SAN FRANSISCO CRIME CLASSIFICATION

By

Yousef Mohamed Zook

April-2018

yousefzook@outlook.com

Definition:

Project Overview

Introduction and problem domain:

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.

The domain of the problem is *safety* domain where the model should help in minimizing the crimes in *San Francisco* based on the given data and understanding it.

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. This is a multi-class classification problem, given time and location, the goal is to predict the category of crime that occurred.

Similar problems:

This crime category prediction problem is similar to other problems like predicting the crime rate which is discussed here and here. Also, temperature prediction using machine learning, this problem uses the locations and some features like humidity and sunshine to train the model and make it predicts the right temperature at the right time. This problem is discussed more in this IEEE doc and this paper.

Data sets:

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.

The data reference: https://www.kaggle.com/c/sf-crime/data

Problem Statement

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. This is a multi-class classification problem, given time and location, the goal is to predict the category of crime that occurred.

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 output data should indicate the crime category for the given time and location. This can be solved using multi-class classification algorithms.

Metrics

It is noticed that the data is imbalanced as there some targets that isn't appear a lot – ex. TREA. So, we shouldn't use accuracy here due to imbalanced data, but we may use multi-class logarithmic loss or F1-score.

multi-class logarithmic loss is given by the next equation:

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

where N is the number of cases in the test set, M is 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.

Analysis:

Data Exploration

Data fields

1. **Dates** - timestamp of the crime incident
2. **Category** - category of the crime incident (only in train.csv). This is the target variable that is going to be predicted.
3. **Descript** - detailed description of the crime incident (only in train.csv)
4. **DayOfWeek** - the day of the week
5. **PdDistrict** - name of the Police Department District
6. **Resolution** - how the crime incident was resolved (only in train.csv)
7. **Address** - the approximate street address of the crime incident
8. X - Longitude
9. Y – Latitude

Categories occurrences:

LARCENY/THEFT	174885	DISORDERLY CONDUCT	4318
OTHER OFFENSES	126165	DRUNKENNESS	4280
NON-CRIMINAL	92300	RECOVERED VEHICLE	3138
ASSAULT	76872	KIDNAPPING	2341
DRUG/NARCOTIC	53971	DRIVING UNDER THE INFLUENCE	2268
VEHICLE THEFT	53772	RUNAWAY	1946
VANDALISM	44724	LIQUOR LAWS	1903
WARRANTS	42206	ARSON	1513
BURGLARY	36754	LOITERING	1225
SUSPICIOUS OCC	31412	EMBEZZLEMENT	1166
MISSING PERSON	25989	SUICIDE	508
ROBBERY	22999	FAMILY OFFENSES	491
FRAUD	16679	BAD CHECKS	406
FORGERY/COUNTERFEITING	10609	BRIBERY	289
SECONDARY CODES	9985	EXTORTION	256
WEAPON LAWS	8555	SEX OFFENSES NON FORCIBLE	148
PROSTITUTION	7484	GAMBLING	146
TRESPASS	7325	PORNOGRAPHY/OBSCENE MAT	22
STOLEN PROPERTY	4539	TREA	6
SEX OFFENSES FORCIBLE	4387		

Data shape

	Dates	Category	Descript	DayOfWeek	PdDistrict	Resolution	Address	X	Y
0	2015-05-13 23:53:00	WARRANTS	WARRANT ARREST	Wednesday	NORTHERN	ARREST, BOOKED	OAK ST / LAGUNA ST	-122.425892	37.774599
1	2015-05-13 23:53:00	OTHER OFFENSES	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	ARREST, BOOKED	OAK ST / LAGUNA ST	-122.425892	37.774599
2	2015-05-13 23:33:00	OTHER OFFENSES	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	ARREST, BOOKED	VANNESS AV / GREENWICH ST	-122.424363	37.800414
3	2015-05-13 23:30:00	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Wednesday	NORTHERN	NONE	1500 Block of LOMBARD ST	-122.426995	37.800873
4	2015-05-13 23:30:00	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Wednesday	PARK	NONE	100 Block of BRODERICK ST	-122.438738	37.771541

```
train shape: (878049, 9)
```

