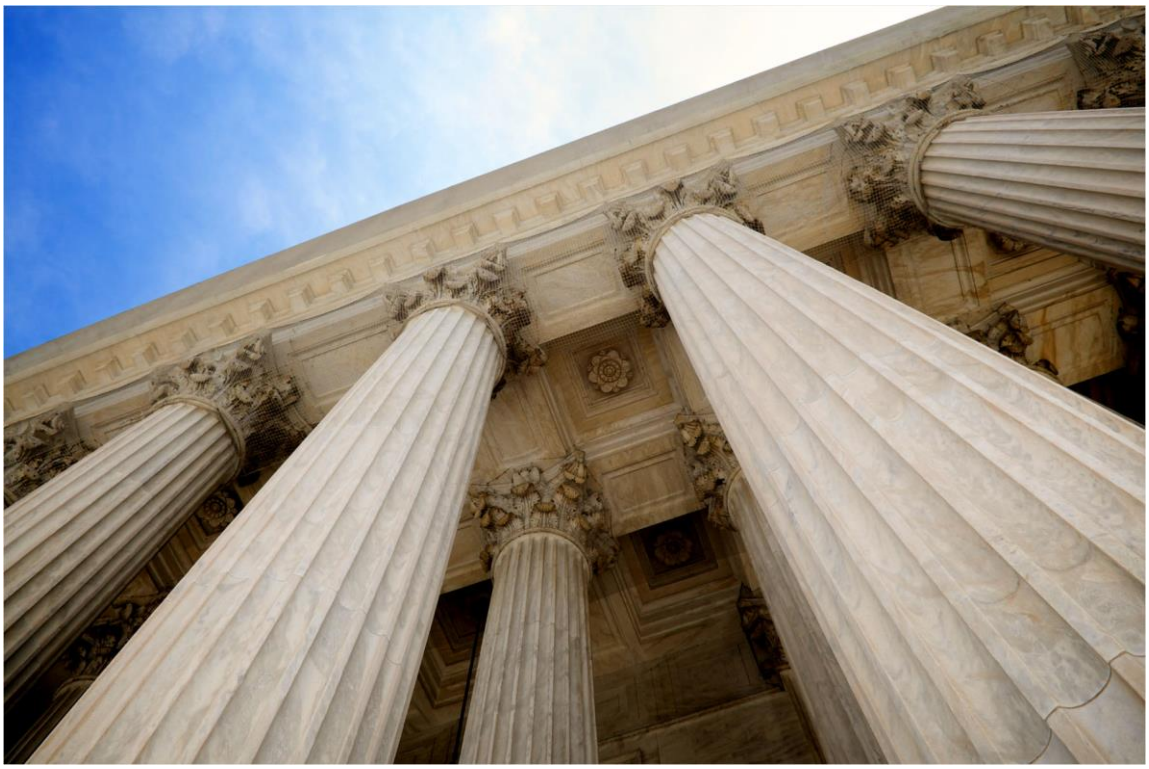


# Capstone Project

## SAN FRANCISCO CRIME CLASSIFICATION

Mohammed Deifallah | Machine Learning Nanodegree Program | 17/08/18

# Definition

## PROJECT OVERVIEW

From 1934 to 1963, **San Francisco** was infamous for housing some of the world's most notorious criminals on the inescapable island of Alcatraz. Today, the city is known more for its tech scene than its criminal past. But, with rising wealth inequality, housing shortages, and a proliferation of expensive digital toys riding BART to work, there is no scarcity of crime in the city by the bay. Hence, **Safety** is a great issue to be concerned about. I think it's a great deal to focus on producing a very good solution for that purpose.

The project is mainly concentrated on ***supervised learning*** technique. As it's a very interesting and trending field, it's an opportunity to make use of algorithms and techniques learned through the course, to solve such a problem. Also, note that this problem has a huge intention from other organizations, and there're many papers ae published about it, such as this paper, and this paper too.

The Dataset used for that problem is the dataset provided by **Kaggle**. The dataset contains incidents derived from **SFPD** Crime Incident Reporting system. The data ranges from 1/1/2003 to 5/13/2015. The data is divided to 2 sets, Training set and Testing set. The training set and test set rotate every week, meaning week 1,3,5,7... belong to test set, week 2,4,6,8 belong to training set.

## PROBLEM STATEMENT

As it's a competition on Kaggle, so it's better to copy the problem statement here as it is. It's as follows: "From 1934 to 1963, San Francisco was infamous for housing some of the world's most notorious criminals on the inescapable island of Alcatraz. Today, the city is known more for its tech scene than its criminal past. But, with rising wealth inequality, housing shortages, and a proliferation of expensive digital toys riding BART to work, there is no scarcity of crime in the city by the bay. From Sunset to SOMA, and Marina to Excelsior, this competition's dataset provides nearly 12 years of crime reports from across all of San Francisco's neighborhoods. Given time and location, you must predict the category of crime that occurred.". You can find the competition here. As can be observed from the description, it's a *classification* problem, where the input is about the climate of the incident, as: the district, the crime category, the date of the crime... etc. The target variable is the crime category in the testing set.

The project is mainly concentrated on ***supervised learning*** technique. As it's a very interesting and trending field, it's an opportunity to make use of algorithms and techniques learned through the course, to solve such a problem. As performed before, ***supervised learning*** can be used for such a classification problem like what is done in *Charity donors* project.

The training data has 9 features, which are given in the next figure:

**Data fields**

- **Dates** - timestamp of the crime incident
- **Category** - category of the crime incident (only in train.csv). **This is the target variable you 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

where the target variable is the **crime category**.

As mentioned earlier, the project is an application for ***supervised learning***. It means that the best way to make the most use of it is to apply many algorithms and techniques learned in that field, as: *Linear Regression*, *Decision Trees*, *Neural Networks*, *SVM (Support Vector Machine)*, *Ensemble B&B* ... etc. Some or all those techniques may be implemented in the project and compare the results for reaching the best accuracy.

METRIC

As it's a competition, it's reasonable to consider the same evaluation metrics of the competition itself to apply on the implementation of the project. It's evaluated using multi-class logarithmic loss. The formula is shown in the figure below:

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

Also, **F-score** is to be used as a metric for the solution. So, I'd like to say that it's logical and reasonable to use those to metrics to measure the solution of a multi-class problem, and to avoid using only **accuracy**, as the data may be suffering from imbalancing.

# Analysis

## DATA EXPLORATION

The dataset contains incidents derived from **SFPD** Crime Incident Reporting system. The data ranges from 1/1/2003 to 5/13/2015. In details, the figure below explains the data fields:

**Data fields**

- **Dates** - timestamp of the crime incident
- **Category** - category of the crime incident (only in train.csv). **This is the target variable you 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

Training data has 878049 points with 9 variables each, while Testing data has 884262 points with 7 variables each. Also, it's shown above that the target variable is 'Category'. So, we give more attention to that column. So, it's attached the number of occurrences for each category in descending order:

```
LARCENY/THEFT 174885
OTHER OFFENSES 126165
NON-CRIMINAL 92300
ASSAULT 76872
DRUG/NARCOTIC 53971
VEHICLE THEFT 53772
VANDALISM 44724
WARRANTS 42206
BURGLARY 36754
SUSPICIOUS OCC 31412
MISSING PERSON 25989
ROBBERY 22999
FRAUD 16679
FORGERY/COUNTERFEITING 10609
SECONDARY CODES 9985
WEAPON LAWS 8555
PROSTITUTION 7484
TRESPASS 7325
```

```
STOLEN PROPERTY 4539
SEX OFFENSES FORCIBLE 4387
DISORDERLY CONDUCT 4318
DRUNKENNESS 4280
RECOVERED VEHICLE 3138
KIDNAPPING 2341
DRIVING UNDER THE INFLUENCE 2268
RUNAWAY 1946
LIQUOR LAWS 1903
ARSON 1513
LOITERING 1225
EMBEZZLEMENT 1166
SUICIDE 508
FAMILY OFFENSES 491
BAD CHECKS 406
BRIBERY 289
EXTORTION 256
SEX OFFENSES NON FORCIBLE 148
GAMBLING 146
PORNOGRAPHY/OBSCENE MAT 22
TREA 6
```

