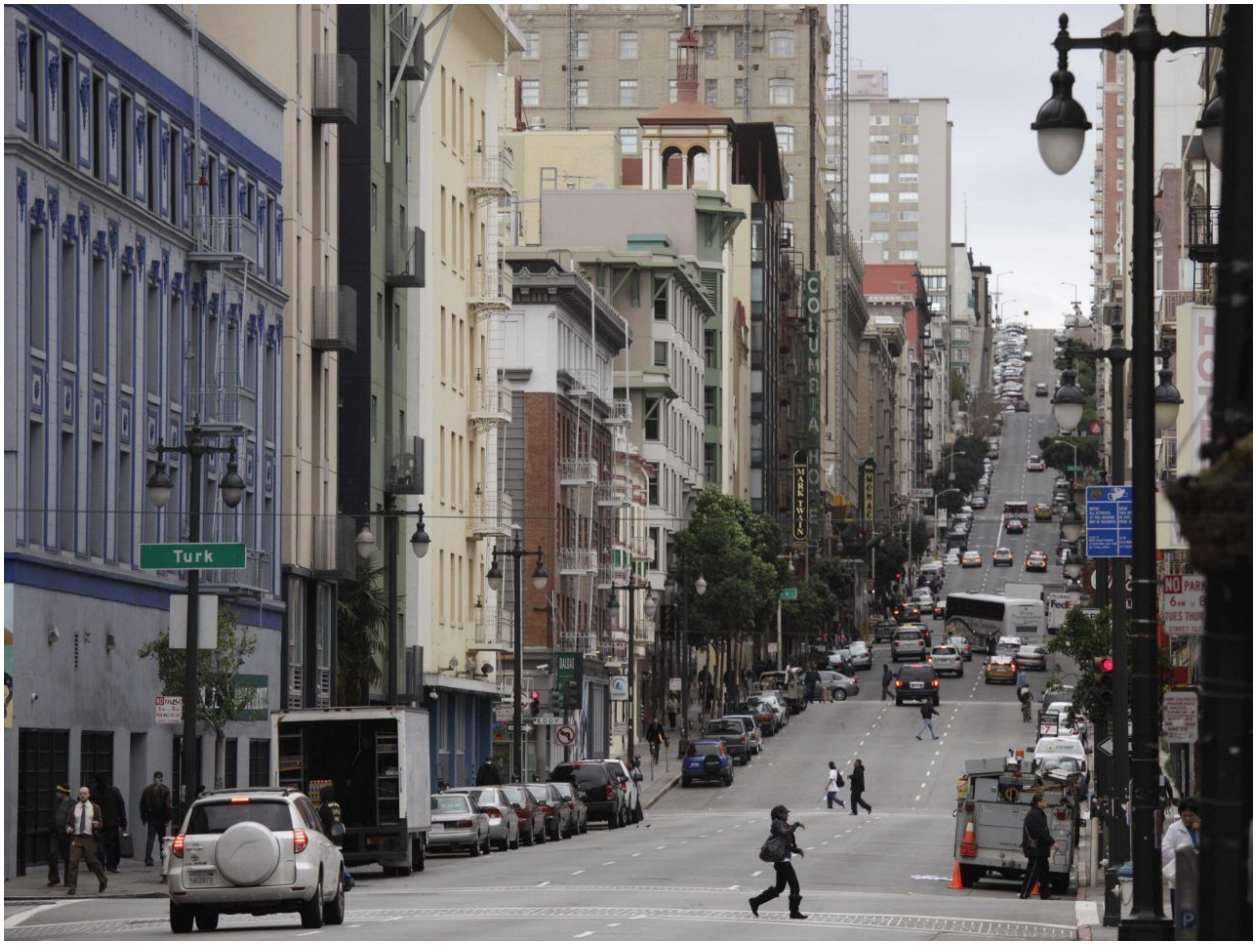
SAN FRANCISCO CRIME PREDICTION

Udacity Machine Learning Engineer Nanodegree Capstone Project



Dominic Wong

INTRODUCTION

Once upon a time I had a short stay in Tenderloin, the so-called “one of the most dangerous neighborhoods” in San Francisco. To be honest, I did not feel very comfortable when I was on the street at night and it would certainly help if I have more information about what is happening in my neighborhood. First of all, knowing which neighborhoods are historically crime-ridden helps me avoid going there. Some people like to try their luck when they hear certain places are not safe, statistics might be able to stop them. However, people just can’t avoid going to certain places sometimes and it will help to notify them in real-time if something bad is happening or given past events they should get out of where they are as soon as possible.

If we can get our hands on San Francisco’s historical crime data and perform analysis against it to understand what happens where and when, it will help us to avoid unwanted losses/injuries/deaths.

I found San Francisco’s crime data between 2003 and 2015 on Kaggle’s “San Francisco Crime Classification”[1] competition and my goal is to first run exploratory analyses on the data to have an understanding on the crimes’ trends. Then, I am going to use a few machine learning methods to predict the probabilities of certain crimes happening given the date, time and location.

