San Francisco Crime Classification

Junbo Ke
Computer Science and Engineering
University of California,San Diego
juke@eng.ucsd.edu

Xinyue Li
Computer Science and Engineering
University of California,San Diego
xil431@eng.ucsd.edu

Jiajia Chen
Computer Science and Engineering
University of California,San Diego
jic289@eng.ucsd.edu

ABSTRACT

The task of crime prevention is constrained by police resources. For the safety of a city, if the police is aware of what kinds of crimes are mainly going on, and what their distribution is over the city, they can identify where to target police resources and help alleviate crimes. In this paper, given time and location, we predicted the category of crimes that occurred from 2003 to 2015 in San Francisco's neighborhoods based on a dataset derived from SFPD Crime Incident Reporting System. We investigated classification models including Naïve Bayes, k-NN, Gradient Tree Boosting and analyze their pros and cons on this prediction task. Since this is a competition currently held by Kaggle, we also submitted our results to Kaggle and compared them with the public leaderboard, ranking nearly top 10% on our best performance in the end.

Keywords

Crime Classification; Naïve Bayes; k-NN; Gradient Tree Boosting

1. INTRODUCTION

Our goal is to predict crime category of the past crime recordings. In Section 2 we introduce the dataset we used in the experiment. Then we talk about details of our predictive task in Section 3 and what features we extracted from the original dataset in Section 4. Some related work on crime classification is presented in Section 5. Section 6 discusses the models we used for the task and analyze their pros and cons. We employed several models like Naïve Bayes, k-NN, and Gradient Tree Boosting to predict the category of the crimes, and the performance of the models shows significant differences. The models are evaluated using multi-class logarithmic loss. We also submitted our result to Kaggle to see our models' performance.

2. DATASET

We used a dataset from Kaggle [1] to build the classifier. This dataset contains incidents derived from SFPD Crime Incident Reporting system. The data range from 1/1/2003 to 5/13/2015. The training set and test set rotate every week, meaning week 1,3,5,7... belong to the test set, and week 2,4,6,8... belong to the training set. For each row of data, there are 9 columns:
Dates: timestamp of the crime incident
Category: category of the crime incident *(only in train.csv).* This is the target variable we 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

Some interesting findings emerge from over 800,000 records of crimes.

In this dataset, there are 39 types of crimes in total, among which the top 5 frequent are theft, other offenses, non-criminal, assault, and drug/narcotic. We concluded from Figure 1 that the distribution of crimes satisfies the long-tailed property, meaning that several most common crime categories make up the majority of all crimes. Statistically, the top 5 crimes account for 66% of the whole records. So it is reasonable to suggest allocating more police resources to dealing with these crimes as they are more likely to occur.

As part of our exploratory analysis, we also looked at how the occurrences of crimes vary from different police department districts. From Figure 2 we could figure out

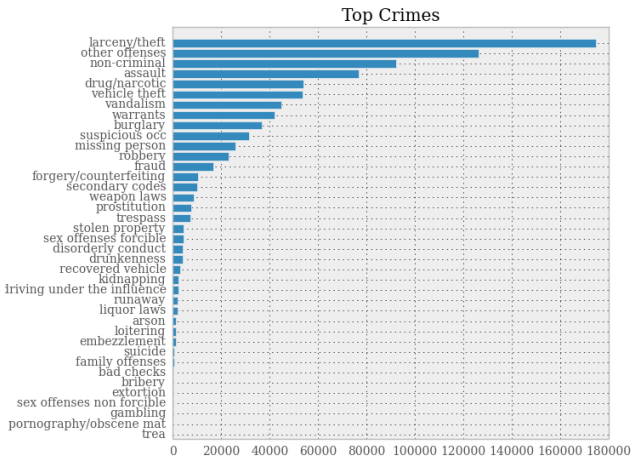

Figure 1: Crime Distribution

that Southern Police Department District handles the most crimes, almost 4 times more than Richmond Police Department District.