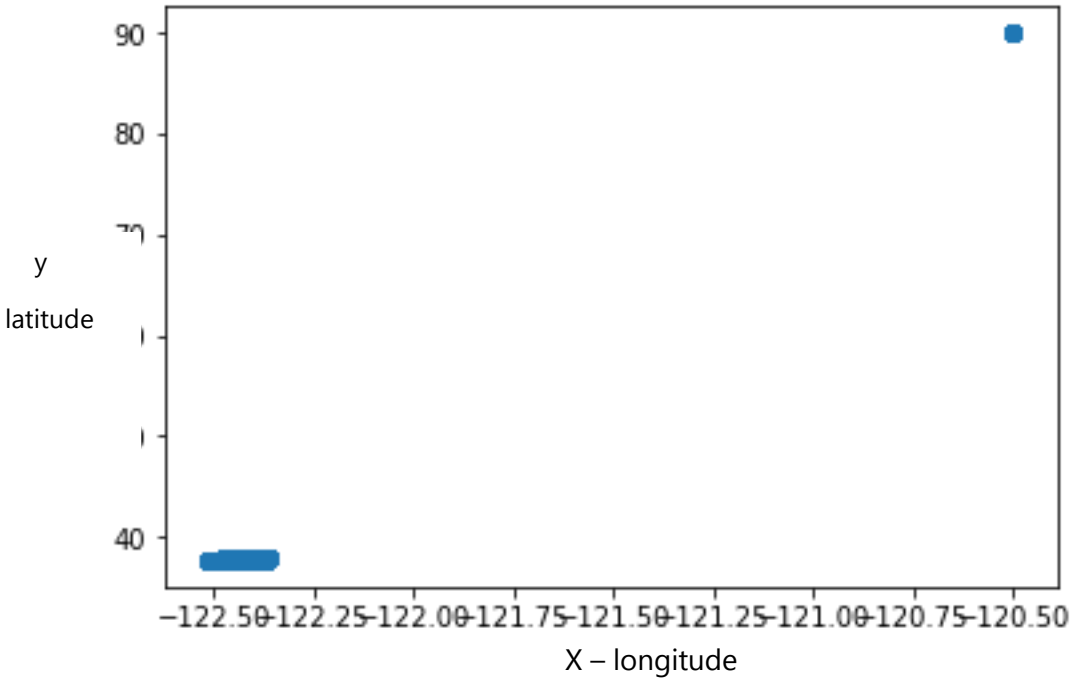
	Id	Dates	DayOfWeek	PdDistrict	Address	X	Y
0	0	2015-05-10 23:59:00	Sunday	BAYVIEW	2000 Block of THOMAS AV	-122.399588	37.735051
1	1	2015-05-10 23:51:00	Sunday	BAYVIEW	3RD ST / REVERE AV	-122.391523	37.732432
2	2	2015-05-10 23:50:00	Sunday	NORTHERN	2000 Block of GOUGH ST	-122.426002	37.792212
3	3	2015-05-10 23:45:00	Sunday	INGLESIDE	4700 Block of MISSION ST	-122.437394	37.721412
4	4	2015-05-10 23:45:00	Sunday	INGLESIDE	4700 Block of MISSION ST	-122.437394	37.721412

```
test shape: (884262, 7)
```

Longitude, latitude statistics:

```
Longitudes summary:
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 summary:
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
```



Observation:

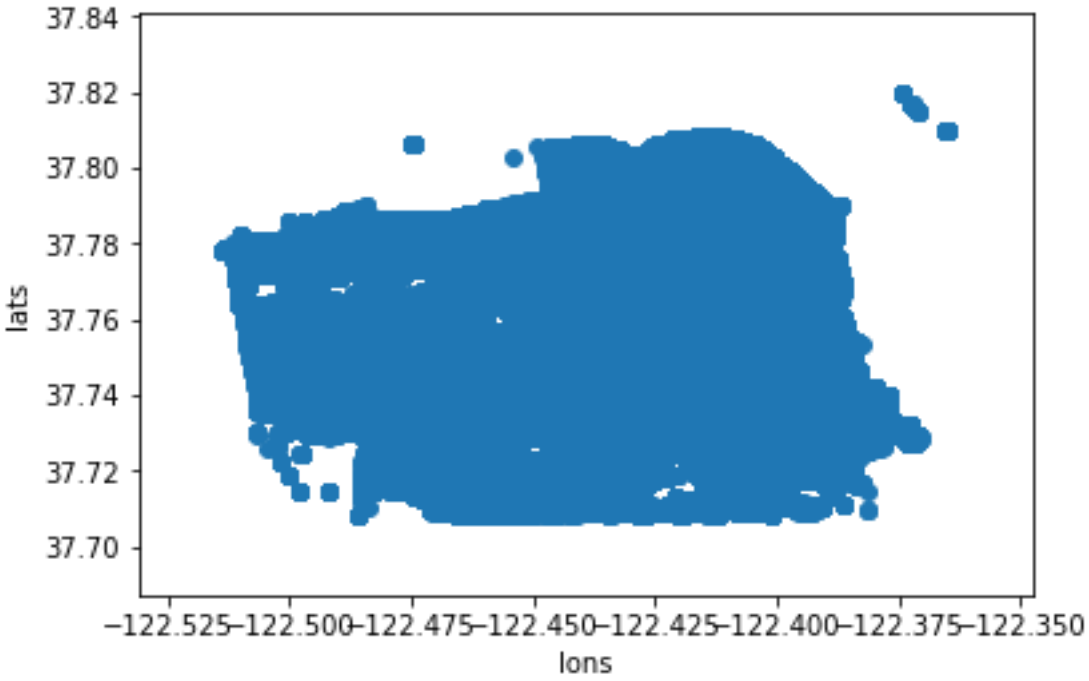
- longitudes are between [-122.52, -120.5], each value different slightly from others
- latitudes are between [37.708, 90]
- here as shown there exist some bad outlier values -i.e. close to 90-, the reasons that this is bad that
  - first, San Fransisco latitudes are between [37.707, 37.83269] , reference: google maps
  - Second, as shown in the statistics that the most values are close to 37.7
  - Also, in longitudes, San Francisco longitudes are between [-122.517652, -122.3275], from google maps

Observation:

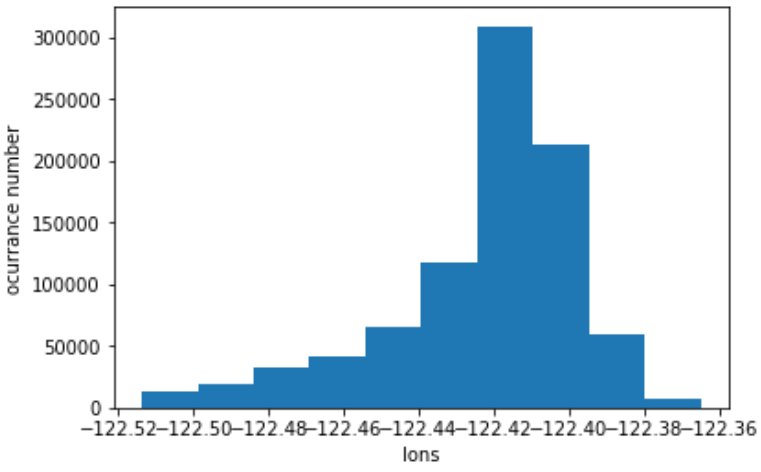
As shown above, there exist 2 columns -features- that are considered as redundancy. `Descript` and `Resolution` are these 2 columns as they don't exist in the testing values and also not a label required from the models, so they should be removed.