The point of doing all these data analyses/machine learning is to help the general public better understand where to avoid in San Francisco, and what one should expect if he has no choice but to visit certain high risk areas. Ultimately, this analytics engine will become a crime alerts/predictions service by SMS/mobile apps/chatbots. I call this SaaS (Safety as a Service).

[1] https://www.kaggle.com/c/sf-crime/data

THE GOAL

The goal is to create a crime alert system for the citizens of San Francisco such that they are more aware of their surroundings and potentially the crime rate will become lower.

There are two main components in the system. The first component is the result of exploratory data analyses against the dataset. Having the crime history summary, we can identity where crimes happen, what kind of crimes happen, what time do crimes happen, etc. The summary can act as a general guideline for the citizens as well as an internal guideline for deciding how often we want to send alerts to the users and what kind of crime should be qualified for an alert. The second component will be the prediction service. If a citizen is subscribed to our service and he happens to step on a not so safe neighborhood, we should be able to predict the kind of crimes he might run into, given the time and his location. This prediction service will be built on top of a supervised learning algorithm because to predict the category of a crime is a classification problem. We are going to run naive Bayes classifier, multinomial logistic regression and decision tree classifier to see which model has the best performance and optimize the model with the highest accuracy. They are the appropriate choices because we are going to need the most likely crime category's probability per case and compare it with a predefined threshold. All three models have the abilities to predict the probabilities of each crime category for each case.

The final product will be a crime alert system that alerts subscribed citizens if he steps foot in a dangerous area. The system will predict the crime the citizen will run into if the probability passes a predefined threshold.

MEASURING THE SYSTEM'S PERFORMANCE

The second component of the system is a supervised learning problem. The system has to predict what kind of crime the user is likely to run into given his location and time. I am going to explore a few machine learning methods and see which one works best. In order to compare each machine learning model's accuracy, we have to give each of them a score. The alert system requires the prediction to return the probability for each crime category and we need to use a score that is able to take in probabilities. I am going to use the multi-class logarithmic loss loss to determine each model's accuracy.

Multi-class logarithmic loss loss is defined as

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

, where N is the number of cases in the test set, M is the number of class labels, log is the natural logarithm. yij is 1 if observation i is in class j and 0 otherwise, and pij is the predicted probability that observation i belongs to class j.[2] The lower the total loss, the more accurate the predictions are because the lower the pij (for 0 < pij <= 1), the greater the absolute value of log(pij).

I am also going to run a grid search for the chosen model if it is parameterized to determine the best parameters combination that creates the most accurate model.

[2] https://www.kaggle.com/c/sf-crime/details/evaluation

THE ANALYSIS

I downloaded the dataset from Kaggle's "San Francisco Crime Classification" competition. The file "train.csv" contains crimes that happened in San Francisco between 2003 and 2015. I could not use the file "test.csv" because it does not have labels and it is only for Kaggle competitors to predict crimes as part of the competition.

THE DATASET

Each row in the dataset represents a case. It has the following columns:

Dates (Date and time), Category, Description, Day Of Week, Police Department District, Resolution (Was the criminal arrested?), Address, X, Y (Latitude and longitude)

FILTERING THE DATA

I decided to only include the data between 2013 and 2015 because a city's overall environment is always changing because of various reasons such as urban development and economic conditions. For San Francisco, the tech boom has changed the population structure of the city and is likely to affect how and where crimes are done. It implies old crimes are less relevant than recent crimes.

I am also going to remove crimes with irrelevant categories because the goal is to alert citizens when there are real-time danger. There are crime categories in the dataset like "Fraud" or "Embezzlement", which are not something a normal citizen walking down the street would be concerned about. The categories I think are relevant are "Larceny/Theft", "Vehicle Theft", "Robbery", "Assault", "Weapon Laws", "Burglary", "Drunkenness", "Drug/Narcotic", "Kidnapping", "Driving Under The Influence", "Sex Offenses Forcible", "Arson", "Sex Offenses Non-Forcible" and "Extortion". These kinds of crimes directly or indirectly cause physical danger/potential property loss to citizens.

EXPLORATORY DATA ANALYSIS

First, let's take a look at the top 5 crime categories in San Francisco.

LARCENY/THEFT	44564
ASSAULT	14859
VEHICLE THEFT	8199
BURGLARY	7166
DRUG/NARCOTIC	6837

Table 1. Top 5 crime categories in San Francisco

3 out of 5 of the top crimes involve stealing other people's properties. Assault is the second most common crime, which it is usually a way to get the victims to give the attackers their personal properties.

It appears San Francisco's main problem is related to stealing, which might have to do with the widening wealth gap in the city that comes with the tech boom. Many people can no longer afford to live in the city and they have to take illegal measures to survive.

The top 7 crime categories are accountable for (87933 / 91198 * 100 = 96.4%) of the total crimes, so I am going to exclude the rest as they are outliers that will hurt the model's performance.

Next, let's look at where the crimes happened.