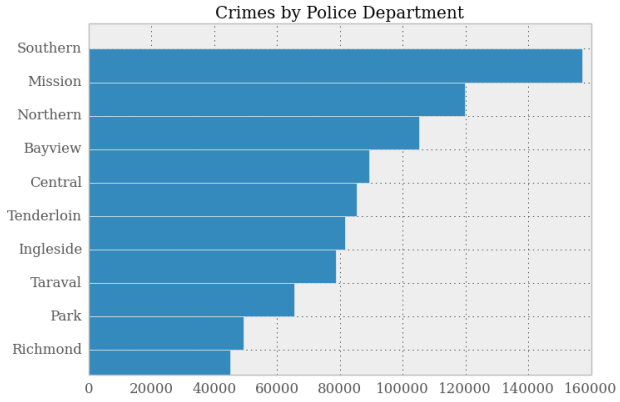

Figure 2: Crimes by Police Department

Moreover, to explore the dataset in a geometrical manner, we drew the scatter plot on the map of San Francisco. In Figure 3 we labeled different crimes with different colors. It could be inferred from the map that crimes concentrate most on the northeastern part of the city, which is the downtown area of San Francisco. This observation is consistent with our expectation.

Meanwhile, we were interested in whether there is a correlation between crimes and days of a week. As shown in Figure 4 (only top 15 crimes are illustrated for clearity), Friday witnesses the most criminal incidents and the least take place on Sunday, though the difference between them is not significant. We noticed that the proportion of each crime does not change markedly with days of a week.

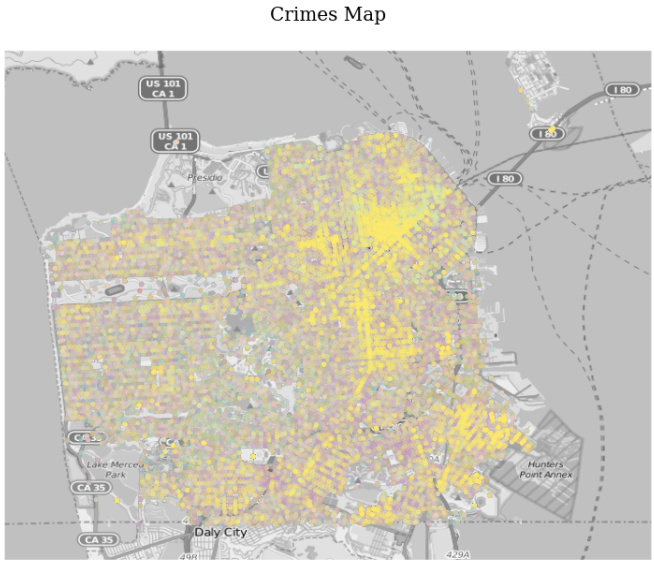

Figure 3: Crimes Over The City

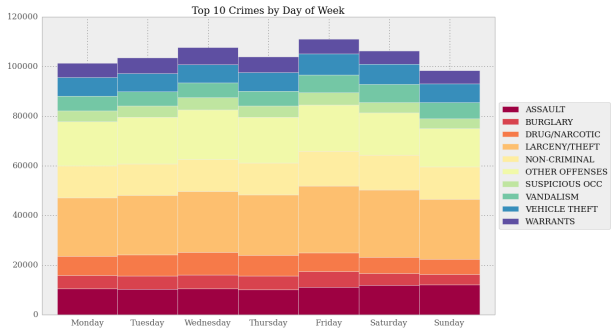

Figure 4: Crimes Stacked By Day

Hours of a day tell a better story about when crimes happen most probably. According to Figure 5, crimes rarely break out between 3:00 and 6:00, but reach to their second peak at 12:00, and first peak around 17:00 to 18:00. So when police resources are limited, we would advise assigning more policing from noon to midnight.

