Final Project: SF Crime Classification

Sheng-Chu, Chou (USC ID: 1026-3739-74)
Wen-Kai, Tsai (USC ID: 2290-7257-59)
Zenas Huang (USC ID: 8806-6035-51)
Mrinal Gupta (USC ID: 9772-7150-99)

1. DATASET

1.1 Description

The dataset contains incidents derived from the SFPD Crime Incident Reporting system, which classifies different crime incidents in San Francisco ranging within the time period from 1/1/2003 to 5/13/2015. Given time and location data, our goal is to attempt to predict the type of crime that occurred.

There are 878049 listings in total, and for each listing, there are nine attributes provided in the dataset. The following section provides some details about the name of each attribute and provides a brief description of how the dataset was processed.

1.2 Feature Engineering

- Observe that no NaN values are present in this dataset.
- Remove 2323 duplicate entries.
- Transform the time data into Datetime and extract the year, month, day, hour, and minute into separate fields.
- Drop "descript" and "resolution" from the train data since those features are not in the test data.
- Extract useful information from the Address: marked as "0" if the incident has taken place in a crossroad, otherwise (in building block), marked as "1"
- Account for the periodicity of months and days of week by creating the *number of days* and *week of years* since the first day in the data**.**
- Encode the incident time into day (1) and night (0) to mitigate risk of overfitting.

1.3 Data Visualization

We first display the accumulated density distribution of each crime. The plot [*Fig. 1.1*] reveals that some crimes are widely dispersed across the city, while others are more concentrated in certain areas.

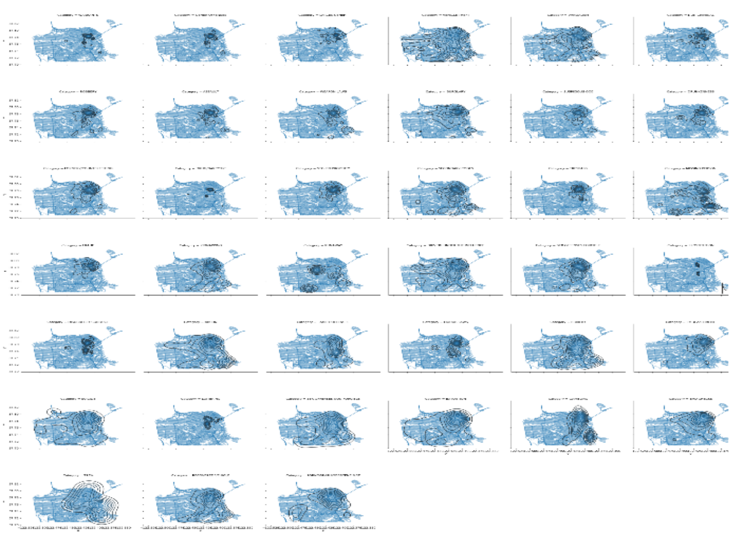

Figure 1.1 Geographic distribution of each crime type on map of SF

From *Fig 1.2*, another distinguishing feature that can be seen in this dataset is the frequency of crime incidence. Some types of crimes happen very frequently in San Francisco, while the others occur occasionally. We can also observe the trends in how different types of crimes fluctuate over the training set date range while others remain relatively stable.

Figure 1.2 Mean and variance of each crime type (year-over-year)

1.4 Feature Selection

Some of the data pre-processing steps we used were to convert categorical features into one-hot encoding features. We use the following features to build our model:

- **Geographical Features**: *latitude, longitude, PdDistrict, and Block*
- **Time Features**: *Year, Month, Day, Hour, Minute, IsDay, DayOfWeek, NumberOfDays,* and *WeekOfYear*

After getting these variables as dummies, there were a total of 28 features for us to use in building our models. Furthermore, since it is a multi-class classification task, the competition stipulates mean log loss as its performance evaluation metric. As an additional performance metric for model evaluation, we also include the Jaccard Index scores.

2. MODEL

2.1 Algorithms and Model

The specific problem is a typical multiclass classification problem, and there are several categories of algorithms for solving it. Initially, we evaluated several appropriate algorithms from **Linear Models (Logistic Regression), Naive Bayes, K Nearest Neighbors, Ensemble methods (Random Forests & Gradient Boosting) and Boosting Algorithms (XGBoost)** using feature engineering and the default parameters to evaluate if any of them shows a significant baseline performance advantage. As a result of our experiments, we concluded the following:

- We found that the linear model was unable to capture the non-linearity in the data, resulting in relatively poor performance on the classification task.
- As for ensemble methods, we attempted gradient boosting, random forest, and extreme gradient boosting. The Gradient boosting model required a surprisingly large amount of time to train. While it achieved relatively adequate performance in explaining the data, we decided not to use it for tuning due to the limitations on our available computing resources.
- Random forest was a good choice because it circumvented the decision tree models tendency to overfit the training set. Additionally, it was also

relatively inexpensive to train which made it an ideal candidate for parameter tuning.

- Finally, extreme gradient boosting required an acceptable amount of time in training since it supported GPU acceleration. Along with its comparatively strong performance, we decided to also use it for tuning alongside the Random Forest model.

2.2 Results Table

The table shows that the ensemble methods have an overall better performance than the other models in this dataset. We chose Extreme Gradient Boosting and Random Forest to refine their parameters given their lower test cross entropy and training time requirements.

