San Francisco Crime Classification
CS 434 Final Report

Godfrey Yeung
yeungg@oregonstate.edu

Cameron Bowie
bowiec@oregonstate.edu

Ben Brook
brookb@oregonstate.edu

Abstract

San Francisco has a dataset of 12 years of crimes committed within the city. As a part of a Kaggle machine learning challenge, they have released this dataset with the intent of predicting what crime is committed based on the time and location of the crime. We extracted features from the dataset and tested several multi-class multi-output supervised learning classifiers on the data. Using k-fold cross-validation and the log loss error function, we tuned the various algorithms and found that the AdaBoost classifier performed the best on our validation runs.

1 Introduction

The project we have chosen is a submitted Kaggle challenge, located here: `https://www.kaggle.com/c/sf-crime`. It involves predicting the probability of a crime taking place at a given location (See Figure 1). Each location has several attributes, such as the date, the day of the week, the police district, the address, and the latitude/longitude of the crime. 39 different class labels, each representing a type of crime, are available to choose from. The contest wants a probability for each type of crime predicted for every point in the test set.

Crime prediction allows police officers to focus their efforts in areas where crimes tend to occur and be better prepared for the most commonly occurring crimes. Prediction and analysis of the data can also be used to detect trends in crimes throughout the city although this effort is tangent to our project.

2 Problem Formulation

We are approaching this problem from the perspective of supervised learning. Our dataset contains 878,049 crime incidents each of which is labeled with the crime committed. The data is discussed in more detail in section 4.1.

Since the Kaggle competition specifies that we must return the probability of each crime being committed for a given test input. Thus, our problem is a multioutput-multiclass problem. Kaggle does not supply the labels for the training data. Instead, we have to construct an array of probabilities for each piece of test data. This gets submitted to Kaggle, where it is graded on their private test labels. The log loss scores for every submission are put on a leader board. The people with the top scores get a small sum of money.