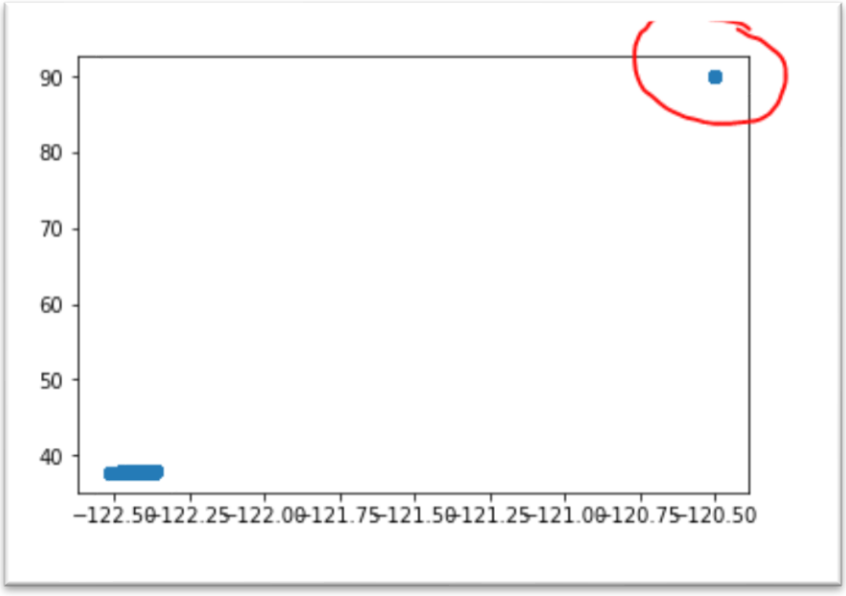
For a wider point of view about the data, we can see that figure:

| | Dates | Category | DayOfWeek | PdDistrict | Address | X | Y |
|---|---|---|---|---|---|---|---|
| 0 | 2015-05-13 23:53:00 | WARRANTS | Wednesday | NORTHERN | OAK ST / LAGUNA ST | -122.425892 | 37.774599 |
| 1 | 2015-05-13 23:53:00 | OTHER OFFENSES | Wednesday | NORTHERN | OAK ST / LAGUNA ST | -122.425892 | 37.774599 |
| 2 | 2015-05-13 23:33:00 | OTHER OFFENSES | Wednesday | NORTHERN | VANNESS AV / GREENWICH ST | -122.424363 | 37.800414 |
| 3 | 2015-05-13 23:30:00 | LARCENY/THEFT | Wednesday | NORTHERN | 1500 Block of LOMBARD ST | -122.426995 | 37.800873 |
| 4 | 2015-05-13 23:30:00 | LARCENY/THEFT | Wednesday | PARK | 100 Block of BRODERICK ST | -122.438738 | 37.771541 |
| 5 | 2015-05-13 23:30:00 | LARCENY/THEFT | Wednesday | INGLESIDE | 0 Block of TEDDY AV | -122.403252 | 37.713431 |
| 6 | 2015-05-13 23:30:00 | VEHICLE THEFT | Wednesday | INGLESIDE | AVALON AV / PERU AV | -122.423327 | 37.725138 |
| 7 | 2015-05-13 23:30:00 | VEHICLE THEFT | Wednesday | BAYVIEW | KIRKWOOD AV / DONAHUE ST | -122.371274 | 37.727564 |
| 8 | 2015-05-13 23:00:00 | LARCENY/THEFT | Wednesday | RICHMOND | 600 Block of 47TH AV | -122.508194 | 37.776601 |
| 9 | 2015-05-13 23:00:00 | LARCENY/THEFT | Wednesday | CENTRAL | JEFFERSON ST / LEAVENWORTH ST | -122.419088 | 37.807802 |

Also, we can take a look on the geography of the data by the description of longitude and latitude features:

```
Longitudes:
count    878049.000000
mean       -122.422616
std           0.030354
min        -122.513642
25%        -122.432952
50%        -122.416420
75%        -122.406959
max        -120.500000
Name: X, dtype: float64

Latitudes:
count    878049.000000
mean         37.771020
std           0.456893
min          37.707879
25%          37.752427
50%          37.775421
75%          37.784369
max          90.000000
Name: Y, dtype: float64
```

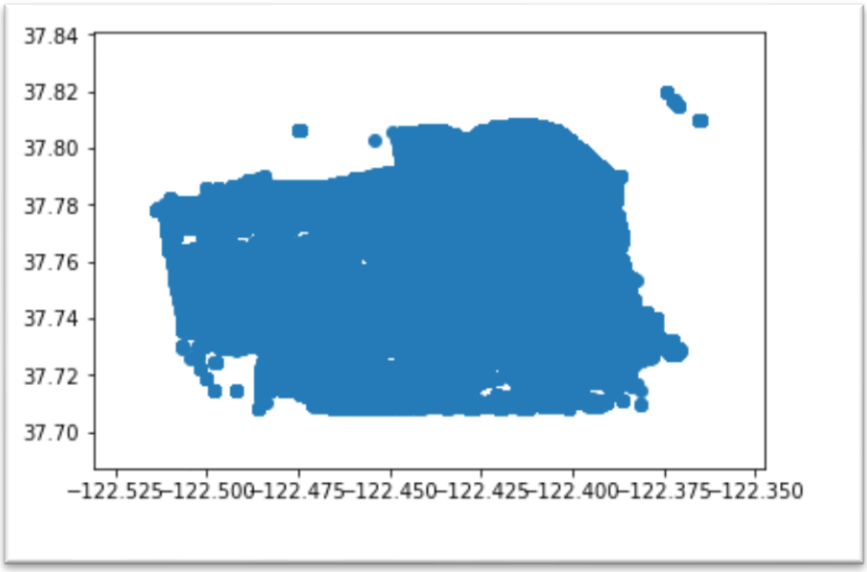
As shown, the data has some corrupted points that don't belong to San Francisco, and the notebook has the whole details about that issue and San Francisco location on Google Maps. But to be illustrative, that's a plot for the longitude vs. latitude to detect the outliers:




That's why the data needs to be preprocessed to get rid of those invalid points.

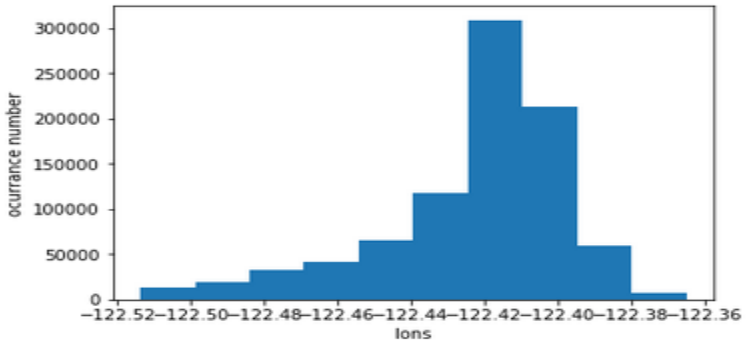
## EXPLORATORY VISUALIZATION

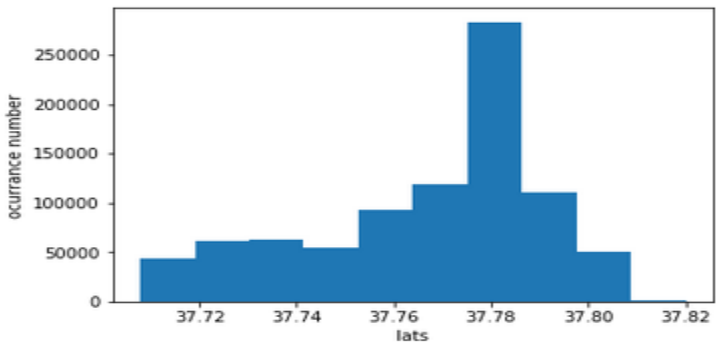
### Visualization by location




This is the previous plot but after removing the outliers. Now, it's about the crimes inside the borders of San Francisco.