3. PREDICTIVE TASK

Our task is to predict the category of the crimes. We will evaluate the performance of our models on Kaggle [2], which measures the prediction error by multi-class logarithmic loss. Each case is labeled with one true category. For each record, we calculated a set of predicted probabilities (one for every class). The formula is then,

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

where N is the number of incidents in the test set, M is

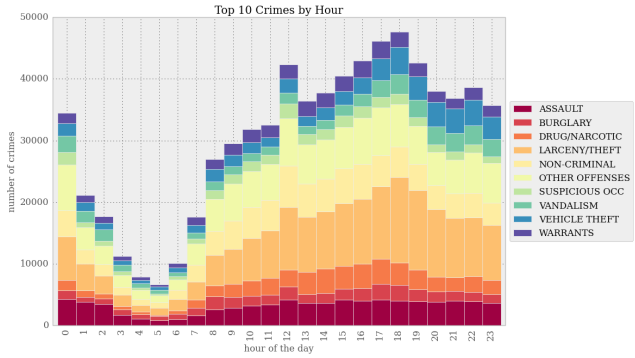

Figure 5: Crime Stacked By Hour

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.

We randomly split the whole dataset of known labels into 95% for training and 5% for validation, and assessed the validity of our models' predictions by calculating the log loss on the validation set for each model. Also, we submitted our results to Kaggle's rating system to justify our solutions.

As a baseline, we predicted all crimes as the most common type of crime, in this case, theft. So we set the 'theft' column to 1 and left all the rest columns as 0 for every record. The baseline got a score of 32.89184 on the leaderboard.

4. FEATURE SELECTION

In Section 2, we have seen that each record has 9 columns. Generally, we will adopt proper representations of **Dates**, **DayOfWeek**, **PdDistrict**, **X**, **Y** and **Address** as our feature set:

Dates: We extracted **year**, **month**, and **hour** from this field and used their trivial representation as our features. Day wasn't included, for it could be reflected by day of week.

DayOfWeek: At first we represented this column with numerical value 0 - 6, but such way implied days as linearly related. So later, we switched to utilize 7-d binary vectors. For example, [1 0 0 0 0 0 0] represents Monday and [0 1 0 0 0 0 0] represents Tuesday, etc.

PdDistrict: Just like how we dealt with **DayOfWeek**, we initially represented the 10 police department districts with numerical values and then transformed to 10-d vectors.

Address: Inspired by Microsoft Azure[3], we used a 39-d vector to represent each address. Each vector represents the frequency distribution over 39 crime categories at the corresponding address. Specially, we used Laplace Smoothing to avoid zero frequencies. Each element (dimension) in a 39-d vector, the plain frequency value p, is replaced by its *logodds* value, which is defined as:

$$log\frac{p}{1-p}$$

Throughout the whole process, we experimented with 3 different combinations of feature sets. They shared common features **year**, **month**, **hour**, **X**, **Y**, and were different in:

#1 numerical representation of **DayOfWeek**; numerical representation of **PdDistrict**. (**Address** field is left out to construct a baseline.)

#2 vectorized representation of **DayOfWeek**; vectorized representation of **PdDistrict**.

#3 vectorized representation of **DayOfWeek**; vectorized representation of **PdDistrict**; frequency distribution (used in Naïve Bayes) or *logodds* frequencies (used in all other models) of **Address** over all crime categories.

They will be referred to as feature set #1, #2 and #3 hereon.

5. RELATED WORK

We used an existing dataset of incidents derived from SFPD Crime Incident Reporting system. It comes from SF OpenData, the central clearinghouse for data published by the City and County of San Francisco. SF OpenData also provides datasets of public services in other categories, ranging from City Infrastructure, City Management and Ethics, Culture and Recreation, to Economy and Community, Energy and Environment, and Public Safety.