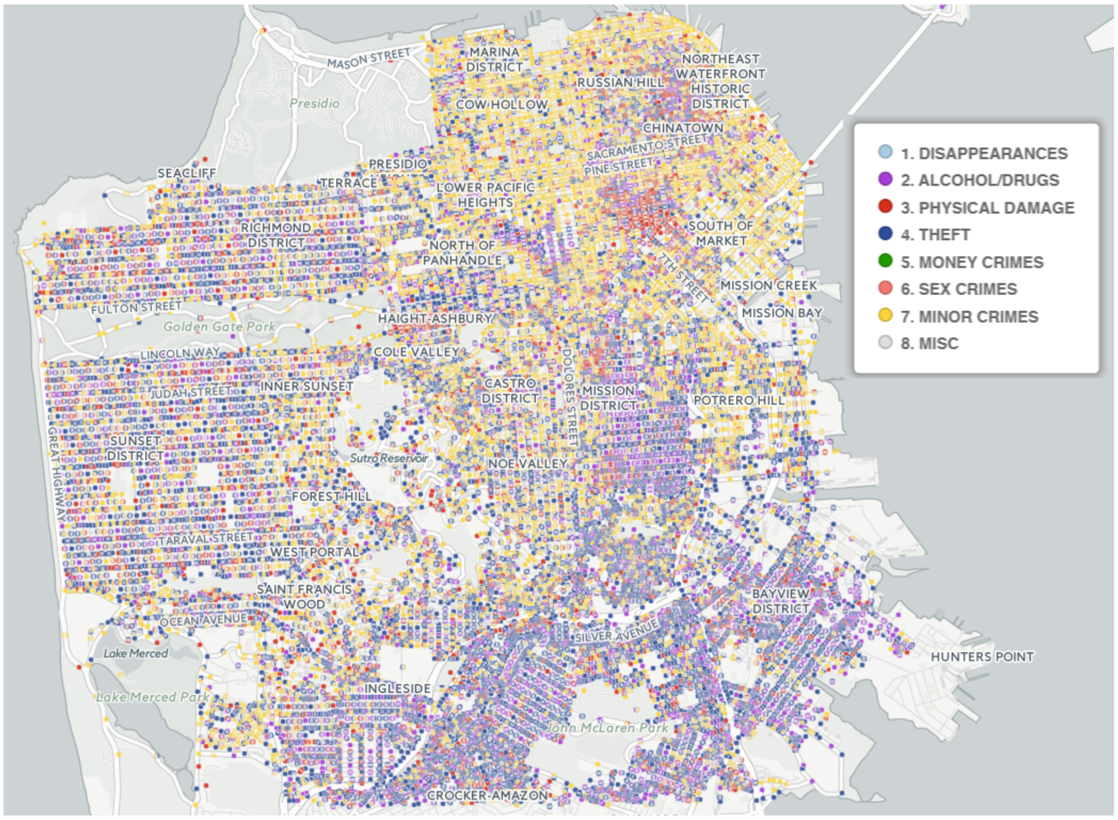

Figure 1: Map of San Francisco crime data. The thirty nine crime labels are grouped into eight categories. One can see that minor crimes dominate downtown San Francisco, alcohol and drug crimes dominate areas south and west of downtown, and that there is a hotspot of sex crimes in the middle of downtown, northwest of South of Market. The categorization of crime labels is as follows:

1. Disappearances: kidnapping, missing person, runaway, family offenses, suicide;
2. Alcohol/drugs: driving under the influence, drunkenness, drug/narcotic, liquor laws;
3. Physical damage: assault, arson, weapon laws, vandalism;
4. Theft: robbery, burglary, larceny/theft, vehicle theft, stolen property, recovered vehicle;
5. Money crimes: bribery, forgery/counterfeiting, bad checks, gambling, embezzlement, fraud, extortion";
6. Sex crimes: prostitution, pornography/obscene mat, sex offenses non forcible, sex offenses forcible;
7. Minor crimes: disorderly conduct, loitering, trespass, non-criminal;
8. Misc: secondary codes, trea, other offenses, warrants, suspicious occ;

Dates	Category	Descript	DayOfWeek	PdDistrict
2015-05-13 23:53:00	WARRANTS	WARRANT ARREST	Wednesday	NORTHERN
2015-05-13 23:53:00	OTHER OFFENSES	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN

Resolution	Address	X	Y
ARREST	OAK ST / LAGUNA ST	-122.42	37.7
ARREST	OAK ST / LAGUNA ST	-122.42	37.7

Table 1: The training data supplied by Kaggle is in the form shown here. We used the dates, X, and Y fields as our features and predicted the category field.

Id	Dates	DayOfWeek	PdDistrict	Address
0	2015-05-10 23:59:00	Sunday	BAYVIEW	2000 Block of THOMAS AV
1	2015-05-10 23:51:00	Sunday	BAYVIEW	3RD ST / REVERE AV

X	Y
-122.39958770418998	37.7350510103906
-122.391522893042	37.7324323864471

Table 2: The test data supplied by Kaggle is in the form shown here. Note that several of the fields given in the training data are absent from the test data. We excluded these fields from our analysis for this reason.

3 Proposed Approach

Our approach to this problem is to test a broad array of classifiers and to rely on k-fold cross-validation for our parameter tuning. We use Python and the scikit-learn module. We decided to try 5 different classifiers: k-NN, Random Forest, Gaussian Naive Bayes, Bernoulli Naive Bayes, and AdaBoost. Several of these have additional parameters that we have to tune to get the best performance. We tune them by first identifying a reasonable range of values using research on the internet and the default values that scikit-learn provides. Then, we run k-fold cross validation for each possible value and record the log loss. The parameter with the lowest log loss is the correct one to use for the final comparison. If there are multiple dependent parameters, then we try every combination.

Because there are almost 1 million data points, we need all of the computation power that we can get. Random forest is set up to use almost 1000 trees. When it runs on `flip`, it uses 24 cores and almost 50 gigabytes of RAM. Time is a factor when running several of these tests.