SOUTHERN	17743
NORTHERN	12209
MISSION	10568
CENTRAL	10452
BAYVIEW	7483
INGLESIDE	7401
TENDERLOIN	6572

TARAVAL	5787
PARK	4860
RICHMOND	4858

Table 2. San Francisco's crime locations ordered by count

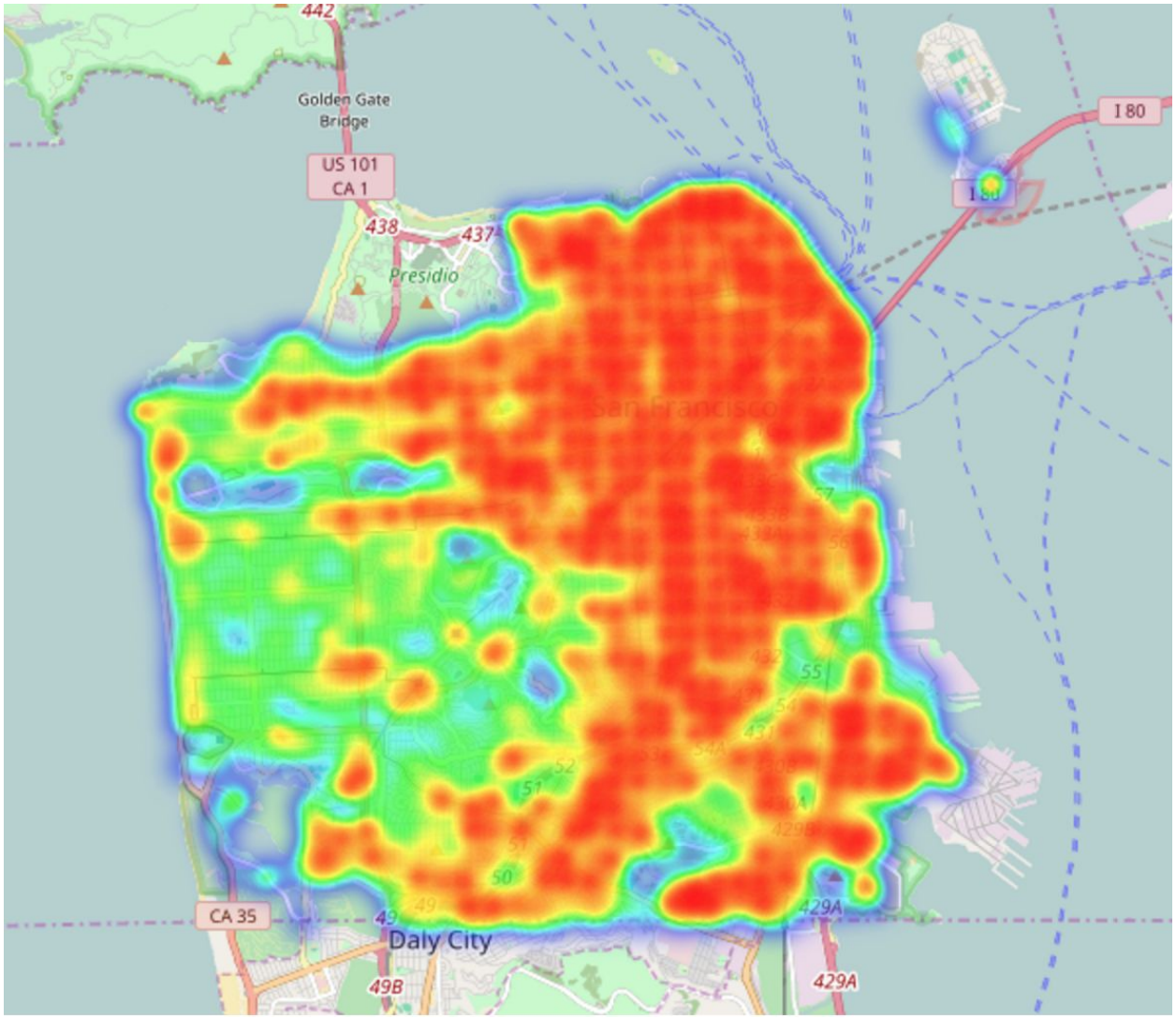


Fig. 1. San Francisco 2013 - 2015 crime heatmap by latitude and longitude

Looking at the heatmap, we can see that most crimes happened on the east side of the city. It means we should put more emphasis on that part by advising citizens to not be there at times when crime rates are high and potentially give more real-time alerts to users when they are there. The crimes at Taraval, Richmond and Park are relatively

low. They are the residential areas, whereas the eastern part of San Francisco is where the commercial and touristy areas are located.

Now, let's look at when these crimes happen.