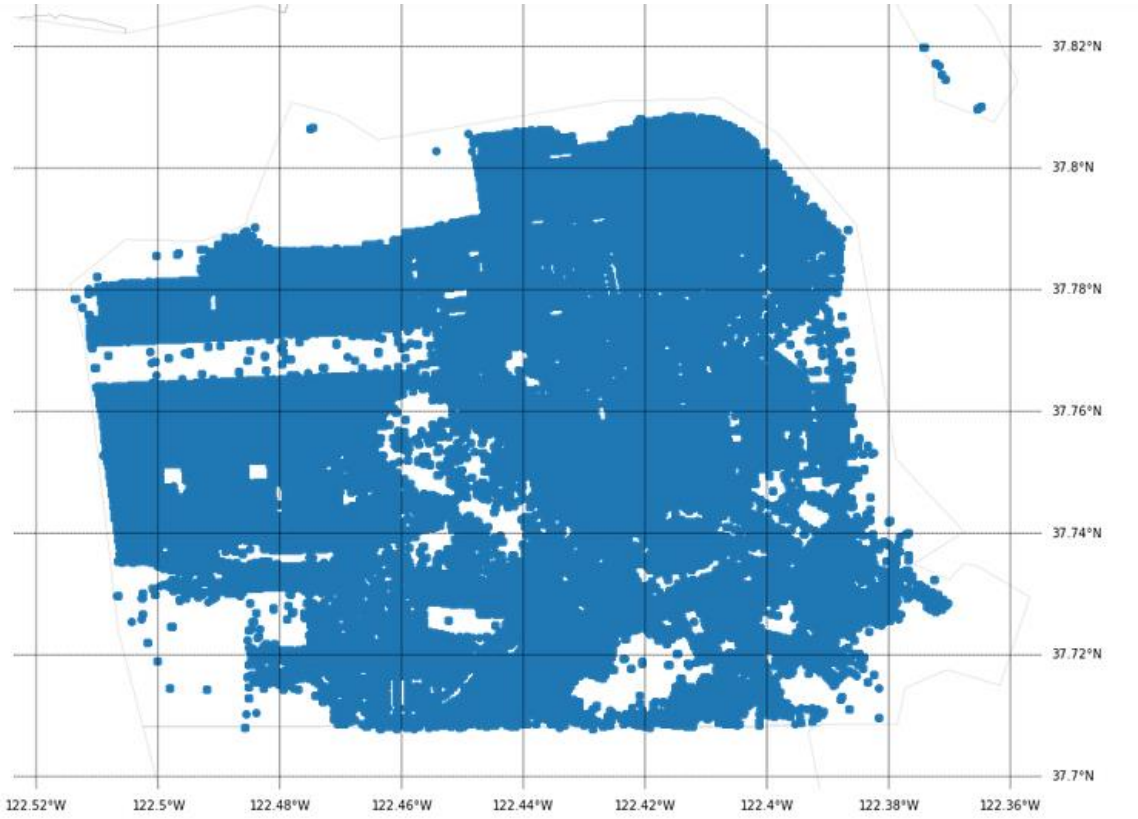
Also, we can compare individually the longitude and latitude with respect to the crime frequencies, as shown in the figure below:



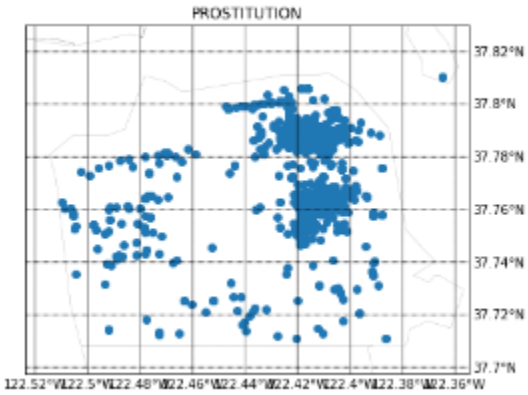



It's clear that the most crimes are in the location of longitude = [-122.44, -122.40] and latitude = [37.76, 37.80]

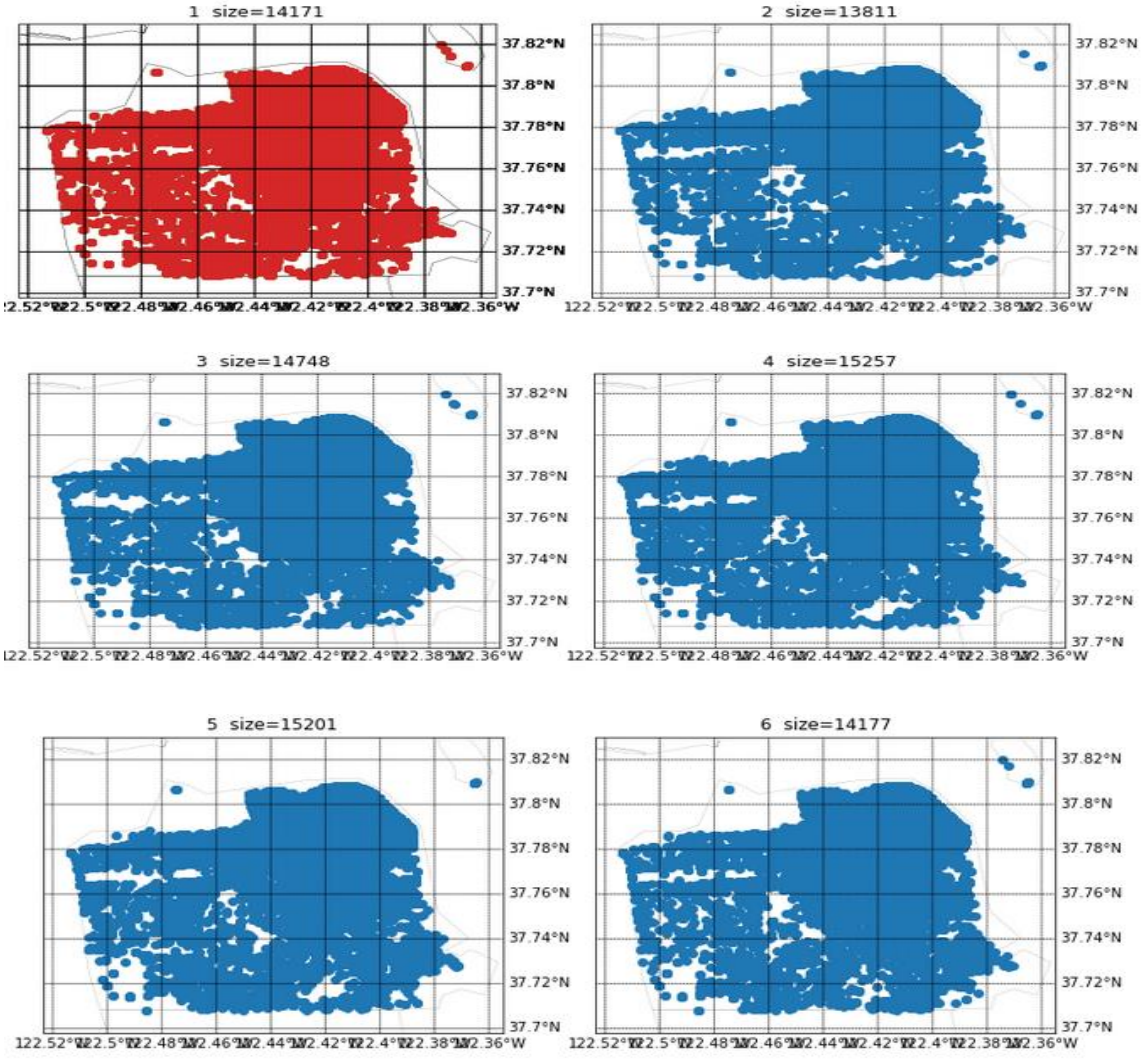
One clearer plot using 'Basemap' library:




Another thing, we can plot a geographical map for each category with respect to its frequency, what can help us to know in which regions a specific crime category happens most. For example, this figure below shows the case of prostitution:

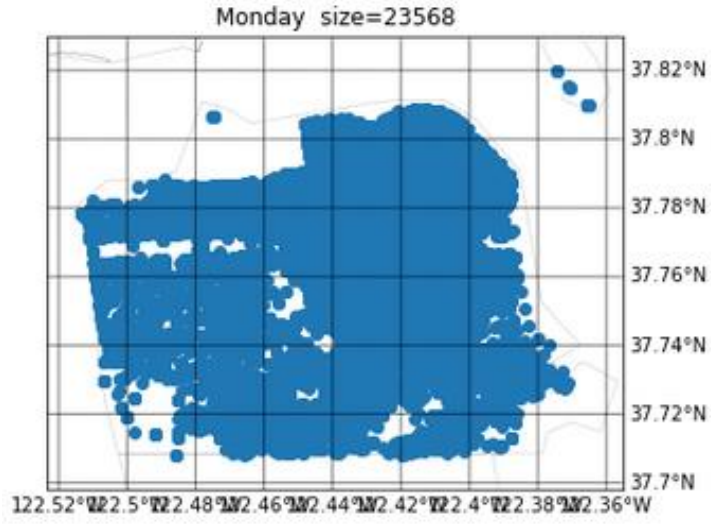


Visualization by time




Those figures above are samples for locating the crimes according to the month they happened in. But, note that those figures are plotted for the most committed crime, which is 'Theft' in our case. Take April as an example for a month with higher rate of crime.

This example below shows the same thing but for the days not months. Take Monday as an example:




## ALGORITHMS AND TECHNIQUES

As mentioned in the proposal, the algorithms which will be used to solve the problem are: *Decision Trees*, *Neural Networks*, *SVM (Support Vector Machine)* and *Ensemble B&B* (*XGBoost* classifier). As it's a **classification** problem, so it doesn't make any sense to use any *Linear Regression* algorithms for this problem.

Let's talk deeper about each algorithm:

1) Decision Trees: It's a representation of the problem, where the leaves are the target classes of the problem, and the branches are the conditions and interleaving of the data features to reach those leaves. It's always an issue to concern about the depth of the tree, as the deeper the depth, the more tendency to overfitting and the longer the training time.
2) Neural Networks: MLPs (Multi-Layer Perceptron) are useful for research regarding their ability to solve problems stochastically, Learning occurs in the perceptron by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected result.
3) SVM (Support Vector Machine): It's working well with linearly-separable data to solve classification problems in order to maximize the margin between classes, where the margin is the distance between the two closest points of different classes to the barrier between them. Also, for non-linear features, it applies **Kernel Trick**, where it creates a new hyperplane to separate the labels.
4) XGBoost: It's an **Ensemble** technique where new learners are used to correct the error rate of other learners. It's called **Gradient Boosting** algorithm as it makes use of ***Gradient Descent*** algorithm.

Another thing is that for preprocessing, we may need to use encoding to solve the problem of categorical features. So, *One-Hot Encoding* is used for features with small

number of unique values not to increase the dimensionality, and *Label Encoding* is used for the other categorical ones. This figure shows the number of unique values of mentioned features:

```
feature: Dates    unique_size: 389229
feature: DayOfWeek    unique_size: 7
feature: PdDistrict    unique_size: 10
feature: Address    unique_size: 23191
```

### BENCHMARK

The easiest way to compare the solution, or measure the model selected is to show where it stands among those scores in the competition leaderboard on Kaggle. But in this project, we can consider a **random version** as a model to compare to. The solution must be more accurate than the random version with a reasonable rate to be regarded as a valid solution.

## Methodology

### DATA PREPROCESSING

As mentioned earlier, the crime category is the target variable. On the other hand, there're 2 redundant features that don't exist in the testing data, which means that they can be dropped with no negative effect.

```
train_data = train_data.drop(['Descript', 'Resolution'], axis=1)
```

Also, as mentioned about San Francisco boundaries, invalid locations and outliers are removed from the training data, so they can't affect the models.

Encoding is used in the project to solve the problem of categorical features. So, *One-Hot Encoding* is used for features with small number of unique values not to increase the dimensionality, and *Label Encoding* is used for the other categorical ones.

### IMPLEMENTATION

Training data is divided to training set and cross-validation set, with *test_size=.2*.

And, now let's take a closer and deeper look to the implementation of models to make our work reproducible as much as possible for further extensibility and maintainability.

There's a general function to train and test all the models, even the random predictor one.

```
def train_test_models(models, names=None):
```

That function takes a list of models with their descriptive names. Each model is being attached with 6 attributes: time taken, **F**-score and **logloss**. Three for the training process, and the other three for the testing process.

Also, there's a function to visualize those attributes to be more illustrative and clear.

```
def visualize(results, names=['random_model', 'DecisionTree', 'MLP_NN', 'XGBoost'], random=True):
```

results: dictionary which has the models as keys, and the list of their attributes as values.

names: list of model descriptive names to set them as a title for the bars.

random: Boolean if the results of the random model will be visualized or not.

For the random predictor implementation, it's an easy mission, and a code snippet could show a brief explanation about it:

```python
class random_model:

    def __init__(self, categories):
        self.categories = categories
        self.classes_ = categories

    # no need for fit here
    def fit(self, X_train, y_train): pass

    def predict(self, X):
        result = [[] for j in range(len(X))]
        for i in range(len(X)):
            result[i] = random.choice(self.categories)
        return result

    def predict_proba(self, X):
        result = [[] for j in range(len(X))]
        for i in range(len(X)):
            row = [0.0] * len(self.categories)
            prediction = random.choice(self.categories)
            for j in range(len(self.categories)):
                if(self.categories[j] == prediction):
                    row[j] = 1.0
                    break
            result[i] = row
        return result
```

**fit**: This function does nothing, as it's a random predictor, then there's no need for such a function. It's just for the interface and generalization.

**predict:** This function is simply predicting a random category from the crime categories.

**predict_proba:** This function only returns probability rather than the crime category itseld. It simply returns 1 for the correctly predicted categories, and 0 otherwise.