4 Experiments

4.1 The Data

The data is supplied in the form shown in table 1. From this we chose to extract a select subset of the information provided. First, we decided that the description (Descript), police department district (PdDistrict), resolution, and address were all either redundant information or unusable for classification. Note in figure 2 that the columns Descript and Resolution are absent from the test data (Category is also, but this is because it is the column that we are predicting). We were forced to exclude these columns since they will not be present when we are performing the test classification for Kaggle submission.

We extracted a number of features from the data. These are the year, month, day of the week, and hour (segmented into 3 hour blocks of time) that the crime was committed as well as the latitude and longitude.

Year	Month	Day of Week	Hour	X	Y
2015	5	2	7	78	59
2015	5	2	7	79	82

Table 3: The features from the training data following pre-processing.

We based our choice of time-features based on recommendations from the Kaggle discussion forums.

In order to handle numpy constraint that all of our data is of one type, we discretized the position into X, Y coordinates in a grid. This was done by normalizing the latitude to a range of 0 to 100 and using the same normalization constant to normalize the longitude as well. This ensured the latitude and longitude distances were scaled in a consistent manner.

To perform training, we first pre-process the training data a store the features in the form demonstrated in table 3. The day of week starts with Monday and is a value from 0 to 6.

4.2 Experimental Setup

For this project, we relied heavily on the scikit-learn toolset as well as matplotlib and our own implementation of k-fold cross-validation.

First, we pre-processed the data into the form shown in figure 3 which was stored as a CSV. In this step, we also converted the labels from strings to integers. In doing so, we created and stored to disk a mapping from the new identifiers to the original strings.

Once this was complete, we performed cross-validation to tune hyper-parameters of the various classifiers. For each classifier that we tested, we ran a k-fold cross-validation with $k = 5$ and with 5 different random divisions. We then stored this data to a JSON file for plotting. The plots generated from this process and a detailed discussion of the classifier specific tuning is given in section 4.3.

4.3 Hyper-Parameter Tuning

4.3.1 k-NN

Based on testing, we found k-NN to perform poorly regardless of parameter tuning. This is shown in Figure 2. We suspect this has to do with the features present in the data. Since the data contains both time and location data as well as a number of fields that are difficult to measure distance with (day of the week, for example), distance metrics are not a particularly effective measure of similarity. Thus, k-NN does not provide good results.

4.3.2 Random Forest

Figure 3 shows the result of of tuning the `min_samples_leaf` hyper-parameter in the random forest algorithm. This parameter determines when trees should form leaves and thereby controls tree depth [1]. We used k-fold validation to test values from 1 to 5000. As the figure shows, there are diminishing gains after a size of 1000.

Figure 4 shows tuning of the `max_features_tuning` hyper-parameter, which controls how many features are randomly chosen to be considered at each split. The results are fairly inconclusive. There is almost no change in the log loss between these options, so we decided to leave it at its default value of $\sqrt{n\_features}$.

Online recommendations said to use as many trees as possible. We had access to a lot of memory on flip, so we used 1000 trees [2].

4.3.3 Gaussian and Bernoulli Naive Bayes

There were no parameters to tune for either scikit-learn Naive Bayes implementation. Their log loss error results are discussed in a comparison with the others in section 4.4.