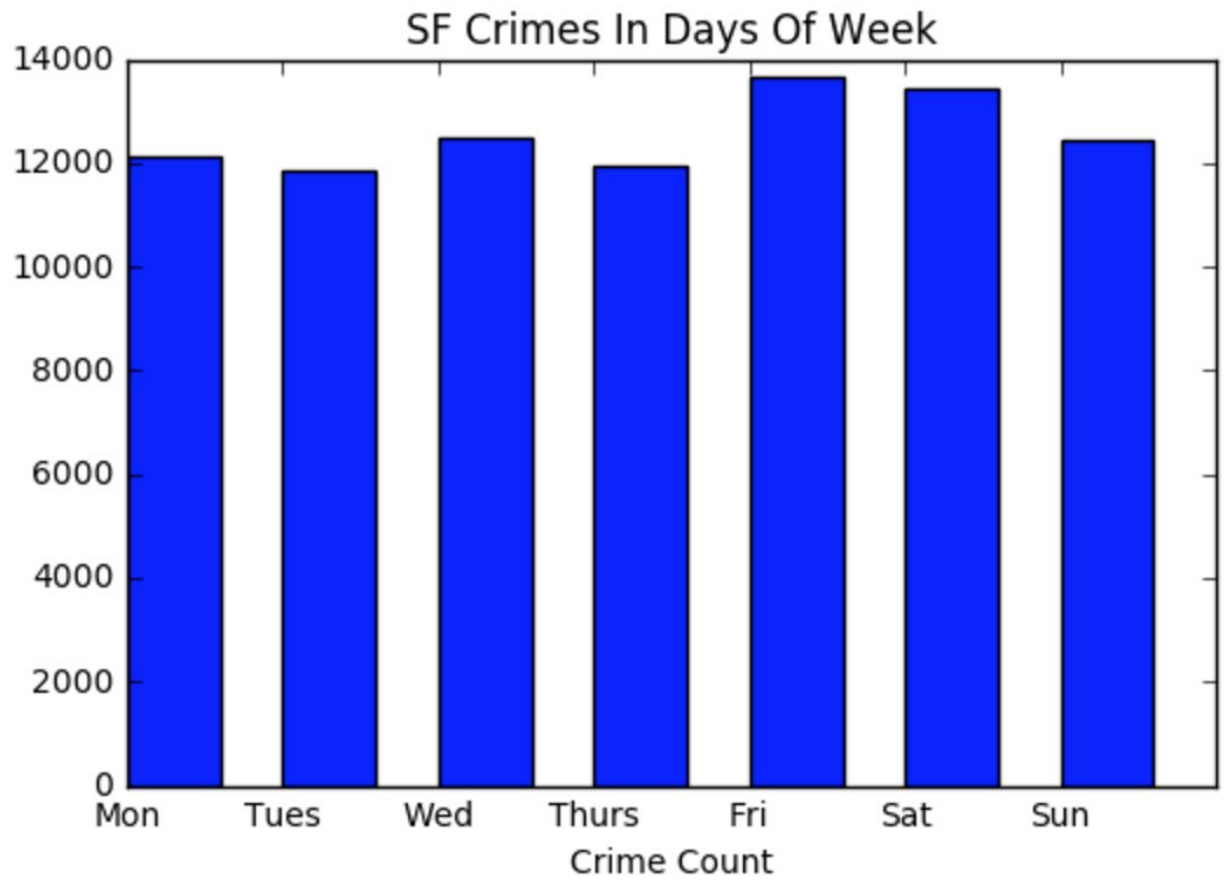


Fig. 2. San Francisco Crimes by Day of Week

MONDAY	12146
TUESDAY	11838
WEDNESDAY	12474
THURSDAY	11965
FRIDAY	13651
SATURDAY	13422
SUNDAY	12437

Table 3. San Francisco's crimes grouped by day of week

It looks like the crime counts are evenly distributed during the days of week, with slightly more crimes happening on Friday and Saturday. A possible reason is more people go out during the weekends.

There are not really any insights from the graph above, let's see if it makes a difference by segmenting the crimes by hour.

The dataset does not have a column called "Hour", but it has a column called "Dates" in the format "yyyy-mm-dd hh:mm:ss". We need to create a new column called "Hour" by extracting the hour from the Dates column. The code to create the Hour column looks like this:

```
# df is a pandas DataFrame
df.Hour = df.Dates.apply(lambda date: int(date.split(" ")[1].split(":")[0]))
```

With the new Hour column, we can group the crimes by the hour.