There is not much research on crime data mining yet. [4] studies various crime data mining techniques, entity extraction, association, prediction, pattern visualization included, and matches them with most suitable crime types for analysis, such as traffic violations, sex crime, theft, fraud, arson, gang/drug offenses, violent crime, and cybercrime. [5] investigates another crime dataset acquired from the UCI machine learning repository website [6] from a different prospective. The paper basically categorizes all U.S. states into areas of low, medium, and high crime rates based on twelve social features, mainly population density, unemployment rate, median household income, and population under the poverty threshold. Two algorithms, namely Naïve Bayes and Decision Trees, are applied to the task and evaluated by accuracy, precision, recall and F-Measure. Decision Trees turns out to out perform Naïve Bayes with a 83.9519% accuracy.

Despite the lack of related literature, this competition on Kaggle arouses a very heated discussion. Users [7] in the top tier of the leaderboard mention several useful and powerful python libraries that they have applied apart from scikit-learn, including Keras [8], Lasagne [9], and XGBoost [10]. These libraries make full use of CPUs, GPUs, parallel computing and distributed computing, thus speed up the training step dramatically. On top of that, there are also novel ideas regarding feature engineering. For instance, NicolasCte [7] maps (**X**, **Y**) to a 400×400 grid of San Fransisco's geographical map; papadopc [11] introduces a seasonal feature (spring, summer, fall, winter), and leverages PCA to reduce the dimension of the feature vector.

Another critical technique to increase accuracy mentioned on Kaggle is model ensembling [12]. Ensemble learning uses multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms [13, 14, 15]. The less correlated the individual classification algorithms are, the better results an averaging ensemble yields.

6. MODELS

In this prediction task, we employed several models to complete the classification task and compared the performance of them.

6.1 Naïve Bayes Classifier

Naïve Bayes Classification algorithm is commonly used in text classification tasks. The training and test features used for Naïve Bayes are usually represented as counting vectors or TF-IDF vectors, in which numeric values are always nonnegative. In order to apply Naïve Bayes model into our prediction task, we transform the **X** and **Y** values to the corresponding absolute values. However, the Naïve Bayes model assumes that features are conditionally independent, which may be wrong with our feature vector because the field **PdDistrict** is related to the coordinate (**X**,**Y**) to some extent. Although suffering from the double counting problem, this model results in a relatively low *logloss* value. On the other hand, the features we deploy do not perfectly meet the requirement of Naïve Bayes. For example, we usually use count or frequency vector as feature vector of Naïve Bayes, but the **X** and **Y** values are not appropriate to represent either count or frequency.

6.2 Multiclass Support Vector Machines Classifier

Support Vector Machines (SVMs) are non-probabilistic supervised learning models used for classification. It aims at optimizing the misclassification error rather than the likelihood. While SVM is traditionally a binary classifier, it can be simply generalized to multiclass classification by a one-vs-one scheme, that is, training a binary predictor for each class. However, the fit time complexity becomes more than quadratic with the number of samples. As the official document of the scikit-learn [16] library claims SVMs hard to scale to datasets with more than a couple of 10,000 samples, our training dataset of around 800,000 samples is doomed to fail because of the extremely long fit time.

6.3 *k*-Nearest Neighbors Classifier

k-Nearest Neighbors algorithm (or k-NN for short) is a rather simple approach among all machine learning algorithms. It can be used for both classification or regression tasks. In *k-NN classification*, the basic idea is quite clear and obvious: any unlabeled object is assigned to the most common class of its k nearest neighbors (k is a positive, typically small integer). The neighbors are labeled, in other words, to which classes that they belong are known in advance. This can be treated as the training set of the algorithm. Problem arises when the training data is not in uniform distribution, e.g. they are dominated by a certain class. Then new objects tend to be classified into that class due to the large number of its neighbors in that class. This is a significant shortcoming of *k-NN classification*.

