

# Class Project: San Francisco Crime Classification

Student Name(s): Jordan Gaeta and Geri Rista

December 12, 2015

## 1 Intro:

In this project, various machine learning methods were utilized in an attempt to predict the category of crime in the city of San Francisco. Due to the categorical nature of the data, it was necessary for the data to be preprocessed into numerical values. Then, the following classifiers were ran on the data: Naive Bayes, Random Forest, Neural Networks, and Logistic Regression. The Neural Network classifier was then explored further due to initial promising results.

## 2 Data Set

### 2.1 Data Set Overview

This data set contains incidents taken from the San Francisco police department Crime Incident Reporting System. The data set was collected from January 1st, 2003 until May 13th, 2015. The data available consists of the date the crime occurred, the description of the crime incident, the day of the week, name of the police district, how the crime was resolved, address, and longitude and latitude, with the category of the crime being the target variable.

A prediction in this competition consists of predicting the probability for each class, and then predicting which class of crime was committed based on certain features. This data was evaluated using the multi-class logarithmic loss metric which is given by the formula,

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

where  $N$  is the number of observations,  $M$  is the number of class labels,  $\log$  is the natural logarithm,  $y_{i,j}$  is 1 if observation  $i$  is in class  $j$  and 0 if it is not, and  $p_{i,j}$  is the predicted probability that the observation  $i$  is in class  $j$ . Since most of this data was categorical in nature, data preprocessing techniques were used to convert it into a form usable by the algorithms.

### 2.2 Data Preprocessing

Since this competition provided us with data that consisted of categorical features, we were required to transform this data into numerical features so that we could learn and predict using the data. In order to achieve this, we decided use the pandas library. Pandas has a method for transforming categorical features into dummy numerical data so that you may run machine learning algorithms to learn and predict on the data. The method is named 'pd.get\_dummies'.

There were two categorical features that we had to convert using this method. They were the day of the week and the police district. The idea is to create a column for each day of the week and each police district, and then assign either a 0 or a 1 depending on whether or not a specific sample fell into a specific police district or day of the week. This is done with the piece of code at the top of the next page.

```
dummy_days = pd.get_dummies(dataset.DayOfWeek)
dummy_dis = pd.get_dummies(dataset.PdDistrict)

traindata = pd.concat([dummy_days, dummy_dis], axis=1)
```

We also were tasked with parsing the dates of each crime. Pandas has a feature for this where simply indicate which column the dates lie in. This is done as follows:

```
file_path = '../data/train.csv'

dataset = pd.read_csv(file_path, parse_dates=['Dates'])
```

Then it will parse it into month, hour, date and year. From this, we were able to make four separate features: month, hour, date and year as follows:

```
hour = dataset.Dates.dt.hour
year = dataset.Dates.dt.year
month = dataset.Dates.dt.month

traindata['Month'] = month
traindata['Hour'] = hour
traindata['Year'] = year
```

After exploring the data, we noticed that there were certain samples with longitude and latitude coordinates that did not make any sense. They suggested that the crimes occurred near the North Pole. We dealt with this by removing these samples from the training set since there was a small number of them as follows:

```
dataset = dataset[abs(dataset['Y']) < 50]
```

However, since we had to make predictions for each sample in the test set, we had to come up with a way to solve this issue specifically for the test set. We decided to set the longitude and latitude for these specific cases to the mean of each respective feature. This is done as follows:

```
mean_x = dataset.X.mean()
mean_y = dataset.Y.mean()

testdata['X'] = dataset.X.map(lambda x : mean_x if abs(x) < 121 else x)
testdata['Y'] = dataset.Y.map(lambda x : mean_y if abs(x) > 40 else x)
```

Although this wasn't ideal, it provided good performance in practice.

## 2.3 Feature Engineering

In order to be competitive, we decided that it would be necessary do some feature engineering. We started by brainstorming what kinds of things could lead to a different category of crime occurring. The first one that became apparent was whether it was night or day. We were able to create this feature by checking whether a certain crime occurred in a specific time interval. If the crime occurred between 11 p.m. and 5 a.m., we classified the crime as having occurred at night. This meant assigning a 1 to this feature for a specific sample if it met this condition and a 0 if it did not. This was done with the following piece of code:

```
testdata['Night'] = dataset.Dates.dt.hour.map(lambda x : 1 if (x > 22 or x < 6) else 0)
```

After looking at plots on the kaggle forum associated with this competition, there was much discussion between other users about different types of crimes occurring at intersections rather than

elsewhere. We decided that we would use this as a feature as well. Upon examining the data, it is clear that intersections are represented by placing a '/' between two streets. We simply checked the address for this character, and if it contained this character we assigned the sample a 1 for this feature and a 0 otherwise. This was done as follows:

```
testdata['Intersection'] = dataset.Address.map(lambda x: 1 if '/' in x else 0)
```

