in the background. I chose Random Forest for this problem because it always gives high performance.

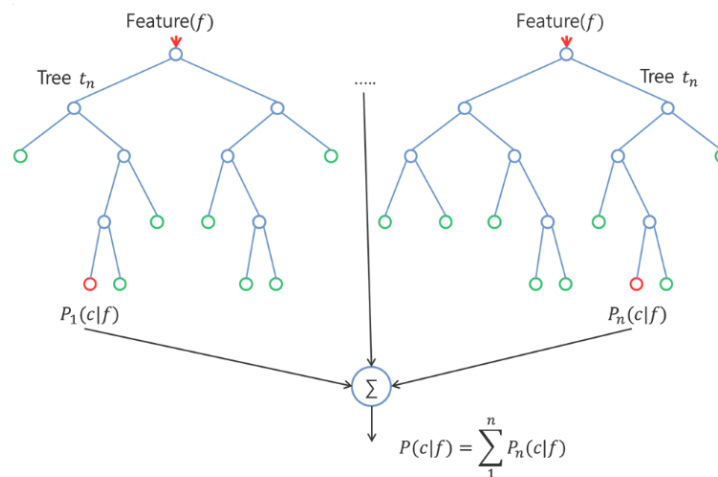


Figure 7: Random Forest method

Extreme Gradient Boosting (XGBoost)

Another algorithm I will try for this specific problem is XGBoost. XGBoost is short term for 'Extreme Gradient Boosting'. XGBoost is very famous in Kaggle competitions known as best-performed-model and winning solution in most of the competitions. It is a library used for optimized for boosted tree algorithms. Main aim of boosting is push the extreme of the computation limits to create scalable, portable and accurate for large scale tree boosting. It is little harder to tune parameters of the model than Random Forests but since results are more accurate it is worth to try.

For this project, I didn't choose SVM and kNN algorithms. SVM is easy to build but handicap is in large datasets that more than 10000 points are performing really bad. As we know, kNN depends on majority of votes from its neighbors. The reason I didn't use kNN is; in this example we don't have normal distributed data in category of crimes. As we can see, our data is skewed; most of the crimes belong to top 6-7 categories from 39 categories. Second thing is we have

too many categorical features in the dataset. So according to that; I don't think we will get good results on lazy algorithm like kNN.

After I select the best model depends on log-loss score I will use GridSearchCV algorithm to find best parameters to optimize the model. Below are the list of classification models and some parameters to optimize and good range of values for the parameters:

Model	Parameters to optimize	Good range of values
Logistic Regression	<ul style="list-style-type: none"> • Penalty • C 	<ul style="list-style-type: none"> • L1 or l2 • 0.001, 0.01.....10...100
Naive Bayes (all variations)	NONE	NONE
Lasso	<ul style="list-style-type: none"> • Alpha • Normalize 	<ul style="list-style-type: none"> • 0.1, 1.0, 10 • True/False
Random Forest	<ul style="list-style-type: none"> • N_estimators • Max_depth • Min_samples_split • Min_samples_leaf • Max features 	<ul style="list-style-type: none"> • 120, 300, 500, 800, 1200 • 5, 8, 15, 25, 30, None • 1, 2, 5, 10, 15, 100 • 1, 2, 5, 10 • Log2, sqrt, None
Xgboost	<ul style="list-style-type: none"> • Eta • Gamma • Max_depth • Min_child_weight • Subsample • Colsample_bytree • Lambda • alpha 	<ul style="list-style-type: none"> • 0.01,0.015, 0.025, 0.05, 0.1 • 0.05-0.1,0.3,0.5,0.7,0.9,1.0 • 3, 5, 7, 9, 12, 15, 17, 25 • 1, 3, 5, 7 • 0.6, 0.7, 0.8, 0.9, 1.0 • 0.6, 0.7, 0.8, 0.9, 1.0 • 0.01-0.1, 1.0 , RS* • 0, 0.1, 0.5, 1.0 RS*

Figure 8: Parameters to optimize

Benchmark

We have total 39 choices of category of crimes, with only one of them correct. A perfect model would assign 1.00 to correct choice and 0.00 to all other choices. This will give you Log Loss = $\ln(1)^{-1}$, which is 0.00. When we change $\ln(1)^{-1}$ to let say $\ln(0.3)^{-1}$. now your Log Loss becomes 1.203. According to people who worked on the problem before, benchmark model log loss of the problem is around 35. Top scorer in Kaggle public leaderboard has 1.95. Top 1000 is started from 2.56 log loss score. My aim is to beat this value with my results.