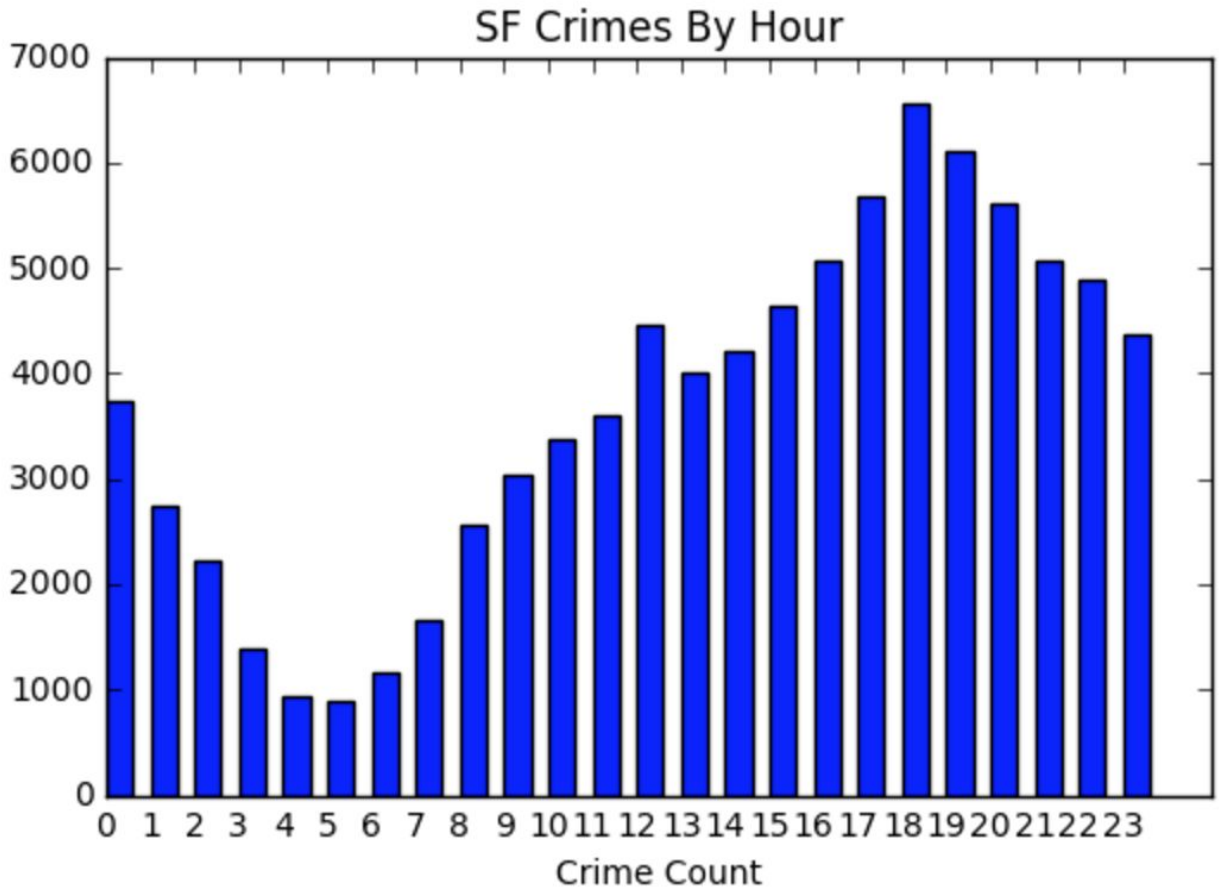


Fig. 3. San Francisco Crimes by Hour

The statistics agrees with the traditional wisdom that there are more crimes when it is dark, with most of the crimes happening between 6PM and 12AM. Surprisingly, there were the most number of crimes between 6 and 7 PM but not later in the night.

Based on the analysis' result, the predictive system should be configured to be more active in pushing alerts to a user if he appears in the east side of San Francisco between 6PM and 12AM.

SELECTING THE PREDICTIVE MODEL

Now we have the first component taken care of by the exploratory data analysis, it is time to build a predictive model. We have a classification problem and we are going to pick the best algorithm among naive Bayes classifier, multinomial logistic regression

and decision tree classifier.

Naive Bayes classifier is chosen because calculating the probability for each type of crime to happen given a priori condition and picking the one crime that has the highest chance of happening is basically the Bayes' theorem. It is defined as

$$P(Crime \mid LocationTime) = \frac{P(LocationTime \mid Crime)\, P(Crime)}{P(LocationTime)}$$

It is known for being fast to train and it has a surprisingly good real-world predictive power even with a naive assumption that the each pair of features are independent.

Multinomial Logistic Regression is chosen because the prediction's outcome is categorical (crime type) and it is an one versus all problem (only one crime type will be selected in the end). Logistic regression uses the sigmoid function to calculate a vector's value. If the value is greater than or equal to the predefined threshold, it returns the positive crime type. Otherwise, it returns the negative crime type. The sigmoid function is defined as

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

, where θ is the vector and $\theta^T x$ is the trained equation.

Decision Tree Classifier is chosen because it is simple to implement and understand. It can be represented as a diagram that encapsulates multiple logic statements.

E.g.

```
If it is in the evening between 6PM and 12AM at Tenderloin, the predicted crime will be
DRUG/NARCOTIC.

If it is in the afternoon between 12PM and 6PM at SOUTHERN, the predicted crime will be
LARCENY/THEFT.
```

RUNNING THE COMPARISONS

Before running the predictions, we first have to select and transform the useful features. I have identified that time and location are the two most important criteria in determining the type of crime. The rest are not useful for various reasons and it will be explained in the table below:

Descript	It is a more specific description of the crime's category and it varies from crime to crime.
Resolution	Resolution is the outcome of each crime and one cannot predict the type of crime given the criminal "will be arrested". It is not logical.
Year	The crime's year is unlikely to provide any patterns

Table 4. Reasons for ignoring certain features in the dataset

I am going to include "Dates", "DayOfWeek", "PdDistrict", "Address" columns as my features. However, the features require transformation before being fed to the algorithms. Here are a few things I would do:

1. Break "Dates" into "Month", "Day", "Hour" and "Minute"
2. Transform "DayOfWeek" and "PdDistrict" into numbers and potentially doing one-hot encoding
3. The "Address" column comes in 2 formats. The first type is "some number `of` some street" and the second type is "some street / some street". The first type indicates the address is in the middle of the street and the second type indicates the address is an intersection of two streets. There will be a new column called "IsIntersection" with the value 1 (True) or 0 (False) given the address' format.
   I also attempted to create a new column called "Street". For this new column, it will either store the intersection's street names or the street name without the number. So if we have an address "500 Block of COLLEGE AV", the street column will store it as "COLLEGE AV". It makes the address column slightly more generic and I found out from experiments that the F1 score would improve by around 1%. Unfortunately, it was not as generic as I wished it to be and I ended up having 7857 columns after applying one-hot encoding. The models were too slow to justify the 1% gain in accuracy. In the end, the transformation of "Address" column will only result in an "IsIntersection" column.
4. Round "X" and "Y" to 3 decimal places to group crimes by locations. When we try to predict a crime, the predictive model can look at the crimes that happened within ~111m of the user's current location.