Table 2.2 (test) Jaccard Index and (test) Cross Entropy

	Algorithm	Parameters	(test) Jaccard Index	(test) Cross Entropy
2	Gradient Boosting	default	28.57%	1.0574
3	Extreme Gradient Boosting	default	27.71%	1.0449
1	Random Forest	default	27.26%	0.4045
0	Logistic Regression	solver=[lbfgs], multi_class=[multinomial]	23.63%	1.1063
5	K-Nearest Neighbors	n_neighbors = 10	20.31%	Unable to calculate
4	Naive Bayes	default	0.92%	13.4938

3. Hyperparameter Tuning

3.1 Chosen Parameters

Since cross validation required a tremendous amount of memory space, our resource constraints compelled us to split the process into 2 steps: first we find a reasonable number of trees (n_estimators) given a fixed parameters according to the rules of thumb that we researched online; then we adjusted the max_depth afterwards. Again, due to computational resource constraints, tuning over 3 parameters was found to be prohibitively expensive, so we decided to tune only these two parameters.

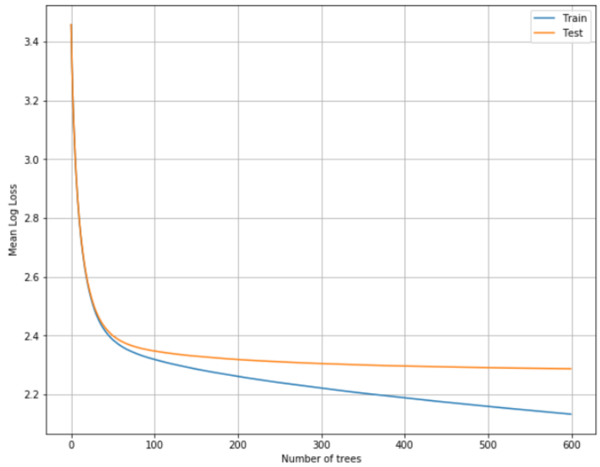

Figure 3.1 No significant improvement after 100 trees.

Although Random Forest had a slight advantage over other models on an efficiency basis, its' memory requirements still exceeded our maximum memory space. Therefore, we again split the process into two steps and found our optimal parameters for XGBoosting and random forest to be 100 estimators and 50 estimators for XGBoosting and Random Forest respectively, both with max_depth 3.

Table 3.1 Optimal Parameters for XGBoosting and Random Forest found by hyperparameter tuning

Parameters	XGBoosting	Random Forest
n_estimators	100	50
max_depth	3	3

3.2 Result Analysis

Fig. 3.2 below corroborates the observation in Fig. 1.1 that the geospatial coordinates (X, Y) of a crime incident plays an important role in classifying its crime type. Moreover, our engineered feature *n_days* was also found to have a surprisingly significant role in predicting the crime class which suggests that there may be a significant temporal pattern in the crime incident types.

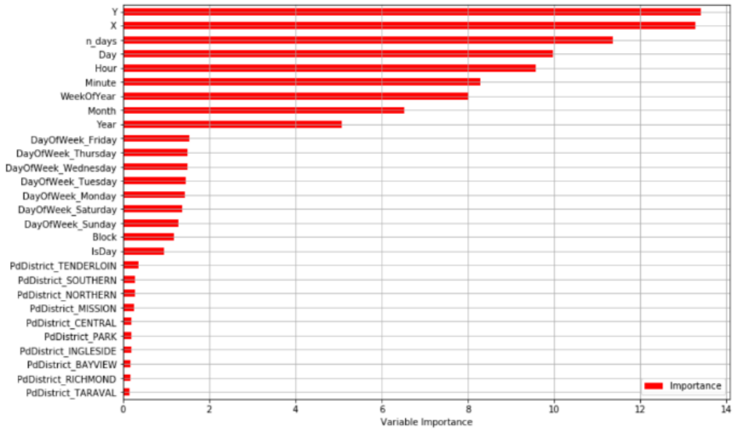

Figure 3.2 Feature importance weighting assigned by the Random Forest Model

3.3 Kaggle Result

After tuning the parameters, we used cross validation to compare the result of two models:

Random Forest: 2.91

Extreme Gradient Boosting: 5.33

Since we found that Random Forest had a lower mean log loss, we submitted the Random Forest model. The final score from Kaggle was determined to be: 2.58605.

For comparison purposes, we also decided to submit the result of XGB and found it yielded a better score of 2.35412:

We suspect that the reason for this unexpected outcome may be due to the fact that we engineered a set of temporal features (*n_days*, *WeekofYear*, etc.) that were better classifiers than many of the geographic features but that these engineered temporal features did not generalize well to the Kaggle evaluation data. In essence, our Random Forest model had "overfit" the provided test data and, in so doing, by excluding important geographic features (due to the Random Forests efforts to reduce overfitting of the train set) such as *PdDistrict_ABC etc*., this had the effect of introducing omitted variable bias into the Random Forest model, thereby resulting in an underfitting of the true underlying data generating process. This is consistent with the observation of a temporal pattern in the crime classifications as suggested by the high feature importance weighting assigned to *n_days*. In contrast, the XGBoosting model retained all features, including the Geographic features like *PdDistrict_ABC* (to which XGBoosting assigned high feature importance) and was consequently able to achieve better performance in the final Kaggle submission.