III. Methodology

Data Preprocessing

Before we begin analysis, let's dive into our dataset. First, let's see how many features, how many total rows/columns we have. Is there any missing values in the dataset? To answer these questions. I use .info function for training and test set:

```
train.info()
print ("-----")
test.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 878049 entries, 0 to 878048
Data columns (total 9 columns):
Dates      878049 non-null datetime64[ns]
Category   878049 non-null object
Descript    878049 non-null object
DayOfWeek   878049 non-null object
PdDistrict  878049 non-null object
Resolution  878049 non-null object
Address     878049 non-null object
X           878049 non-null float64
Y           878049 non-null float64
dtypes: datetime64[ns](1), float64(2), object(6)
memory usage: 67.0+ MB

-----
<class 'pandas.core.frame.DataFrame'>
Int64Index: 884262 entries, 0 to 884261
Data columns (total 7 columns):
Id          884262 non-null int64
Dates       884262 non-null datetime64[ns]
DayOfWeek    884262 non-null object
PdDistrict   884262 non-null object
Address      884262 non-null object
X            884262 non-null float64
Y            884262 non-null float64
dtypes: datetime64[ns](1), float64(2), int64(1), object(3)
memory usage: 54.0+ MB
```

We have total 878049 points in the training and 884262 points in the test set. First thing I noticed is, some features that we have in training set aren't listed in test set. These are: **Category** (normal, that's what we have to predict), **Descript** and **Resolution**.

train.head()

	Dates	Category	Descript	DayOfWeek	PdDistrict	Resolution	Address	X	Y
0	2015-05-13 23:53:00	WARRANTS	WARRANT ARREST	Wednesday	NORTHERN	ARREST, BOOKED	OAK ST / LAGUNA ST	-122.425892	37.774599
1	2015-05-13 23:53:00	OTHER OFFENSES	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	ARREST, BOOKED	OAK ST / LAGUNA ST	-122.425892	37.774599
2	2015-05-13 23:33:00	OTHER OFFENSES	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	ARREST, BOOKED	VANNESS AV / GREENWICH ST	-122.424363	37.800414
3	2015-05-13 23:30:00	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Wednesday	NORTHERN	NONE	1500 Block of LOMBARD ST	-122.426995	37.800873
4	2015-05-13 23:30:00	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Wednesday	PARK	NONE	100 Block of BRODERICK ST	-122.438738	37.771541

test.head()

	Id	Dates	DayOfWeek	PdDistrict	Address	X	Y
0	0	2015-05-10 23:59:00	Sunday	BAYVIEW	2000 Block of THOMAS AV	-122.399588	37.735051
1	1	2015-05-10 23:51:00	Sunday	BAYVIEW	3RD ST / REVERE AV	-122.391523	37.732432
2	2	2015-05-10 23:50:00	Sunday	NORTHERN	2000 Block of GOUGH ST	-122.426002	37.792212
3	3	2015-05-10 23:45:00	Sunday	INGLESIDE	4700 Block of MISSION ST	-122.437394	37.721412
4	4	2015-05-10 23:45:00	Sunday	INGLESIDE	4700 Block of MISSION ST	-122.437394	37.721412

I dropped columns **Descript** and **Resolution** from training data. They seem like not necessary features in our prediction modeling. As we knew from exploratory analysis; **Dates**, **PdDistrict** and **DayOfWeek** are important features and Category output has different results when we play these features. That leads me to add these features in our analysis. For **Dates** feature, I use parser and **pandas get dummies** function to split dates into year, month, hour classes. For **DayOfWeek** and **PdDistrict** features, pandas get dummies function is also helpful to convert categorical data into numeric values.

In Category feature we have total 39 types of crimes. But these crimes are in text format and it is not useful if we put in that format to train our models. So I use sklearn's **LabelEncoder** function to convert categorical data to labels.