The actual experiment involves looping through a test size between 0.1 and 0.9 with +0.1 per step to do a train test split on the dataset. The training dataset will then be fitted to each of the algorithm and the fitted model will try to make predictions with the test dataset. Finally, a F1 score will be obtained by comparing the predictions and actual results.

```
for test_size in np.arange(0.1, 1, 0.1):
    X_train, X_test, y_train, y_test = train_test_split(X, df.Category, test_size=test_size, random_state=42)
    y_test = pd.get_dummies(y_test)

    for clf in [
        DecisionTreeClassifier(random_state=42),
        LogisticRegression(random_state=42),
        GaussianNB()
    ]:
        clf.fit(X_train, y_train)
        y_pred = clf.predict_proba(X_test)
        scores = []
        for i in np.arange(0, len(y_pred)):
            score = logloss(y_test.iloc[i].values, y_pred[i])
            scores.append(score)

        final_score = sum(scores) / len(scores)
        results.append([clf.__class__.__name__, test_size, final_score])

classifiers_results_df = pd.DataFrame(columns=["Classifier", "Test Size", "Score"], data=results)
classifiers_results_df.sort(["Classifier", "Test Size"], inplace=True)

display(classifiers_results_df)
```

	Classifier	Test Size	Score
0	DecisionTreeClassifier	0.1	5.043456
3	DecisionTreeClassifier	0.2	5.023023
6	DecisionTreeClassifier	0.3	5.129891
9	DecisionTreeClassifier	0.4	5.237796
12	DecisionTreeClassifier	0.5	5.403235
15	DecisionTreeClassifier	0.6	5.474860
18	DecisionTreeClassifier	0.7	5.632627
21	DecisionTreeClassifier	0.8	5.816798
24	DecisionTreeClassifier	0.9	6.005511
2	GaussianNB	0.1	0.329608
5	GaussianNB	0.2	0.327854
8	GaussianNB	0.3	0.329946
11	GaussianNB	0.4	0.331748
14	GaussianNB	0.5	0.332662
17	GaussianNB	0.6	0.332150
20	GaussianNB	0.7	0.328626
23	GaussianNB	0.8	0.327252
26	GaussianNB	0.9	0.327859
1	LogisticRegression	0.1	0.315284
4	LogisticRegression	0.2	0.314054
7	LogisticRegression	0.3	0.315079
10	LogisticRegression	0.4	0.316539
13	LogisticRegression	0.5	0.316223
16	LogisticRegression	0.6	0.316620
19	LogisticRegression	0.7	0.316928
22	LogisticRegression	0.8	0.317451
25	LogisticRegression	0.9	0.317803

Table 5. Benchmarking each classifier using multi-class logarithmic loss

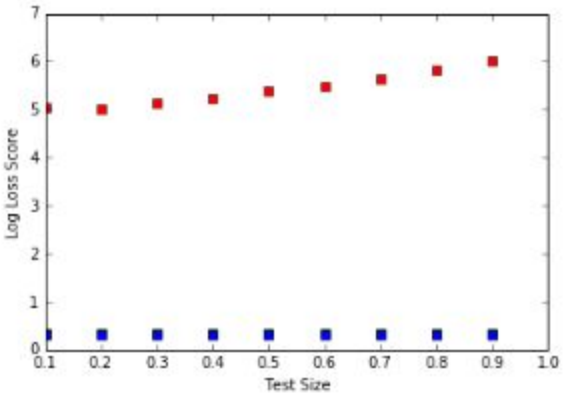


Fig. 4. Benchmarking each classifier using multi-class logarithmic loss (Naive Bayes classifier and multinomial logistic regression has similar scores, hence the blue squares covering the green squares, the red squares belong to decision tree classifier)

Classifier	Test Size	Log Loss Score
Decision Tree Classifier	0.2	5.023023
Naive Bayes Classifier	0.8	0.327252
Logistic Regression	0.2	0.314054

Table 6. Each classifier's best log loss score

Decision tree classifier is not doing so well and I think the reason is that predicting what kind of crime will happen is not simply "draw a tree and if you show up at certain areas at certain times this is what you will run into".

As we can see, logistic regression has the best log loss score. So let's go with logistic regression and try to optimize it.