For other models, this is a code snippet for them:

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier

# initializing models
model_tree = DecisionTreeClassifier(class_weight='balanced', random_state=0)
model_NN = MLPClassifier(solver='lbfgs', random_state=0)
model_SVC = SVC(probability=True, class_weight='balanced', random_state=0)
model_XGB = XGBClassifier(learning_rate=0.1, silent=True, random_state=0)
```

**class_weight:** This parameter in the *balanced* mode is working on solving the problem of the data unbalancing, where the weights are inversely proportional to the frequencies of different classes. So, there's no need to balance the data by hand.

**solver:** It's an optimizer in the family of **quasi-Newton** methods. It converges faster and perform better, compared by other available solvers.

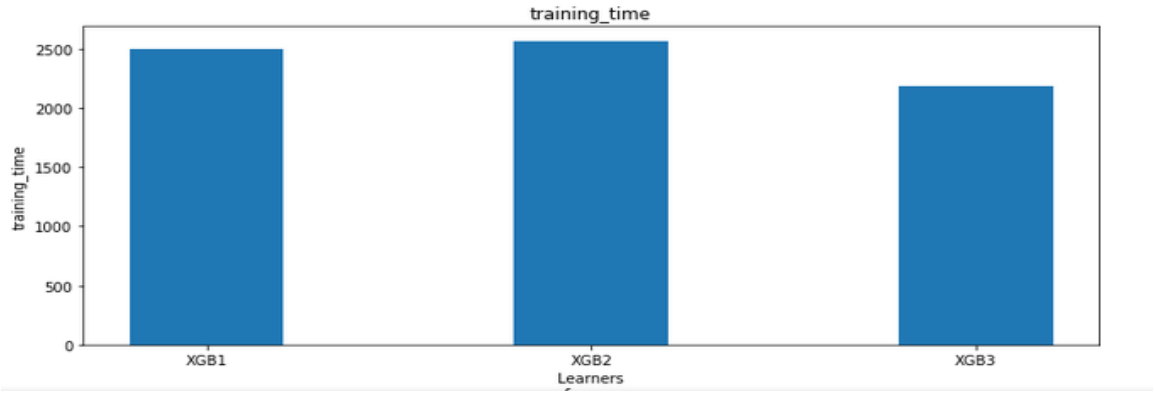
**probability:** This parameter is set for producing probability estimates, hence, calling **fit()** method. But, note that this enabled parameter is slowing down the method.

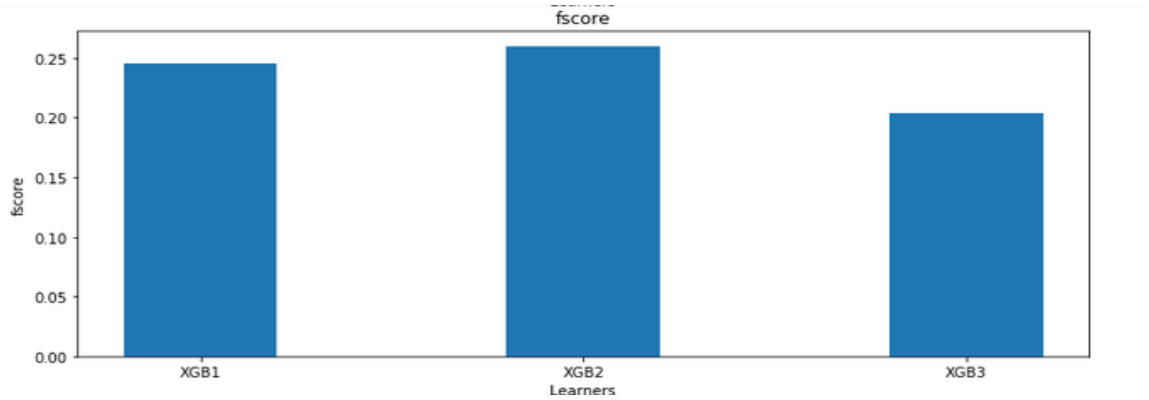
**learning_rate:** a hyper-parameter which controls the convergence and the number of iterations. Here, it's equal to .1 which is a reasonable mean value for the model. The best interval of learning_rate parameter is [0.01, 0.3]

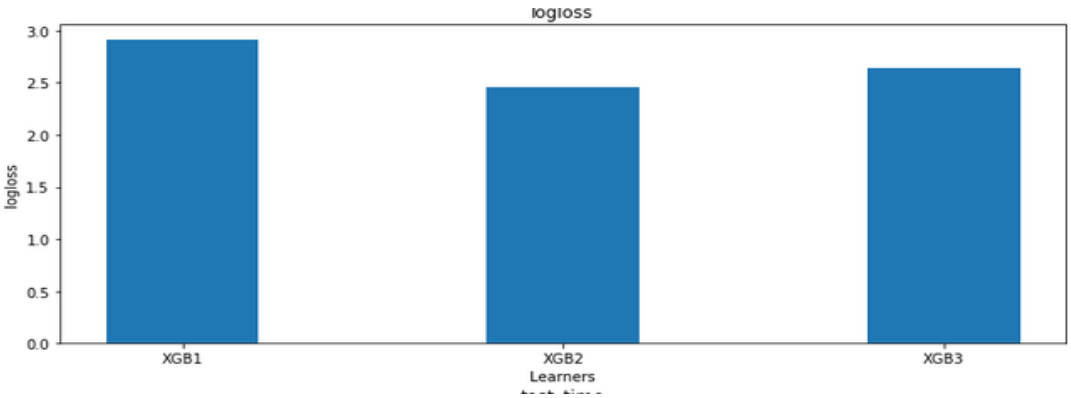
REFINEMENT

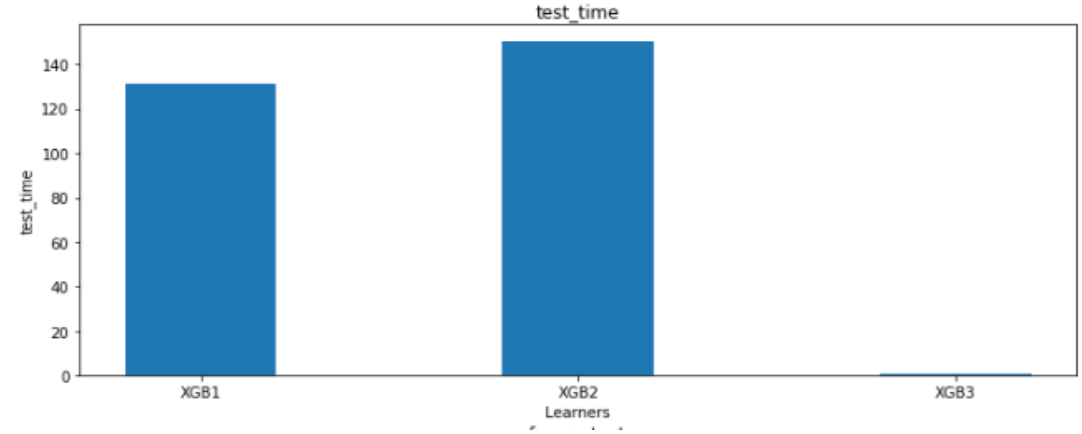
The comparison between all models will be explained in detail in further section, but the important thing now is to note that **XGBoost Classifier** has the best scores, so it's tuned and refined to have a better version of it as shown below:

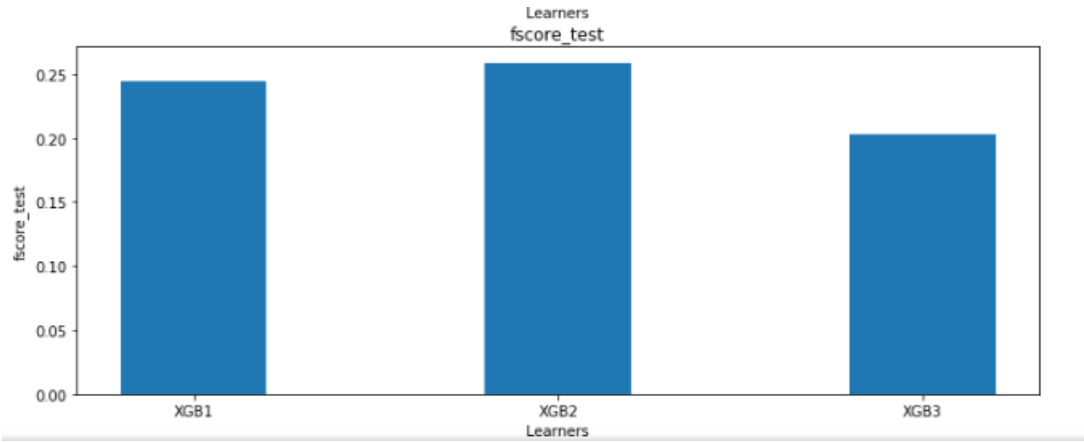
```
model_XGB1 = XGBClassifier(learning_rate=0.01, silent=1, random_state=0, booster='gbtree', max_depth=3)
model_XGB2 = XGBClassifier(learning_rate=0.1, silent=1, random_state=0, booster='gbtree', max_depth=3)
model_XGB3 = XGBClassifier(learning_rate=0.3, silent=1, random_state=0, booster='gblinear')