- Convert the categorical data to labels

```
from sklearn.preprocessing import LabelEncoder

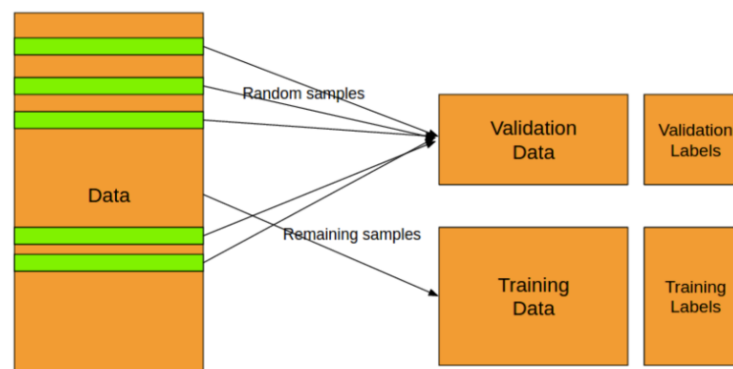
lbl_enc = LabelEncoder()
lbl_enc.fit(xtrain[categorical_features])
xtrain_cat = lbl_enc.transform(xtrain[categorical_features])
```

After data conversion is done for features:

- Month – 12 columns
- Hour – 24 columns
- PdDistrict – 10 columns
- DayOfWeek – 7 columns
- Encoded Category – 1 column

So we have Total: 54 columns. For the start; I don't put X,Y and Address column for now into calculations. As I plotted X,Y coordinates with matplotlib's Basemap library I see that some X,Y coordinates are outliers which are they outside of the Bay area geographically. To overcome this problem; when I put X,Y columns to train models, I make outliers 0 in the training set.

Last thing I do is splitting the data. From Train set, I create another validation data set. In that way I will use train set to fit the model and then use validation set to see model performance. And then finally, I will apply the algorithm into our original test set.



Implementation

I use Python 2.7 environment for all implementations. Python's sklearn machine learning library is very useful, easy and fast to implement. I use Python Jupiter notebook for coding implementation to make debugging easy. First, I import all libraries that I needed for data preprocessing. Then with help of pandas, I opened train and test csv files. Next step is feature engineering. First, I extracted categorical features with **pd.get_dummies** function and created new columns for them. For X,Y coordinates, I use **StandardScaler** to scale all coordinates. There are some coordinates are in outside of Bay Area geographically so I corrected these outliers to 0. With help of **LabelEncoder**, I converted Category column to labels. At the end, I have total 55 columns.

Then the next step is splitting the data. As I mentioned before, best approach is splitting the train set to train and validation sets so we can have more accuracy on results. I use cross validation – **StratifiedShuffleSplit** function to split the data. I chose 80% to 20% split ratio for train and validation sets for implementation. I used that ratio before in my old classification projects and got accurate results so I will stick with it.

```
# Train/test split - 80% training, 20% validation set

from sklearn import cross_validation
from sklearn.cross_validation import StratifiedShuffleSplit

# the number of training points
num_train = 707410 #80% of data points

#the number of testing points
num_validation = X_all.shape[0] - num_train

sss = StratifiedShuffleSplit(y_all, n_iter=2, test_size=num_validation, random_state=40)
for train_index, valid_index in sss:
    print("TRAIN:", train_index, "VALIDATION:", valid_index)
    X_train = X_all.iloc[train_index]
    X_valid = X_all.iloc[valid_index]
    y_train = y_all[train_index]
    y_valid = y_all[valid_index]

len(sss)

# Show the results of the split
print "Training set has {} samples.".format(X_train.shape[0])
print "Validation set has {} samples.".format(X_valid.shape[0])
```

After splitting the data, then I have to choose models and define the classifiers for them. I use 4 models: Random Forest, Naïve Bayes, Logistic Regression and XGB. First I imported model libraries and fit the classifiers for training set. Then make predictions for validation set and calculate log loss metrics for all models.

```
# Random Forest Classifier
from sklearn.metrics import log_loss
from sklearn.ensemble import RandomForestClassifier

clf_A = RandomForestClassifier(max_depth=10, n_estimators=100)
clf_A.fit(X_train, y_train)
clf_probs = clf_A.predict_proba(X_valid)
score = log_loss(y_valid, clf_probs)
print score
```

2.54086994194

After I got log loss score for each model then I apply the same technique to the test set. For probabilities in each class for each model, I converted columns and rows to submission file in .csv format. I noticed that csv files are too big in that conversion so I used `numpy.round(4)` function to decrease the file size. With rounding csv, file sizes decreased to around 500 MB. Then I submit csv files in Kaggle web site to see score for the original test set. As from my submission results, I can say that there is not much difference in log loss score between validation set and the test set in my case.