OPTIMIZING THE SELECTED MODEL

We are going to run a grid search with C (the inverse of regularization strength) as the parameter to be explored and we we are also going to try different datasets balancing strategies because the dataset is skewed to "LARCENY/THEFT" and "ASSAULT".

```
from imblearn.combine import SMOTEENN
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import EditedNearestNeighbours

balancer = EditedNearestNeighbours()
y = df.Category
X_resampled, y_resampled = balancer.fit_sample(X, y)
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

best_clf = None
best_logloss_score = float("inf")
best_C = None

for C in [0.1, 1, 10, 100, 1000]:
    clf = LogisticRegression(random_state=42, C=C)
    clf.fit(X_train, y_train)
    y_pred = clf.predict_proba(X_test)
    final_score = overall_logloss(pd.get_dummies(y_test), y_pred)

    if final_score < best_logloss_score:
        best_clf = clf
        best_logloss_score = final_score
        best_C = C
```

Rebalancing Method	Description	Best Log Loss Score
None	Don't do anything to the dataset	0.314054
Random Over Sampler	Randomly create samples from the minor classes	0.390301
SMOTE + ENN	Create synthetic samples from minor classes	0.189794
Random Under Sampler	Randomly drop samples from the major classes	0.392280
ENN Undersampling	Undersampling based on edited the nearest neighbor method	0.171382

Table 7. Best log loss score for each rebalancing method and their descriptions

The best tuned model's log loss score is 0.171382, which is a 45.4% improvement from the original model. The optimal C value is **1000**, which indicates a weaker regularization. The more the regularization, the more the model punishes overfitting. We end up with a large C

value probably because the data is underfitting the model and it needs less regularization to prevent underfitting. The appeared optimal rebalancing strategy is ENN Undersampling because it results in the lowest log loss score, but we should also look at other accuracy metrics (aka F1 score) instead of only log loss score.

```
from sklearn.metrics import classification_report
print classification_report(y_test, best_clf.predict(X_test))
```

	precision	recall	f1-score	support
ASSAULT	0.31	0.07	0.11	2904
BURGLARY	0.00	0.00	0.00	1416
DRUG/NARCOTIC	0.00	0.00	0.00	1374
LARCENY/THEFT	0.52	0.98	0.68	9015
ROBBERY	0.00	0.00	0.00	946
VEHICLE THEFT	0.24	0.00	0.01	1602
WEAPON LAWS	0.00	0.00	0.00	330
avg / total	0.34	0.51	0.37	17587

Fig. 5. Crime Classification Report without rebalancing the data

```
from sklearn.metrics import classification_report
print classification_report(y_test, best_clf.predict(X_test))
```

	precision	recall	f1-score	support
ASSAULT	0.18	0.10	0.13	8803
BURGLARY	0.23	0.46	0.31	8873
DRUG/NARCOTIC	0.24	0.20	0.22	8976
LARCENY/THEFT	0.25	0.25	0.25	8949
ROBBERY	0.28	0.19	0.22	8945
VEHICLE THEFT	0.31	0.38	0.34	8914
WEAPON LAWS	0.24	0.18	0.21	8930
avg / total	0.25	0.25	0.24	62390

Fig. 6. Crime Classification Report with random over sampler

```
from sklearn.metrics import classification_report
print classification_report(y_test, best_clf.predict(X_test))
```

	precision	recall	f1-score	support
ASSAULT	0.00	0.00	0.00	101
BURGLARY	0.00	0.00	0.00	7
DRUG/NARCOTIC	0.00	0.00	0.00	101
LARCENY/THEFT	0.68	0.24	0.35	2053
ROBBERY	0.00	0.00	0.00	933
VEHICLE THEFT	0.00	0.00	0.00	27
WEAPON LAWS	0.74	0.98	0.85	7886
avg / total	0.65	0.74	0.67	11108

Fig. 7. Crime Classification Report with SMOTE + ENN

```
from sklearn.metrics import classification_report
print classification_report(y_test, best_clf.predict(X_test))
```

	precision	recall	f1-score	support
ASSAULT	0.19	0.15	0.17	348
BURGLARY	0.21	0.53	0.30	332
DRUG/NARCOTIC	0.17	0.13	0.15	351
LARCENY/THEFT	0.28	0.23	0.25	405
ROBBERY	0.26	0.13	0.17	371
VEHICLE THEFT	0.34	0.37	0.35	377
WEAPON LAWS	0.24	0.18	0.20	371
avg / total	0.25	0.24	0.23	2555

Fig. 8. Crime Classification Report with Random Undersampling

```
from sklearn.metrics import classification_report
print classification_report(y_test, best_clf.predict(X_test))
```

	precision	recall	f1-score	support
ASSAULT	0.31	0.07	0.11	2904
BURGLARY	0.00	0.00	0.00	1416
DRUG/NARCOTIC	0.00	0.00	0.00	1374
LARCENY/THEFT	0.52	0.98	0.68	9015
ROBBERY	0.00	0.00	0.00	946
VEHICLE THEFT	0.24	0.00	0.01	1602
WEAPON LAWS	0.00	0.00	0.00	330
avg / total	0.34	0.51	0.37	17587

Fig. 9. Crime Classification Report with ENN Undersampling

I ran the classification report on the prediction model and it turns out the model trained by the ENN undersampled dataset has a pretty low F1 score. It is difficult to be justified as the best rebalancing approach. Other ways to rebalance the dataset also bring interesting changes to the log loss score/F1 score. Randomly over-sampling and random under-sampling the dataset actually gives us poorer log loss scores even though the F1 score has become more evenly distributed. SMOTE + ENN does not help improving the prediction for other crime categories and it even worsens the F1 score for predicting “LARCENY/THEFT”. However, the average F1 score increased from 0.37 (without rebalancing) to 0.67 and the log loss score is very close to the model trained by ENN Undersampling. SMOTE + ENN has the second-best log loss score and the highest average F1 score and so I believe it to be the best approach to rebalance the dataset.