In order to find k objects closed to a query point, we need to define a way to calculate the distance between each pair of objects. The most commonly used distance metric for continuous feature variables is Euclidean distance. However, most features of our feature vector are discrete variables except **X**, **Y** and **Address**, in which case Hamming distance can be applied as an alternative. The scikit-learn library uses minkowski as the default distance metric. In addition, whether to assign weights to the contributions of the neighbors also needs taking into consideration. The weights can be uniform, where all points in each neighborhood are weighted equally; or by the inverse of their distance, where closer neighbors of a query point will have a greater influence than neighbors which are further away. Last and most importantly, different choices of k usually has a great impact on the accuracy of the prediction. In general, larger values of k reduces the overall noise on the classification, but makes decision boundaries between classes less precise, and model fitting computationally expensive. So it is a trade-off problem. [17] suggests that a good starting point is to set k equal to the square root of the number of instances.

6.4 Gradient Tree Boosting

Gradient Tree Boosting is a more generalized ensem-

ble model in classification tasks, which builds the model in a forward stage-wise fashion. The model trains a series of decision trees, where the successive tree is trained based on the previous one to minimize the prediction error rate (defined by a loss function). One of the advantages of Gradient Boosting Tree is that the scale of the feature values does not matter, which makes it unnecessary to normalize features.

We tried to apply the library implementation of Gradient Tree Boosting from scikit-learn. In this implementation, the loss function is defined as the negative multinomial log-likelihood loss function. The number of decision trees is controlled by the model parameter *n_estimators*. In order to avoid overfitting, we did a grid search over a range of parameter values by calculating the *logloss* value on the validation set. Although the documentation says a large number of decision trees usually leads to better results, we find it still suffer from over-fitting on the dataset we use.

A benefit for employing Gradient Tree Boosting model is that the output score of each category can be easily transformed into probabilities. Compared to the previous models, the complexity of Gradient Tree Boosting increases significantly. It took a few hours to train the model over 800,000 records on our laptop.

7. EVALUATION AND RESULTS

All the models described in the previous section were trained and tested in our prediction task. The following subsections talk about the results.

7.1 Naïve Bayes

This was the first model we tried because of its simplicity and efficiency. Table 1 demonstartes our results evaluated by log loss metric.

Table 1: log loss on validation and test set

feature set #	validation	test
1	2.70299	2.71330
2	2.61473	2.63458
3	2.57262	2.59560

A better way to visualize the prediction result is drawing the confusion matrix on the validation set. Here we predict the crime category to be the most probable one calculated by Naïve Bayes model. From Figure 6 we find that the model predicts most of the crime records as #16 ('LARCENY/THEFT') and #21 ('OTHER OFFENSES'). Considering the distribution shown in figure 1, it is a reasonable result because these two categories are the two most frequent crimes.

7.2 *k*-Nearest Neighbors Classifier

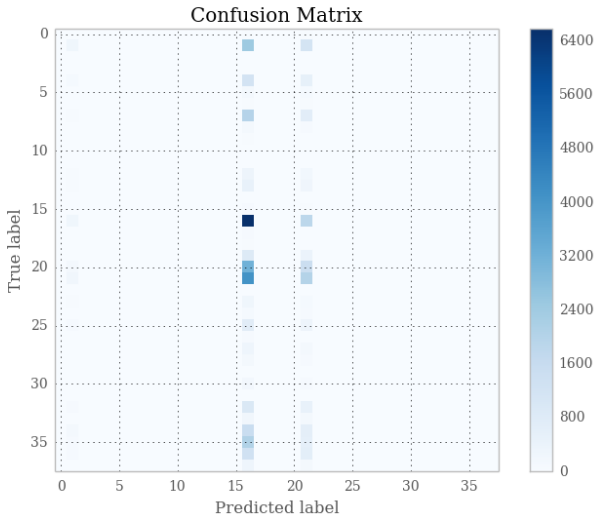

Figure 6: confusion matrix

We ran *k-NN classifier* on feature set #1 and #3 with different values of k and a uniform weight.