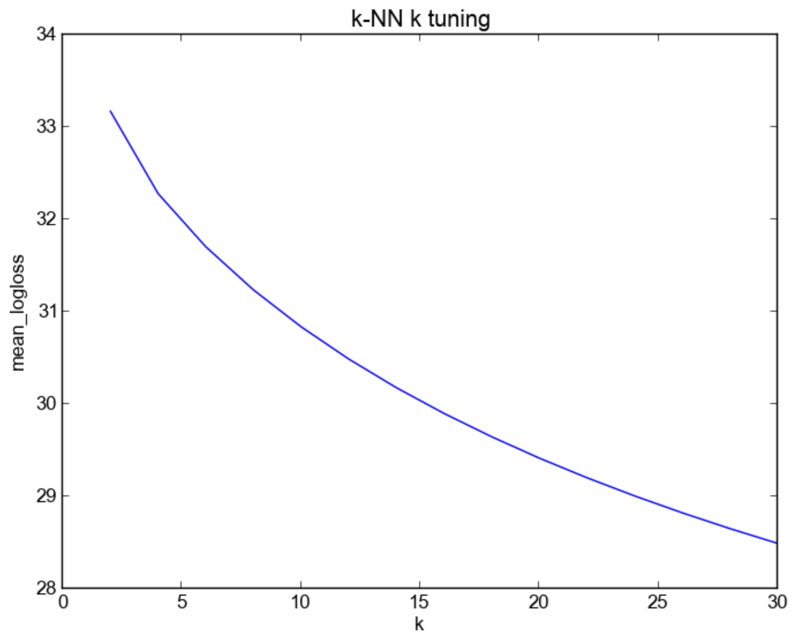

Figure 2: k-NN k hyper-parameter tuning

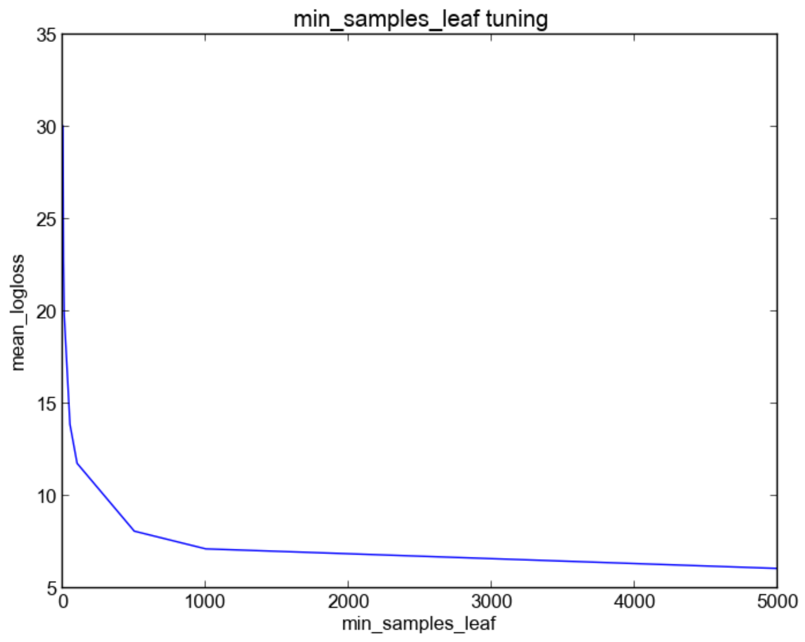

Figure 3: Random Forest min_samples_leaf hyper-parameter tuning

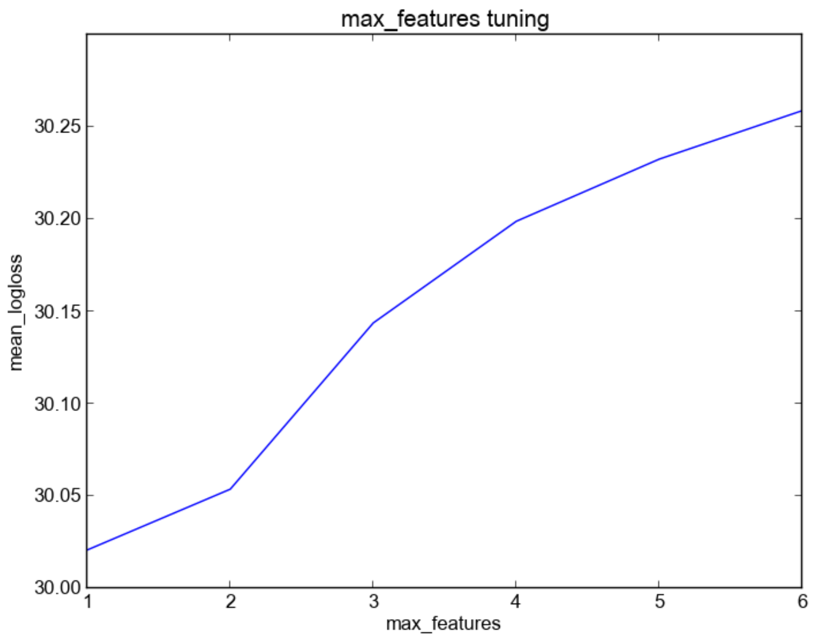

Figure 4: Random Forest max_features hyper-parameter tuning

4.3.4 AdaBoost

Figure 5 shows hyper-parameter tuning for AdaBoost. We tried four different base estimators: Decision Tree stumps, full Decision Trees, Gaussian Naive Bayes, and Bernoulli Naive Bayes. Full Decision Tree produced by far the worst results. This appears to be because the full decision tree reduces the possible variance between the different AdaBoost iterations. This makes it far less effective at reducing the variance of the predictions. The stump, Gaussian Naive Bayes, and Bernoulli Naive Bayes all performed roughly the same, with Bernoulli Naive Bayes beating the others out by a fraction of a point.

AdaBoost's other parameter that we tuned was the number of estimators. This is the maximum number of estimators that are used before the boosting is terminated. The results are in Figure 6.

4.4 Final Results

After running every algorithm with the tuned parameters from Section 4.3, we plotted the results in Figure 7. k-NN was by far the worst algorithm for this task, getting a log loss in the high 20s. All of the others were much better, with AdaBoost performing the best overall with a log loss of 3.57. Tuning our hyper-parameters led to strong gains in the accuracy of all of the algorithms tested.

4.5 Kaggle Results

Using AdaBoost, we performed predictions on the test data and submitted it to Kaggle. We received a score of 3.63 and a ranking of 1706 out of 2313 teams.

5 Conclusion

In this report we tested the performance of 5 different classification algorithms on long-term crime data from the City of San Francisco. All of the algorithms that had additional hyper-parameters were tuned before the final comparison. AdaBoost was the best choice of algorithm, as defined by the log loss metric. AdaBoost used the Bernoulli Bayes algorithm as its base, which we already found did