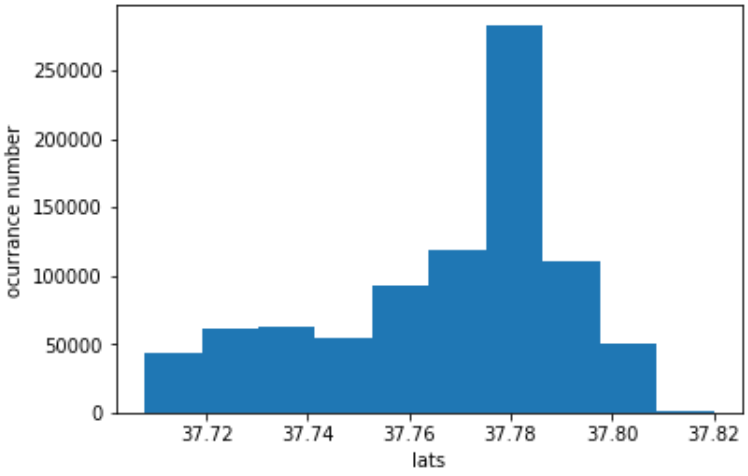
Exploratory Visualization

After removing the outliers, the samples will be distributed on the map as follow



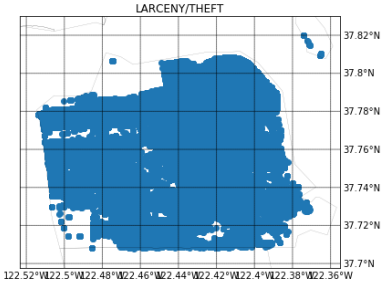
Plotting the crimes occurrences according to longitudes and latitudes



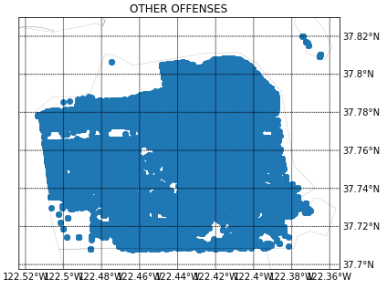


Plotting each crime occurrences according to locations:

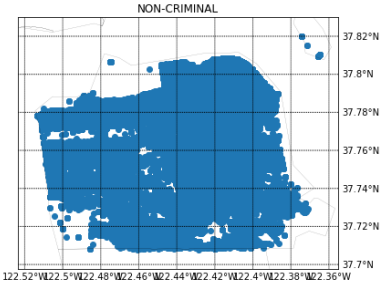
LARCENY/THEFT



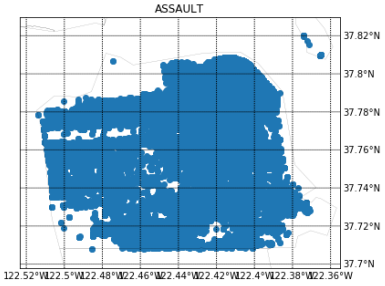
OTHER OFFENSES



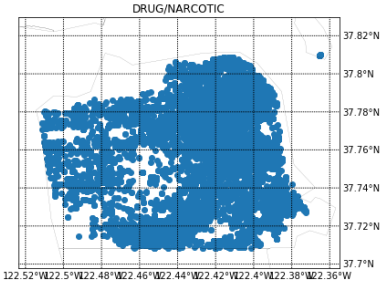
NON-CRIMINAL



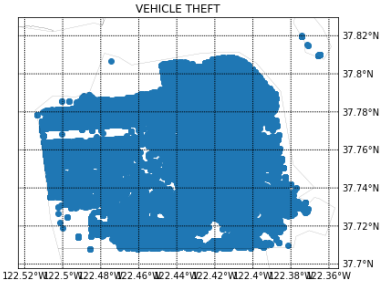
ASSAULT



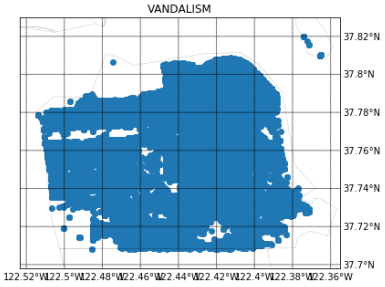
DRUG/NARCOTIC



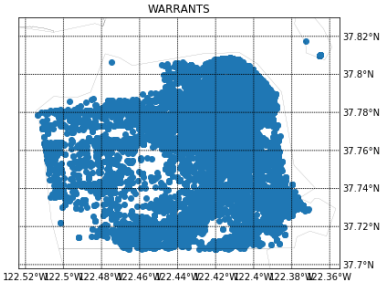
VEHICLE THEFT



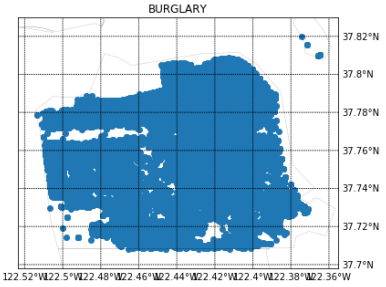
VANDALISM