The next feature that we created was the 'Season' feature. In our past experience, we have heard that crimes tend to spike during different times of the year so we decided to create a feature that would categorize each crime occurrence into either Fall, Winter, Spring or Summer. If the crime occurred in either December, January or February, we mapped the value to a 1 (winter). If the crime occurred in either March, April or May, we mapped the value to a 2 (spring). If the crime occurred in either June, July or August, we mapped the value to a 3 (summer). Finally, if the crime occurred in either September, October or November, we mapped the crime to a 4 (fall). This was done with the following code:

```
def map_season(x):
    if (x > 11 or x < 3):
        x = 1
    elif (x > 2 and x < 6):
        x = 2
    elif (x > 5 and x < 9):
        x = 3
    elif (x > 8 and x < 12):
        x = 4
    return x

testdata['Season'] = dataset.Dates.dt.month.map(lambda x: utils.map_season(x))
```

The next feature that we decided to create was a 'Street Number' feature. Since creating usable data from the address was not feasible due to the large number of unique addresses contained in the data, we decided to use the number of street as a feature. This was extracted from the Address by extracting the number if it existed. If it did not exist, we simply mapped the street number to a zero for the respective sample which was done with the following snippet of code.

```
testdata['StreetNo'] = dataset['Address'].map(lambda x: x.split('_', 1)[0]
                                              if x.split('_', 1)[0].isdigit() else 0)
```

## 3 Models Utilized

### 3.1 Random Forests

A Random Forest is a classifier that consists of an ensemble of decision trees. The decision trees are fit to various sub-samples of the data and then averaging is used to improve the accuracy of the prediction and control over-fitting. Random Forests was one of the first models tested due to being relatively simple to tune. The scikit-learn implementation of Random Forests was utilized, initially starting off with 10 estimators, the amount of estimators was steadily increased up to 512 estimators, each time utilizing the features of police district, day of the week, and whether or not a crime occurred on a street corner. While this gave a promising initial result, further optimizations became difficult to make as adding more trees did not seem to improve accuracy. We also ran into memory issues, and we did not have the hardware to increase the amount of trees any further. A different model was chosen due to these issues.

### 3.2 Naive Bayes

The Naive Bayes classifier is based on applying Bayes' theorem, which is,

$$\mathcal{P}(y|x_1, \dots, x_n) = \frac{\mathcal{P}(y)\mathcal{P}(x_1, \dots, x_n|y)}{\mathcal{P}(x_1, \dots, x_n)}.$$

Since  $\mathcal{P}(y|x_1, \dots, x_n)$  is constant then the following classification rule applies,

$$\hat{y} = \underset{y}{\operatorname{argmax}} \mathcal{P}(y) \prod_{i=1}^n \mathcal{P}(x_i|y).$$

where  $\mathcal{P}(x_i|y)$  follows from the independence assumption. A MAP estimation is then used to compute  $\mathcal{P}(y)$  and  $\mathcal{P}(x_i|y)$  which is then used to compute the Log Loss. The scikit-learn implementation of the Naive Bayes classifier was used primarily due to the speed of the classifier. The features of police department and day of the week were utilized and these features were assumed to be Bernoulli distributed. We tried adding more features, but this only marginally improved our results. Since there were not any tunable parameters, we decided to move onto other classifiers.

### 3.3 Logistic Regression

The logistic regression classifier is similar to most classification approaches in that it allows us to model the posterior probability that the class is positive and learn the parameters from the data. This relationship is given by the equation,

$$P(Y = 1|x, w) = \frac{1}{1 + e^{-w^T x}}$$

where  $x$  consists of our data points and  $w$  as the set of coefficients from the data. If  $P(Y = 1|x, w) > 0.5$  a class label of 1 is assigned and if  $P(Y = 1|x, w) < 0.5$  a class label of 0 is assigned. Due to the presence of multiple classes, a one versus rest classification strategy is used. The probability of each class is calculated using the logistic function, and then these values are normalized across all classes.

### 3.4 Neural Networks

Eventually it was decided that Neural Networks would be the most advantageous approach for this multi-class classification problem. We decided to use a Multi-Layer Perceptron since it was relatively simple to understand, and we felt that a recurrent network would not be necessary for this particular problem. A Multi-Layer Perceptron is an example of a feed-forward Neural Network. By feeding a set of inputs into an input layer, outputs can then be calculated and fed into another layer. Each layer uses its own transfer function such as the Sigmoid function, Tanh, or Rectified Linear Unit. This process is repeated across multiple layers and allows us to model non-linear situations more effectively. Our Multi-Layer Perceptron was trained using a back-propagation algorithm that utilizes stochastic gradient descent where the weight update rule is given by:

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta; x^i, y^i).$$

where  $(x^{(i)}, y^{(i)})$  is a specific pair from the training set. The variable  $x$  represents the training data, and  $y$  represents the specific target for that sample. Then, for a multi-class problem, a Softmax layer is used as the output layer.