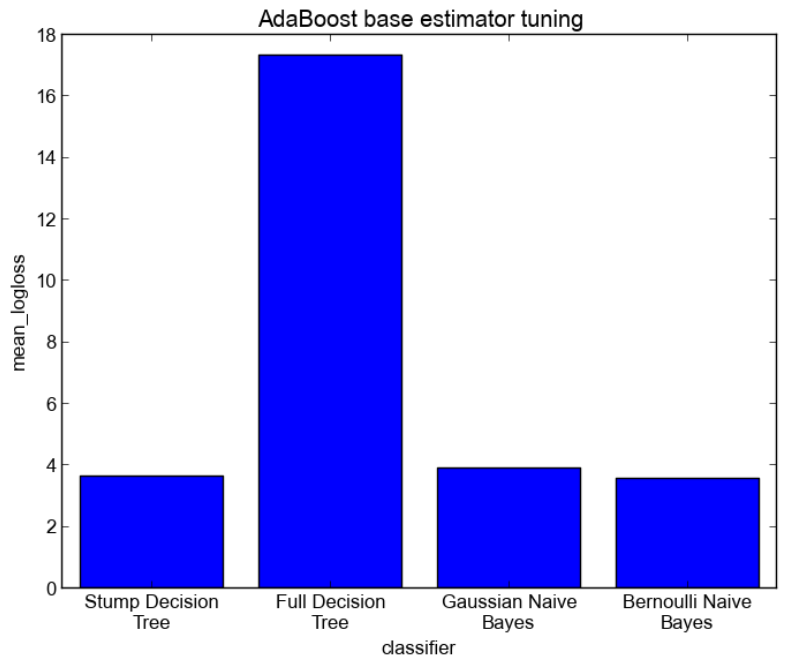

Figure 5: AdaBoost base estimator hyper-parameter tuning

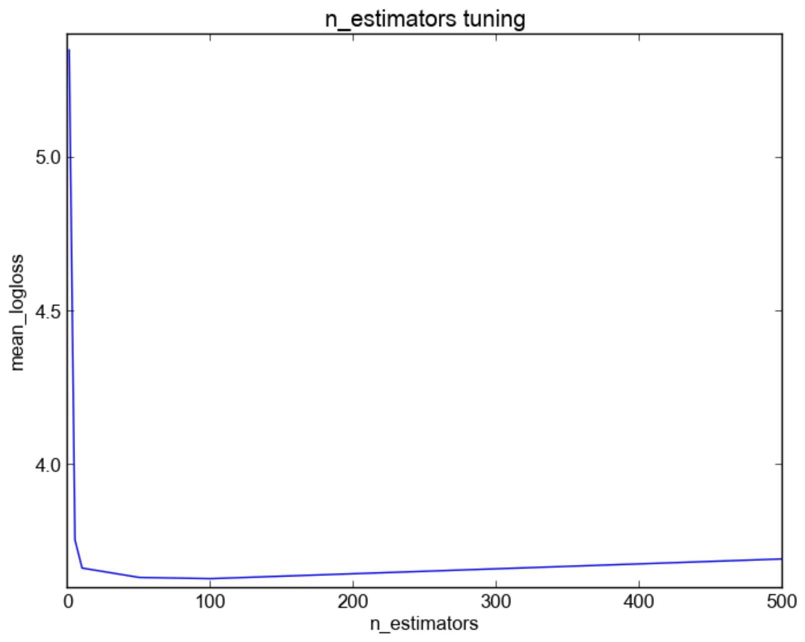

Figure 6: AdaBoost n_estimator hyper-parameter tuning

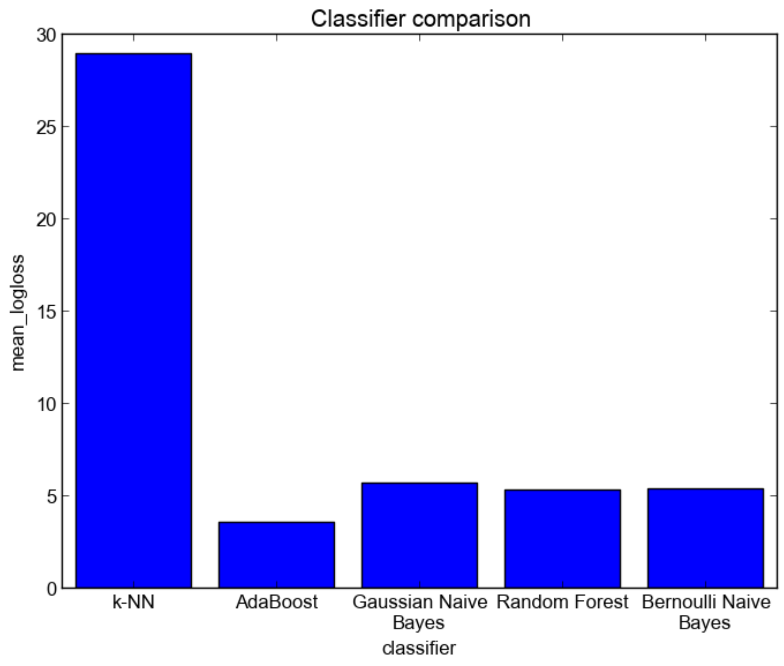

Figure 7: Log loss from 5 algorithms tested

a good job of classifying. AdaBoost seeks to overcome the limitations of one classifier and makes harder classification problems much easier [3]. It makes sense that it would be the best algorithm for this task. If we had more time, we would try Support Vector Machines as well. Our exploratory tests took too long to finish, so we were not able to see how it would compare.

References

[1] (2013) How to tune rf parameters in practice? [Online]. Available: https://www.kaggle.com/forums/f/15/kaggle-forum/t/4092/how-to-tune-rf-parameters-in-practice

[2] T. Srivastava. (2016) Tuning the parameters of your random forest model. [Online]. Available: http://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/

[3] R. E. Schapire, “Explaining adaboost.” [Online]. Available: http://rob.schapire.net/papers/explaining-adaboost.pdf