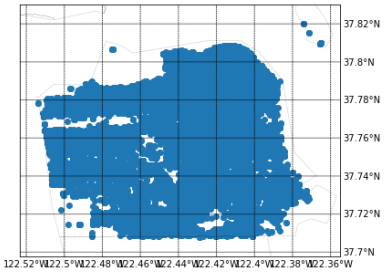
WARRANTS



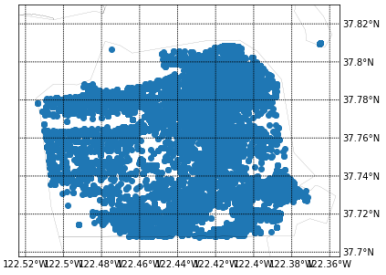
BURGLARY



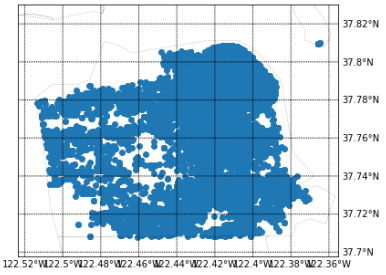
SUSPICIOUS OCC



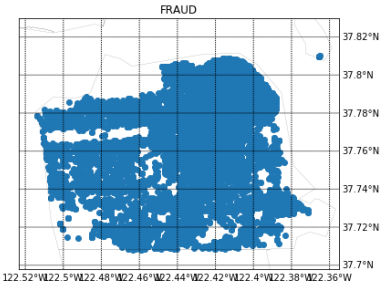
MISSING PERSON



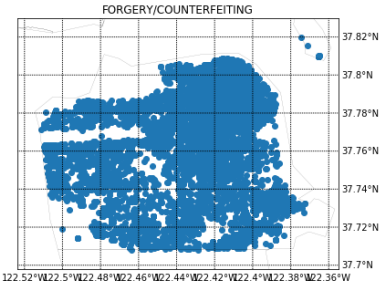
ROBBERY



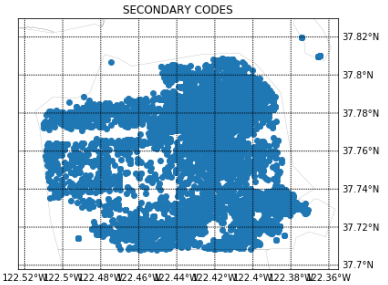
FRAUD



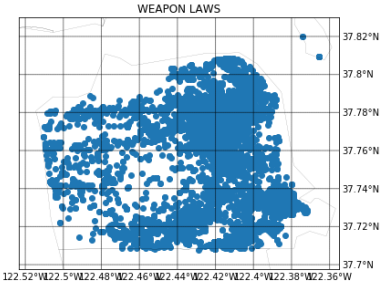
FORGERY/COUNTERFEITING



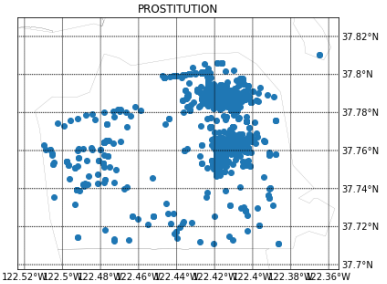