From Figure 7, it is obvious that feature set #3 constantly performs better than feature set #1 on same values of k, which implies that introducing new features of **Address** and replacing a single feature vector for **DayOfWeek** and **PdDistrict** by multi-dimensional vectors do make a great improvement.

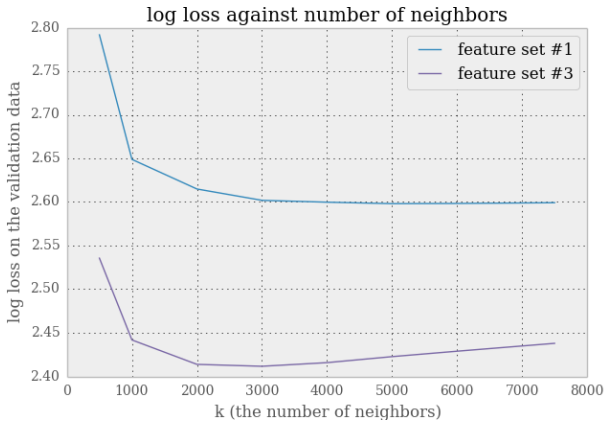

Figure 7: log loss on validation set versus k in k-NN classifier

Based on the logarithmic losses for each round shown in Table 2, we can tell that the optimal value of k is 5000 for feature set #1, and 3000 for feature set #3. It justifies our expectations that:

(1) The *empirical rule-of-thumb* to select k as the square root of the number of instances works:

Table 2: logloss on validation set versus k

k	feature set #1	feature set #3
100	3.86541897948	3.53869884859
500	2.79168755317	2.5355983395
1000	2.64885877334	2.441789077
2000	2.61490537578	2.41380888142
3000	2.60191517479	**2.41161174401**
4000	2.59979359758	2.41579106533
5000	**2.59807194667**	2.42266528866
6000	2.59831262337	2.42888834845
7500	2.59918377092	2.43791392131

Here, $\sqrt{\text{N samples}} \approx \sqrt{800,000} \approx 900$. From Table 2, the log loss is significantly large when k is rather small, but it doesn't drop noticeably after $k > 1000$.

(2) A larger k doesn't guarantee smaller errors:

The larger k is, the more smoothing takes place, and eventually it will end up under-fitting (the output is constant regardless of the attribute values).

Another interesting finding is that the classifier produces far worse performance when we attempted to assign inverse distance weighting to the neighbors, e.g. the log loss with $k = 5000$ on feature set #1 is 9.58989, compared to 2.59807 without weighting. We guess the reason behind this phenomenon is that the concept of *distance* in our data set doesn't have much real contextual meaning.

7.3 Gradient Tree Boosting

Finally we applied Gradient Tree Boosting to complete the prediction task. We directly trained and evaluated this model with feature set #3 and got the results in Figure 8.

Generally, it is always better to employ more decision trees for higher prediction accuracy (lower log loss score). However, in our experiment, the best result occurs when we set *n_estimators* to 50, and it does not turn better as the number of trees increases.

We measured the importance of the features we adopted with 'gini importance' [18] and got the 15 most important features as shown in Figure 9.

From Figure 9, **hour** is the most important feature as expected. Besides, the logodds value of frequencies are generally among the top informative features, which justifies our selection of features. On the other hand, **DayOfWeek** features are among the least important ones, which is consistent with our analysis on the dataset.