models = [model_XGB1, model_XGB2,model_XGB3]
names = ["XGB1", "XGB2", "XGB3"]
train_test_models(models, names=names)
```

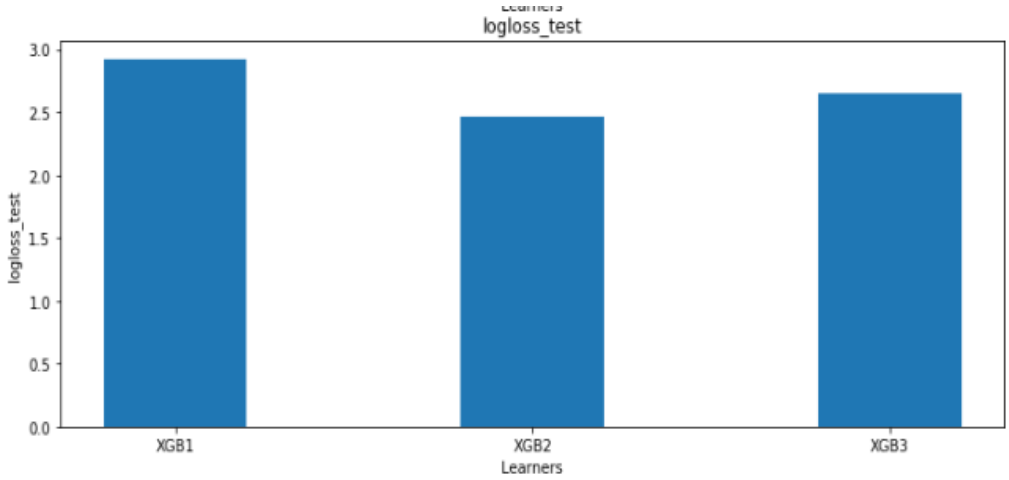



fscore
0.25
0.20
0.15
0.10
0.05
0.00
fscore
XGB1
XGB2
XGB3
Learners


logloss
3.0
2.5
2.0
1.5
1.0
0.5
0.0
logloss
XGB1
XGB2
XGB3
Learners


test_time
140
120
100
80
60
40
20
0
test_time
XGB1
XGB2
XGB3
Learners



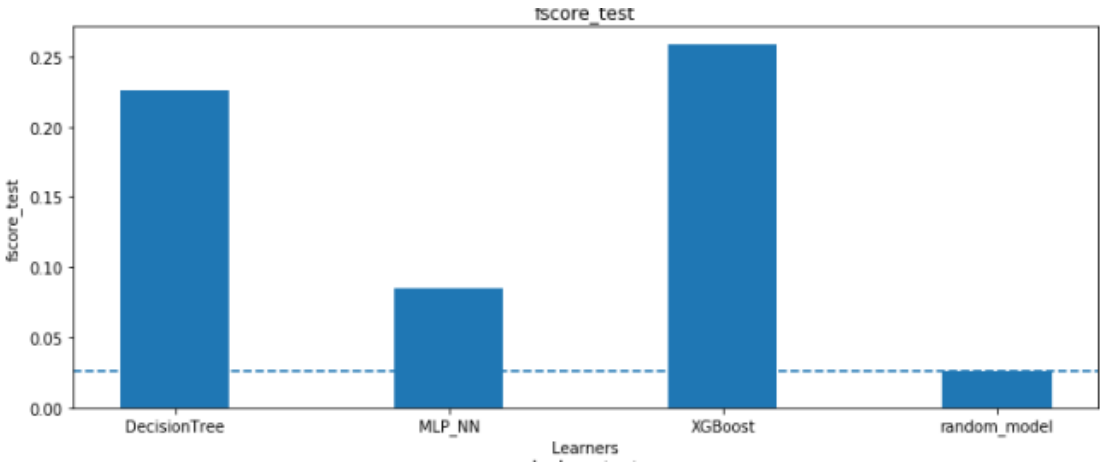



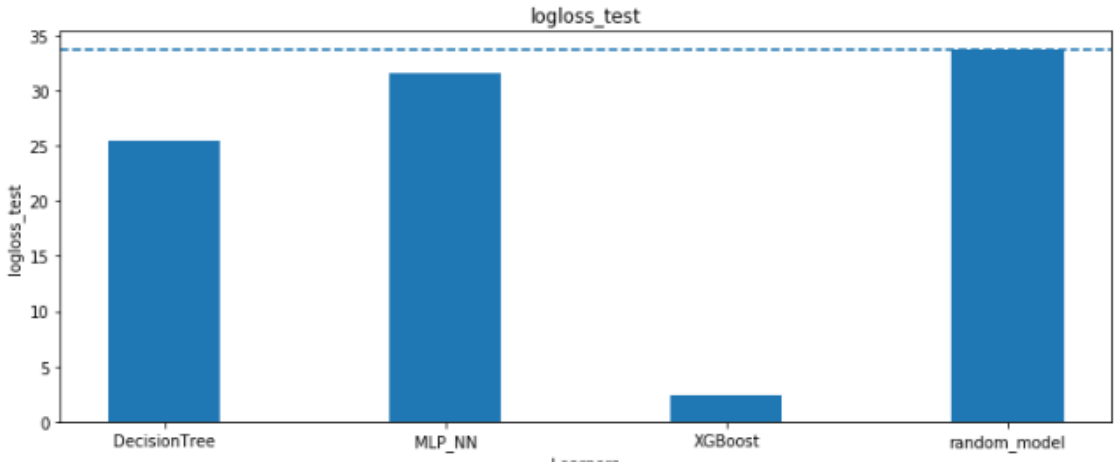

I think it's reasonable to choose the second tuned model to be our main model, as it has achieved the best **F**-score and **logloss** for both training and testing sets.

# Results

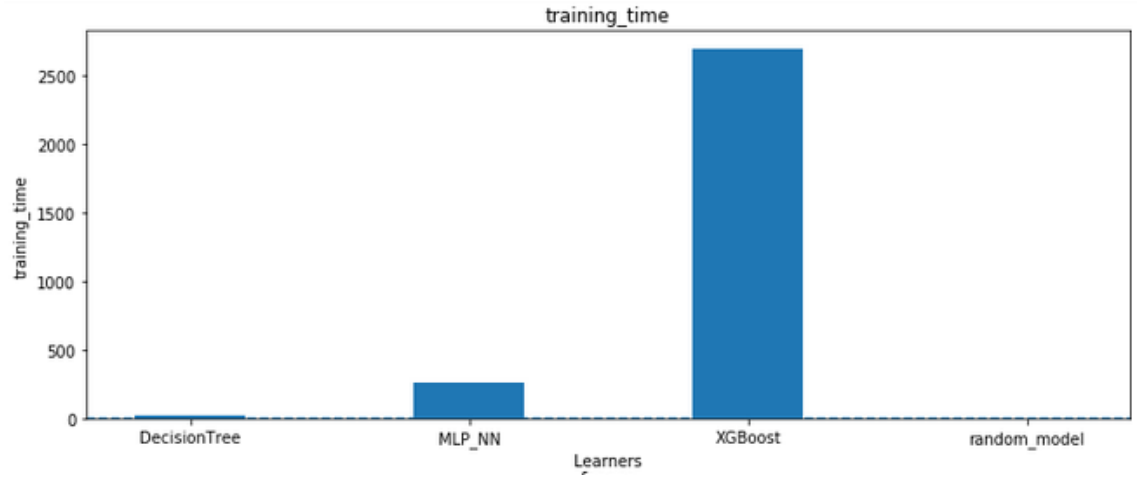
## MODEL EVALUATION AND VALIDATION

The model is general enough to be able to solve the problem as much accurately as possible, and its reliability can be proved by comparing this classifier with respect to other classifiers implemented in the project. For example, check these two plots to determine that this model is the best general model to solve the problem with achieving the best results for both measure metrics:




According to the training time, then **XGBoost** classifier takes longer time than other models, as this plot shows that:




But this time is reasonable and enough to result in good metrics as mentioned before.

Now, let's move to the final model after refinement. As shown before, the final model's qualities are:

```
model_XGB2 = XGBClassifier(learning_rate=0.1, silent=1, random_state=0, booster='gbtree', max_depth=3)
```

Where learning_rate: a hyper-parameter which controls the convergence and the number of iterations. Here, it's equal to .1 which is a reasonable mean value for the model. The best interval of learning_rate parameter is [0.01, 0.3]

silent: boolean whether to print messages while running boosting. Here, it's set to be 1 in order not to be distracted by the classifier's messages while training.

random_state: the seed of random number generator.

booster: specifies which booster to use. Here, it's set to 'gbtree' which is a tree-based model.

max_depth: the maximum depth of the model's tree. The best interval is [3, 10]. Here, max_depth is set to 3 to have the least reasonable maximum depth to have a reasonable time, while having the best performance.

Now, it's time to show what results it has achieved. Here's a figure for this information:

```
model: XGB2
fscore:          0.25906304946717257
logloss:         2.460570737321583
train time:      2562.8924474716187
fscore_test:              0.25872879377210317
logloss_test:    2.466384636960238
test time:       150.39454102516174
```

Although its training time is too high with respect to models of other classifiers, it results in a very good **F**-score and **logloss** score in the *validation* set.

## JUSTIFICATION

Here, we'll discuss why ***XGBoost*** Classifier is prior to other models according to the benchmark used for this project.

First of all, it must be noted that *SVC* is the slowest one, as it consumes a huge amount of resources that I can't get. Also, it takes a lot amount of time for both training and testing. So, according to that link we can assume that SVC will not achieve scores better than other models, especially while in our case we don't have linearly separable data.

Here's a table for the **logloss** score for the other 4 models on the testing data:

| Model | Logloss |
|---|---|
| Random Predictor | 33.665 |
| Decision Tree | 25.4599 |
| Neural Network | 31.5886 |
| XGBoost | 2.46638 |

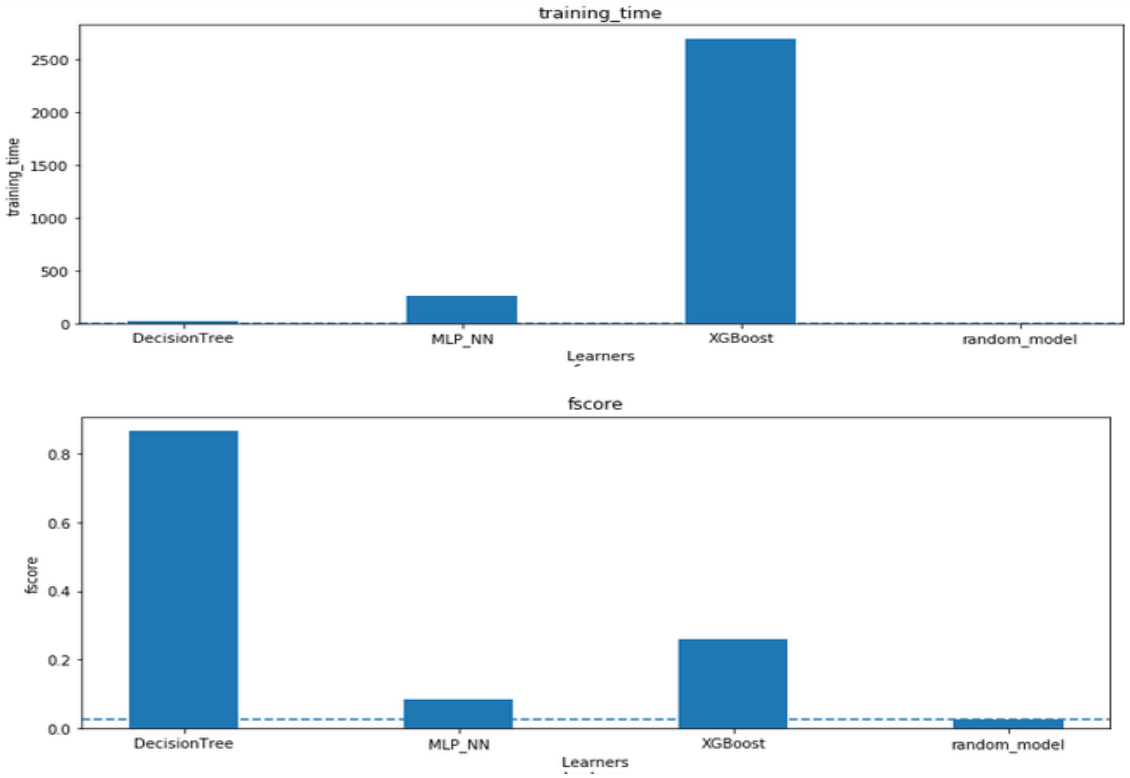
That's why **XGBoost** classifier is the best model and is chosen to be refined and tuned as shown in the previous section in order to improve its performance.

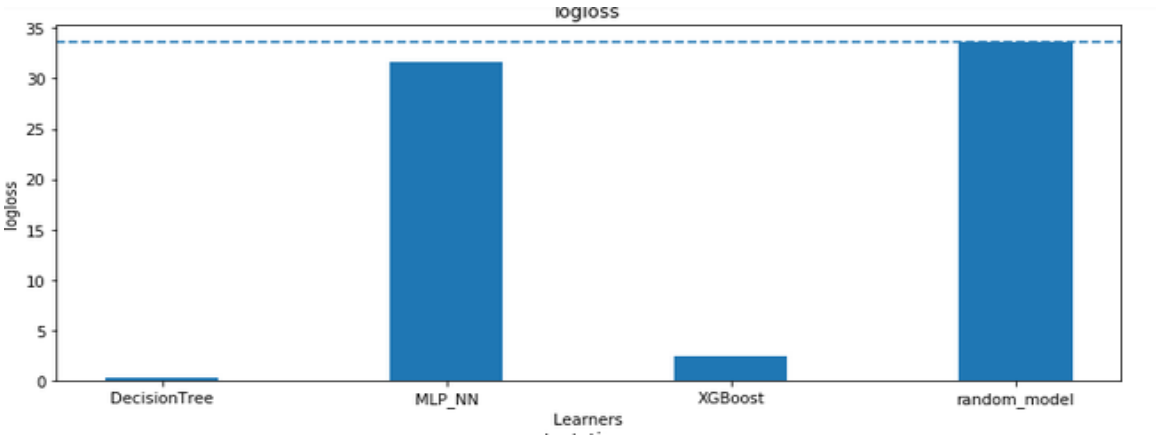
Also, note that the final output result is submitted on Kaggle, and it gets a score of 2.85673 to lay as 1638 out of 2335 competitors in the contest. You can jump to the leaderboard in that link.

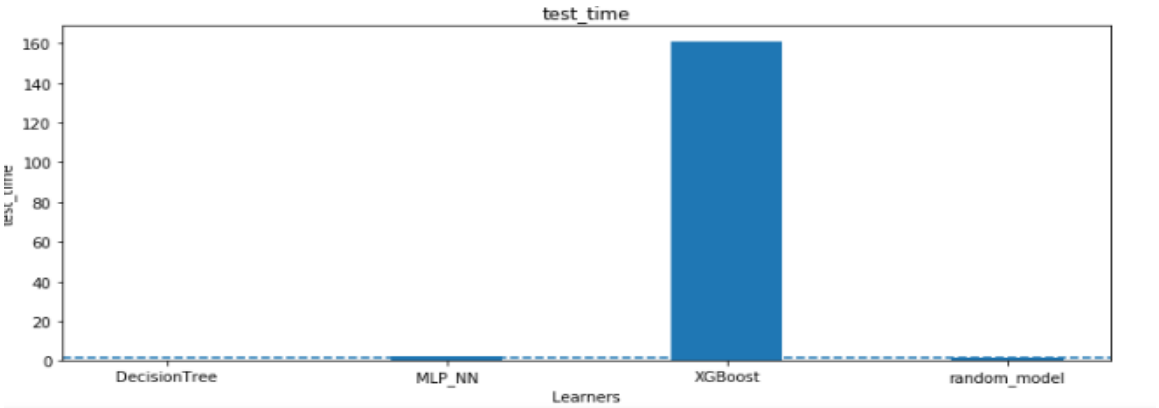
# Conclusion

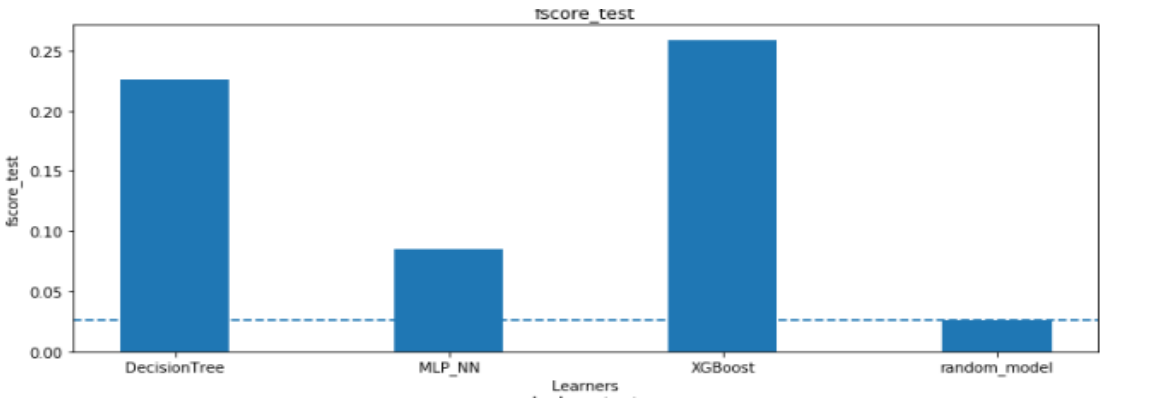
### FREE-FORM VISUALIZATION

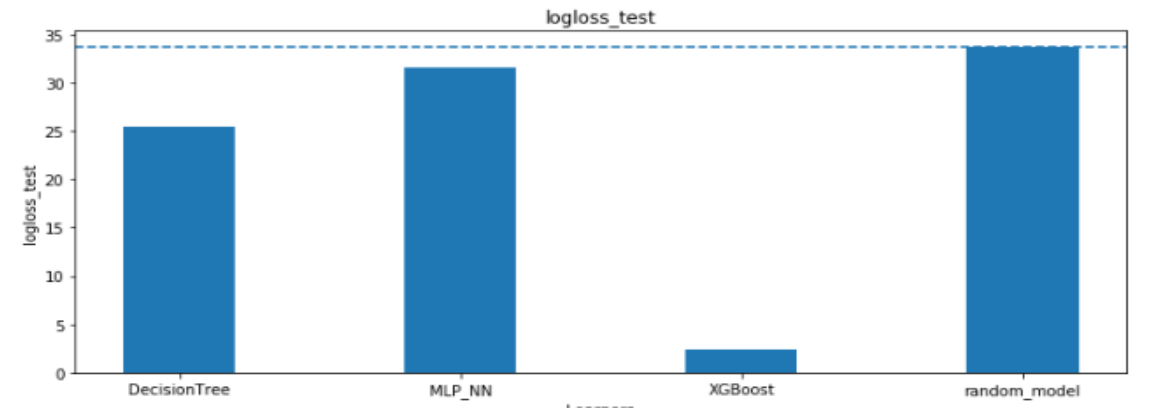
All the models are compared to each other through bar plots for each measure metric using matplotlib library, as follows:




logloss
35
30
25
20
15
10
5
0
logloss
DecisionTree
MLP_NN
XGBoost
random_model
Learners


test_time
160
140
120
100
80
60
40
20
0
test_time
DecisionTree
MLP_NN
XGBoost
random_model
Learners


fscore_test
0.25
0.20
0.15
0.10
0.05
0.00
fscore_test
DecisionTree
MLP_NN
XGBoost
random_model
Learners


logloss_test
35
30
25
20
15
10
5
0
logloss_test
DecisionTree
MLP_NN
XGBoost
random_model

Take attention that there's always a dashed line to make it easy to compare all implemented models with respect to the random predictor model.