SECONDARY CODES



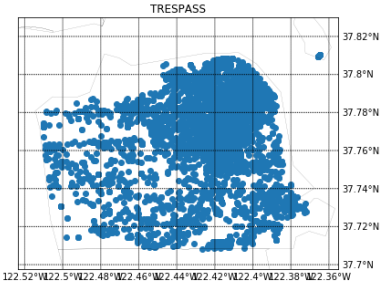
WEAPON LAWS



PROSTITUTION

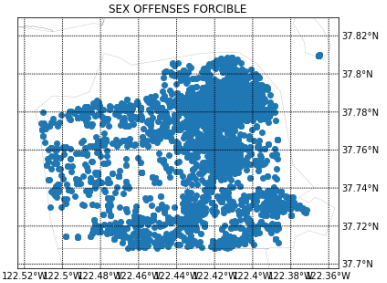


TRESPASS

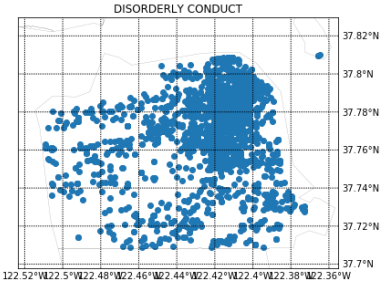


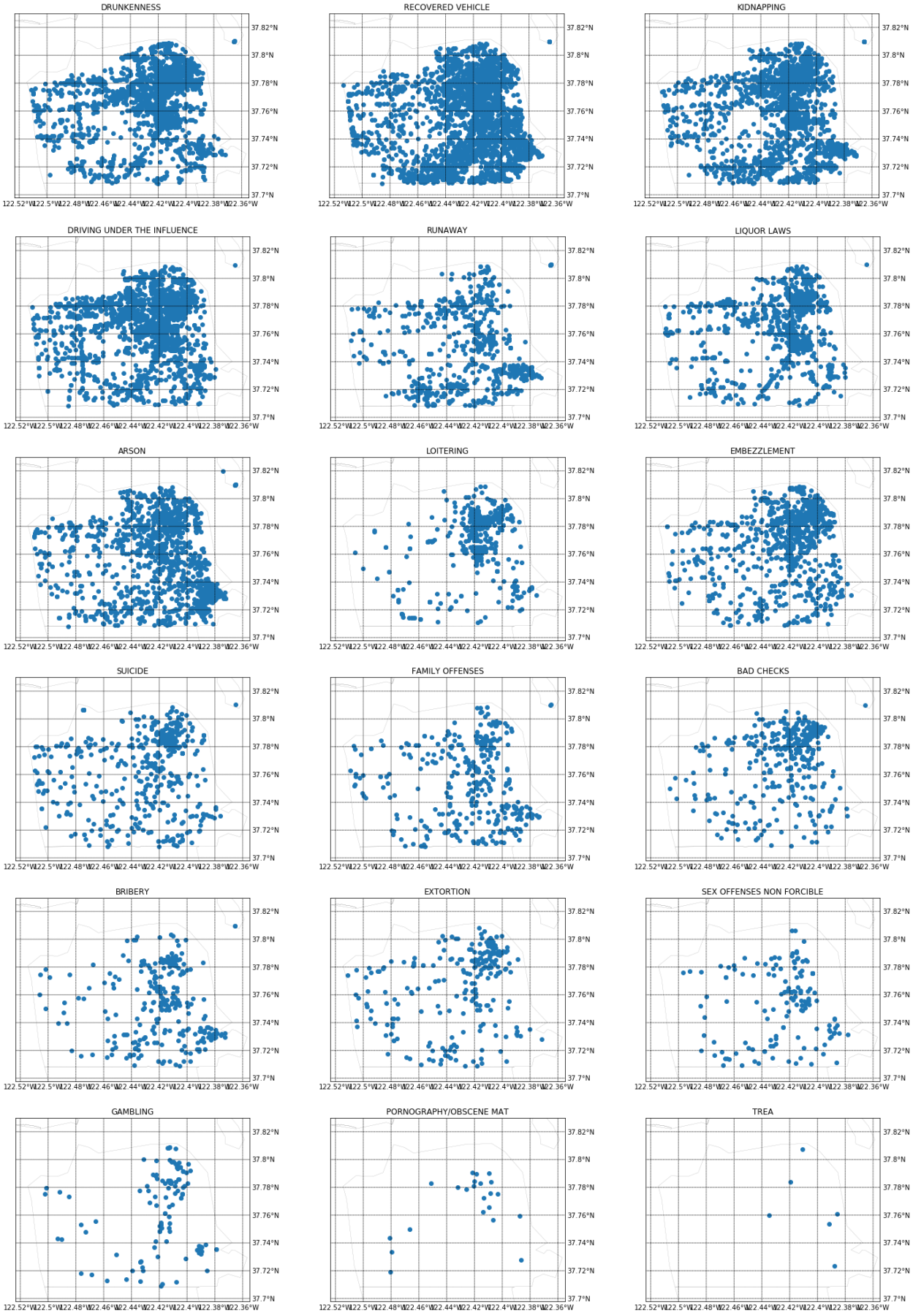
STOLEN PROPERTY

SEX OFFENSES FORCIBLE



DISORDERLY CONDUCT

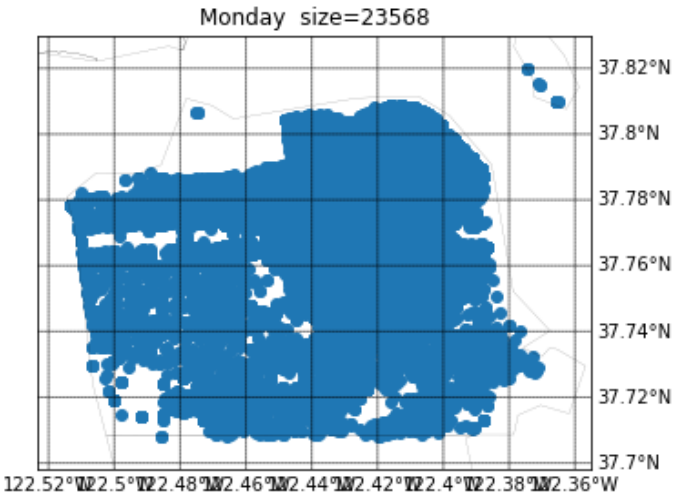


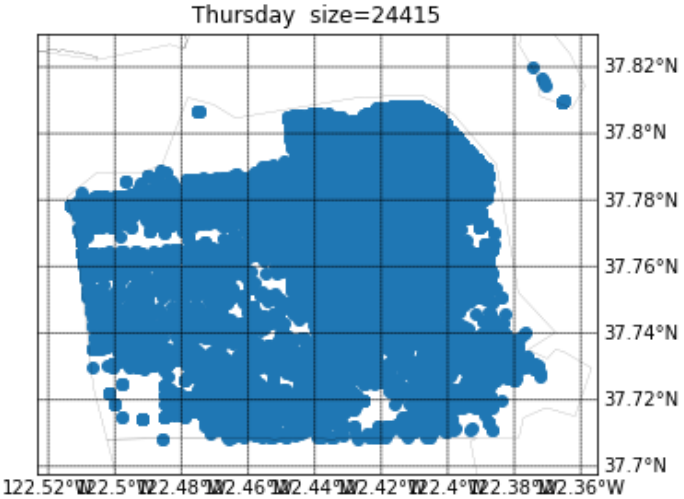


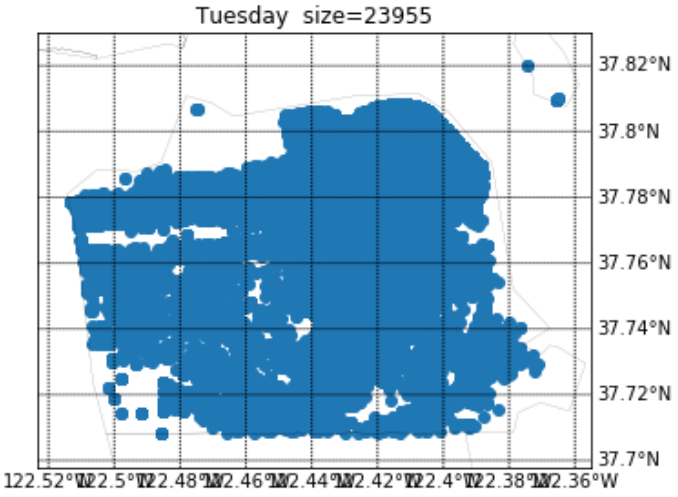