Refinement

In implementation part, I had to tune some model parameters to get accurate results. In Naïve Bayes algorithm, first I used GaussianNB as a classifier but that resulted log loss score of 13.1. Then I switched to BernoulliNB and my score is improved rapidly to 2.6 range. I think because some features have dependency in each other so GaussianNB didn't perform well for this dataset. For Random Forest algorithm, I used `n_estimators=50` in the beginning and `max_depth=8`. Then when I switched to `n_estimators = 100` and `max_depth=10` I got better results. I didn't increase more in both parameters because it takes too much time to calculate.

In XGB model, I try to use default parameters. I selected three main parameters to optimize the my model. First I set **max_depth**=8 which is maximum depth of a tree, increase this value will make the model more complex / likely to be overfitting. According to most Kaggle competitions 8 value gives the best performance so I will start with that value. **N_estimators**=12 which exists as a parameter in XGBClassifier. However, it has to be passed as "num_boosting_rounds" while calling the fit function in the standard xgboost implementation. **Learning_rate**=0.6 which is step size shrinkage used in update to prevents overfitting in the model. I kept `n_estimators=12` and `learning_rate=0.6` and try to optimize `max_depth` parameter to tune the model. I used GridSearchCV to find the best `max_depth` and below are the results for different values:

```
param_test = {'max_depth':range(4,10,1)}
gsearch = GridSearchCV(estimator = xgb.XGBClassifier(learning_rate=0.6, n_estimators=12),
param_grid = param_test, scoring='log_loss',n_jobs=4,iid=False, cv=2)
gsearch.fit(X_train,y_train)
gsearch.grid_scores_, gsearch.best_params_, gsearch.best_score_

([mean: -2.49606, std: 0.00038, params: {'max_depth': 4},
 mean: -2.48313, std: 0.00059, params: {'max_depth': 5},
 mean: -2.47353, std: 0.00329, params: {'max_depth': 6},
 mean: -2.50796, std: 0.02397, params: {'max_depth': 7},
 mean: -2.48557, std: 0.02545, params: {'max_depth': 8},
 mean: -2.47923, std: 0.00609, params: {'max_depth': 9}],
 {'max_depth': 6},
 -2.473532421269641)
```

Figure 9: Tuning XGBoost

According to GridSeachCV, max_depth=6 gives me the best log_loss score 2.4735. So I will continue with max_depth=6 value instead of 8.

Refinement	Log-loss
Before	2.4789
After	2.4735

IV. Results

Model Evaluation and Validation

In first implementation, I didn't put X,Y coordinates taken into account. For each model, log loss metrics is calculated for validation set from sklearn's metric library. And for test set, results are submitted to Kaggle and obtained score. Here are the results:

Model Performance	Valid set Log Loss	Test set Log Loss
Logistic Regression	2.6607	2.6614
Naïve Bayes (Bernoulli)	2.5808	2.586
Random Forest (100,10)	2.5964	2.5999
XGBoost	2.5795	2.5884

Figure 10: Log Loss scores without X,Y

XGB comes in first place with the best score. Random Forest is second, Naïve Bayes is third and Logistic Regression is the worst performer. Then with StandardScaler and data processing on X,Y coordinates, I added them into account and below are the results:

Model Performance	Valid set Log Loss	Test set Log Loss
Logistic Regression	2.6555	2.6722
Naïve Bayes (Bernoulli)	2.6081	2.6203
Random Forest (100,10)	2.5408	2.5469
XGBoost	2.4708	2.5124

Figure 11: Log Loss scores with X,Y

As we can see, the order of the performers didn't change. But I got improvement on scores. After submitted my best performance from XGBoost which 2.5124 my standing on Kaggle public leaderboard is 740. Even slight improvements make you to jump on higher level in the leaderboard.