### 3.4.1 Implementation

In order to implement a Multi-Layer Perceptron, we used a library known as 'scikit-neuralnetwork'. This allowed us to easily build a Multi-Layer Perceptron using Python. 'Scikit-neuralnetwork' is a wrapper for two other Neural Network libraries known as 'lasagne' and 'pylearn2'. We decided to use the 'lasagne' backend since the 'pylearn2' backend is deprecated. The Neural Network was initialized with the following baseline parameters before optimization:

```
from sknn.backend import lasagne
from sknn.mlp import Classifier, Layer

Class NeuralNetwork(Classifier):

    def _init_(self, params=None):
        self.clf = Classifier(layers=[Layer("Tanh", units=128), Layer("Softmax")])
```

Then, the model was trained and predicted on with the following functions:

```
def learn(self, Xtrain, ytrain):
    self.clf.fit(np.array(Xtrain), np.array(ytrain))

def predict(self, Xtest, encoder):
    probs = self.clf.predict_proba(np.array(Xtest))
    return probs
```

### 3.4.2 Optimizations

In order to improve the performance of our Neural Network, the decision was made to scale the features being used. This was done using the scikit-learn "StandardScaler". The "StandardScaler" standardizes the features by removing the mean and scaling the features to unit variance. This was done with the following code:

```
xtrain = self.scaler.fit_transform(Xtrain)
.
.
.
xtest = self.scaler.transform(Xtest)
```

Next, in order to improve the modeling capabilities of our Neural Network, we decided to use a four-layer Neural Network as opposed to a two-layer Neural Network, as follows:

```
from sknn.backend import lasagne
from sknn.mlp import Classifier, Layer

Class NeuralNetwork(Classifier):

    def _init_(self, params=None):
        self.clf = Classifier(layers=[Layer("Tanh", units=128), Layer("Tanh", units=128),
                                      Layer("Sigmoid", units=128), Layer("Softmax")])
```

The decision was made to utilize randomized grid search cross validation in an attempt to optimize our Neural Network's parameters.

After running this, we came to the following Neural Network:

```
from sknn.backend import lasagne
from sknn.mlp import Classifier, Layer

class NeuralNetwork(Classifier):

    def _init_(self, params=None):
        self.clf = Classifier(layers=[Layer("Tanh", units=128), Layer("Tanh", units=128),
                                       Layer("Sigmoid", units=128), Layer("Softmax")],
                              learning_rate=0.08, batch_size=100, dropout_rate=0.4,
                              learning_rule="adagrad")
```

The "dropout\_rate" refers to dropping out a certain percentage of unit activations in each. In this case, forty-percent of the unit activations were dropped in each layer. This is done to prevent overfitting. By default, the Multi-Layer Perceptron will use standard stochastic gradient descent to learn the weights. Adaptive Gradient Descent, or adagrad, adjusts the learning rate on each iteration of the descent.

## 4 Results

After running each of our algorithms on the dataset, the results were as follows:

Model	Log Loss	Training Time (s)
Random Forest	2.592	8.448
Naive Bayes	2.563	0.831
Logistic Regression	2.533	262.242
Neural Network	2.377	153.997



We decided to submit the Neural Network model to Kaggle due to giving the best performance on our dataset. The following is the result of that submission:

 Submitted an entry to [San Francisco Crime Classification](#), obtaining 2.37747 7 days ago

This submission put us at position 93 out of 964 entrants, or the top 10 percent of all entrants.

## 5 Conclusion and Discussion

Out of all of the models that we developed, we found that the Neural Network model worked the best for this particular challenge. The Neural Network model allowed for various optimizations to be made, and its ability to model non-linear situations effectively made it a clear choice that required further exploring. However, after further research, a Gradient Boosted Decision Tree method was used so that we could compare its performance to our Neural Network model. Using the XGBoost library with our engineered features, we were able to achieve the following result:

71  115  2.33855 18 Fri, 11 Dec 2015 19:54:59 (-3.6d)

This put us at 71 out of 964 entrants, the top 7 percent of all entrants. Due to time constraints,

we were not able to explore this model further. However, for future work on this project the use of averaging, which constitutes taking the mean of our best individual model predictions, may lower our Log Loss score. Another option that needs to be explored is the use of a Majority Vote ensemble method consisting of our Neural Network and the XGBoost models, as we believe this could provide improved results in comparison to a single model.

## 6 Contribution

We collaborated on all aspects of the project and the majority of the work was done in a pair programming structure. Jordan Gaeta spent most of his time working on the data processing and feature engineering, and Geri Rista spent his time working on tuning the models.

## 7 References

- [1] <http://fastml.com/regularizing-neural-networks-with-dropout-and-with-dropconnect/>
- [2] <http://mlwave.com/kaggle-ensembling-guide/>
- [3] <https://www.kaggle.com/tfurmston/sf-crime/crimes-that-occur-on-a-street-corner/files>
- [4] <https://github.com/aigamedev/scikit-neuralnetwork>
- [5] <https://github.com/scikit-learn/scikit-learn>
- [6] <https://github.com/pydata/pandas>
- [7] Radivojac, Predrag, and Martha White. B555: Machine Learning Notes. Bloomington, ... 2015. Print.