SMOTE + ENN is an oversampling method that creates synthetic samples from the minor classes. The algorithm finds similar samples and modifies a sample’s attributes one at the time with a random amount within the difference among itself and its neighboring samples.

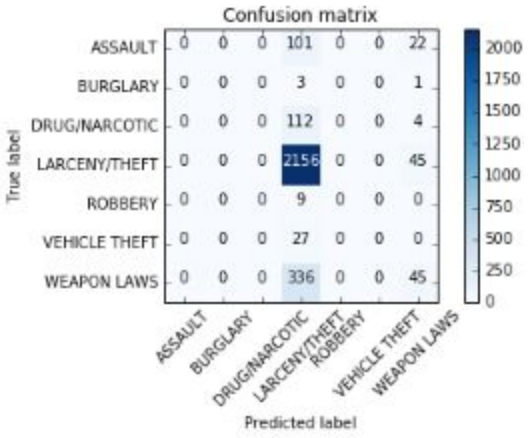


Fig. 10. Confusion matrix for the final model

The confusion matrix for the final model shows that it is pretty good at predicting LARCENY/THEFT and somewhat capable at predicting WEAPON LAWS. It is also likely to misclassify ASSAULT, DRUG/NARCOTIC and WEAPON LAWS as LARCENY/THEFT. BURGLARY, ROBBERY AND VEHICLE THEFT seem not appear enough in the test set for us to conclude that the model is likely to misclassify them as other crime categories, although the model failed to predict these crime categories completely.

CONCLUSION

The “San Francisco Crime Classification” dataset does not appear to be complicated and we ended up with 9 features after transforming and removing columns. We ended up with a log loss score of 0.189794. It is a pretty good score, but there are still rooms for improvement. I believe there are a few reasons why crime is not easy to predict. First of all, the model we ended up with is most likely underfitting the dataset as we can only rely on time and location as features. Features such as “Descript” can potentially have unlimited number of “categories” and they will not be able to help generalizing the dataset. On the other hand, even though the “Resolution” column appears to be categorical, I don’t think it should be included in the prediction model because the alert system’s goal is not to “predict” past crime, but to predict crime that can possibly happen. How can you say the criminal has been arrested if the crime has never happened? The user’s smartphone will only be able to provide his current latitude and longitude. With limited number of features, it is difficult to reach the optimal model.

The dataset definitely required preprocessing before being used for training. The columns “Dates”, “Address”, “X” and “Y” have useful information that might be useful to increase prediction accuracy. In fact, after many trial and error, I found that my current feature transformation from the “Dates”, “Address”, “X” and “Y” columns are the most appropriate. It has close to the lowest log loss score and it takes much less time to train the model. For “Dates”, I included all units down to the minute except “Year”, with “Minute” rounded to the nearest ten minutes. Training the model down to the second sounds like overfitting to me. For “Address”, I created a new column called “IsIntersection” and it did improve the model’s accuracy. However, it was not a good decision to try to train the model with the actual addresses as well because there were simply too many addresses (my model ended up having 7000+ features even after removing the streets’ numbers). The model’s accuracy improved, but it was much slower than a model without this feature. The accuracy gain is not big enough to tolerate the increase in training time. For “X” and “Y”, I rounded the degree to the nearest 3 decimal place for an ~111m location accuracy.

To build an actual crime prediction alert system, I think this is good to go as a starter but it definitely needs a better accuracy because people react differently to crime alerts with different crime categories. For instance, if the crime alert tells me there can be larcenies in the area, I will probably pay less attention than if the crime alert tells me

there can be assaults or even fatal crimes in the area. Since serious crimes happen much less often than petty crimes, if the predictive model fails to alert me properly and tells me there will only be petty crimes while I can actually run into murders, my life is at risk. The alert system should also try not to bother the users too much. As we know, crimes happen all over San Francisco. People still need to go out and do things. There has to be a limit imposed on the system such that it does not over-alert the users. An alert should not be sent to the user if the probability does not exceed a threshold (Naive Bayes classifier will most likely not be used as it is known for not being good at returning accurate probabilities).

To further improve the system, not only do we have to improve the prediction accuracy, the probabilities accuracies must also be improved. Without accurate probabilities, we will still be sending false positives or not sending true positives to users. To get better probabilities, we need to include more relevant features. Using only the location is insufficient because it changes with time. For example, if some Silicon Valley billionaires suddenly decide to put their companies' headquarters and employees housing in the "Southern" police district, which has the highest number of crimes, suddenly the Southern district will become an affluent area and drive people who cannot afford it out of the area. The crime data from before the new communities are formed then becomes irrelevant. Including other features such as the district's average income or the average price of a house might improve the prediction model's accuracy.