8. CONCLUSION

Figure 8: log loss on validation and test set versus *n_estimators*

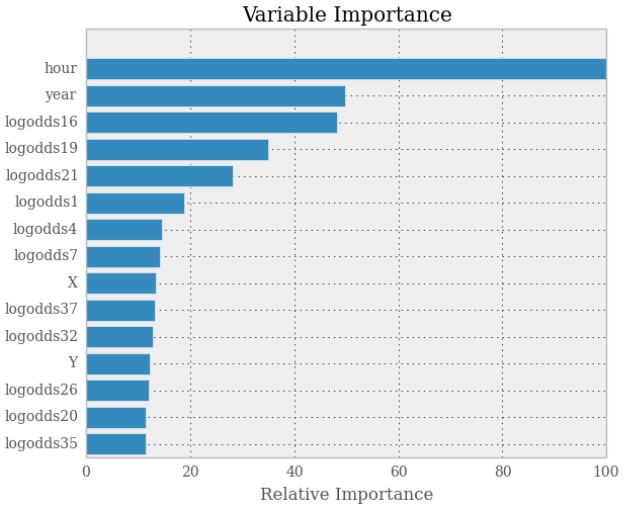

Figure 9: Importance of features

For this prediction task, we started from preprocessing the data set from SFPD Crime Incident Reporting system. Then, we attempted to select some helpful features to represent the attributes of the samples in a proper manner. Adopting *logodds* **Address** turned out to improve the performance of our models by a lot. Finally, by training some models with the features we employed and calculating the probabilities of different categories of crimes, we completed the whole task. As presented in the results, Naïve Bayes is not a perfect model for this task because some of the features do not represent the count or frequency. Once the number of neighbors is properly chosen, k-Nearest Neighbors improves the prediction result significantly. Gradient Tree Boosting turned out to be the best model in our experi-

ment, but it is relatively time consuming. After submitting the best result generated by Gradient Tree Boosting model, we scored 2.39383 and ranked 93 among 878 teams. For improvement of our model, we may extract more features from **Address** and temporal columns.

9. REFERENCES

[1] San Francisco Crime Dataset(2015). Available from: https://www.kaggle.com/c/sf-crime/data

[2] San Francisco Crime Classification Evaluation(2015). Available from: https://www.kaggle.com/c/sf-crime/details/evaluation

[3] Data Transformation / Learning with Counts(2015). Available from: https://msdn.microsoft.com/en-us/library/azure/dn913056.aspx

[4] Chen, Hsinchun, et al. "Crime data mining: a general framework and some examples." *Computer* 37.4 (2004): 50-56.

[5] Iqbal, Rizwan, et al. "An experimental study of classification algorithms for crime prediction." *Indian Journal of Science and Technology* 6.3 (2013): 4219-4225.

[6] UCI Machine Learning Repository (2012). Available from: http://archive.ics.uci.edu/ml/datasets.html

[7] Competition forum entry on Kaggle: Any suggestions to improve the performance? (2015). Available from: https://www.kaggle.com/c/sf-crime/forums/t/16348/any-suggestions-to-improve-the-performance

[8] Keras: Theano-based Deep Learning library (2015). Available from: http://keras.io

[9] Lasagne: a lightweight library to build and train neural networks in Theano (2015). Available from: https://github.com/Lasagne/Lasagne

[10] XGBoost (eXtreme Gradient Boosting): An optimized general purpose gradient boosting library (2015). Available from: https://github.com/dmlc/xgboost

[11] Script on Kaggle: neural nets and address featurization (2015). Available from: https://www.kaggle.com/papadopc/sf-crime/neural-nets-and-address-featurization

[12] Kaggle Ensembling Guide(2015). Available from: http://mlwave.com/kaggle-ensembling-guide/

[13] Opitz, David, and Richard Maclin. "Popular ensemble methods: An empirical study." *Journal of Artificial Intelligence Research* (1999): 169-198.

[14] Polikar, Robi. "Ensemble based systems in decision making." *Circuits and Systems Magazine, IEEE* 6.3 (2006): 21-45.

[15] Rokach, Lior. "Ensemble-based classifiers." *Artificial Intelligence Review* 33.1-2 (2010): 1-39.

[16] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *The Journal of Machine Learning Research* 12 (2011): 2825-2830.

[17] Duda, Richard O., Peter E. Hart, and David G. Stork. *Pattern classification*. John Wiley & Sons, 2012.

[18] Breiman, Leo, et al. *Classification and regression trees*. CRC press, 1984.