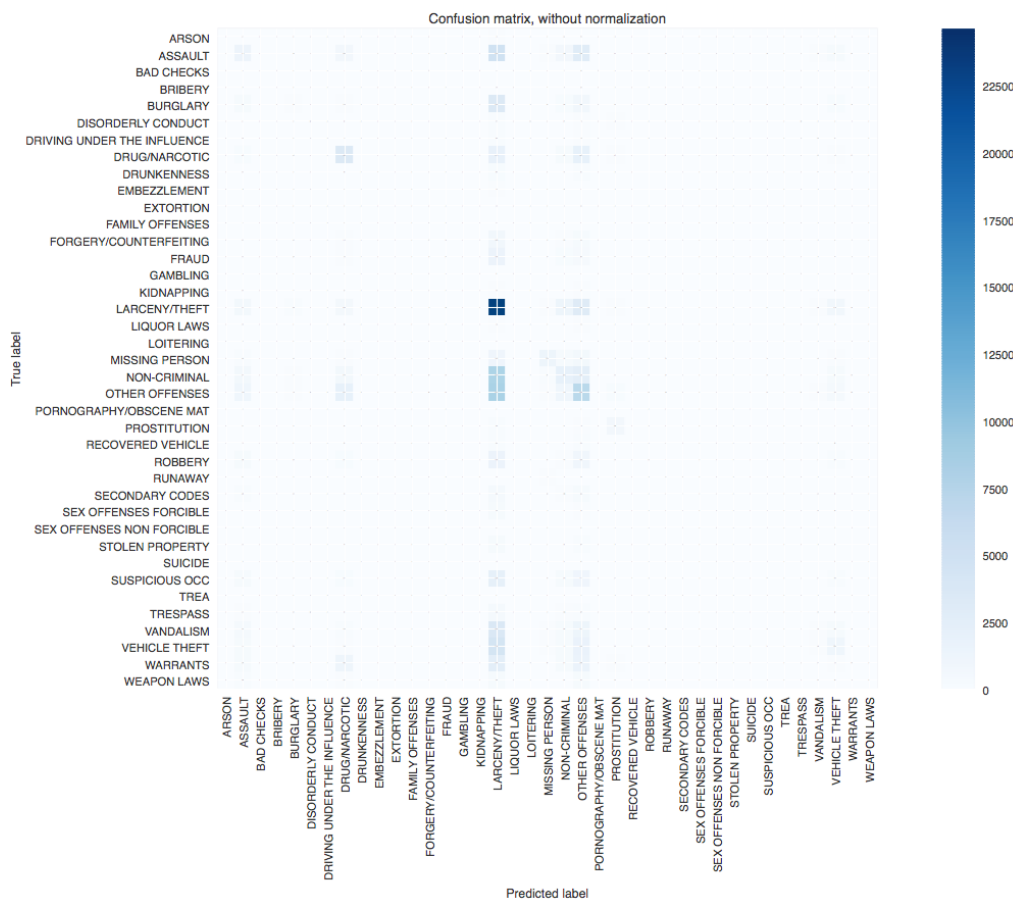
Justification

I chose XGboost as final model and according to benchmark results that I got in the beginning solution is significant enough. According to Kaggle public leaderboard my score is in the top 30%. In the Benchmark section, I mentioned that I will hit the leaderboard under 1000 place and from my best score I already got the spot 740 after post-deadline.

V. Conclusion

Free-Form Visualization

According to our final model XGBoost results, below is the confusion matrix plot without normalization:



In the matrix; X-axis represents predicted labels for all 39 classes and Y-axis represents True labels. As we can see most of the predicted and true labels are intersected over LARCENY/THEFT category. If you remember from our exploratory analysis for our training set from the beginning: Most of the crimes are occurring in this category. So according to our confusion matrix plot, most of our predictions in that category is matching, that means we get accurate results in our analysis.

Reflection

From the beginning, first we started with doing exploratory analysis on our dataset. How features are correlated with each other? Is there any missing values? Is there any outliers? Most of our features are categorical so we don't have to deal with numbers too often. We plotted the graphs which explaining relationships between features. We checked X,Y coordinates and see that we have some outliers in this category where it shows outside of Bay Area. We cleared these outliers to use that features in next steps. With sklearn's data preprocessing packages we did our feature engineering to get ready for our models. At the end of this stage we split our data 80-20% to train and validation sets to get more accurate results.

Since this is a classification problem, we chose 4 different model approach for the problem: Logistic Regression, Naïve Bayes, Random Forest and XGBoost. We trained all models with our training set and apply the classifier in our validation set. Then we checked the log loss scores for each model. According to results, we got the best score on XGBoost model and hits the Kaggle public leaderboard with 740 after post-deadline results.

Improvement

In this project, final model and performance could be improved. In feature selection process, we didn't take into account Address column. According to Kaggle Forums, there is a method called log loss which extracting Address text data into different columns. If we use that method and add Address information to our model we can get better results and decrease log loss. But that Address column extraction also leads to another problem which is too many features we have to deal to train our model. To overcome that, we can apply dimension reduction with help of PCA and again improve the performance. Also playing parameters of XGBoost may also help to get better performance.