## REFLECTION

In this project, we are to compare 4 models to each other to solve the problem of **San Francisco Crime Classification**.

At first, we remove the redundant features that don't exist in the testing set and not target variables. So, they can reduce some dimensionality with no negative effect on our performance. Another thing is that we visualize the data to detect the outliers that lay out of San Francisco boundaries. That helps improve the accuracy of the solution.

Now, it's the turn of categorical features that have to be encoded using both *One-Hot Encoding* & *Label Encoding*. The choice of which technique is applied on each feature is through that link. The **Curse of Dimensionality** requires us to apply *One-Hot Encoding* on features with a small number of unique values. Hence, *Label Encoding* is applied on other features.

The data now is passed to the models to be fit. The visualizations made through the process show that **XGBoost Classifier** is the best model for the validation set as it achieves the best **F-score** & **logloss**. After that, **XGBoost** is refined and tuned through the change of its hyper-parameters to reach the best model out of it until we finish with a logloss score of 2.466 on the validation set.

Throughout the project, there're some interesting parts as **Data Cleansing** where corrupted and invalid rows are removed, and **Data Visualization** where the data is plotted according to its features individually against each other, such as plotting longitudes vs. latitudes, or plotting the crime category against the day of week and the month it happened in. On the other side, the project has some challenges. For example, the training of all the models took some amount of time and resources, even **SVC** took a huge amount of time that makes me unable to get its training and testing results.

## IMPROVEMENT

Certainly, each project can be improved and result in better performance. I think **SVC** can produce reasonable results and takes less amount of time if the parameter **C** has been passed to it with some smaller value than its default of 1 according to that link.

Also, in the part of visualization, I had plotted only the most committed crime against the time. I think if all crime categories are plotted the same way, then there's more information we can have about the distribution of the data.

# References


[1] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[2] https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel

[3] https://xgboost.readthedocs.io/en/latest/python/python_api.html

[4] https://www.quora.com/Why-does-XGBoost-perform-better-than-SVM

[5] https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

[6] http://forums.fast.ai/t/to-label-encode-or-one-hot-encode/6057

[7] https://datascience.stackexchange.com/questions/9443/when-to-use-one-hot-encoding-vs-labelencoder-vs-dictvectorizor

[8] https://plot.ly/matplotlib/bar-charts/

[9] https://www.google.com.eg/maps/place/San+Francisco,+CA,+USA/@37.7407396,-122.4303937,12z/data=!4m5!3m4!1s0x80859a6d00690021:0x4a501367f076adff!8m2!3d37.7749295!4d-122.4194155

[10] http://scikitlearn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[11] https://datascience.stackexchange.com/questions/26747/does-class-weight-solve-unbalanced-input-for-decision-tree