Locations of each crime

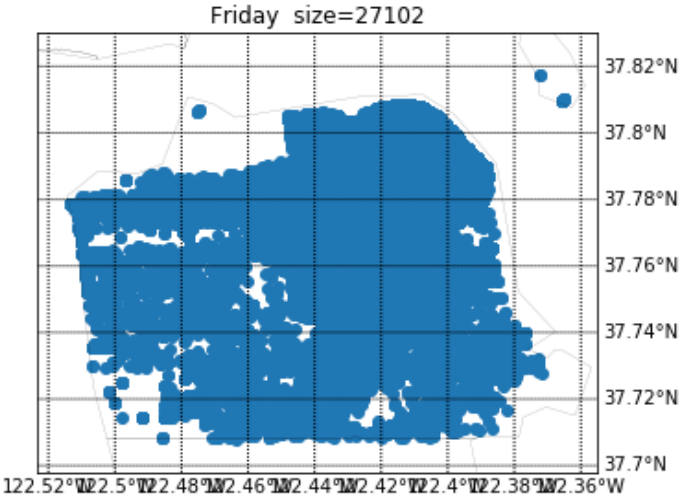
Observation: The maps above till us that there exist some crimes that is focused on certain locations, for example `prostitution` is focused mostly in (37.79N, -122.41W) and (37.76N, -122.41W). Other crimes are spread into the map like `Driving under the influence`.

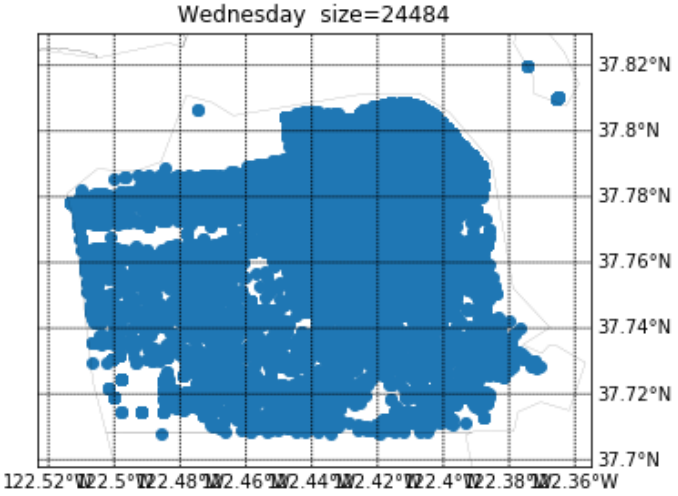
Plotting according to days of week:

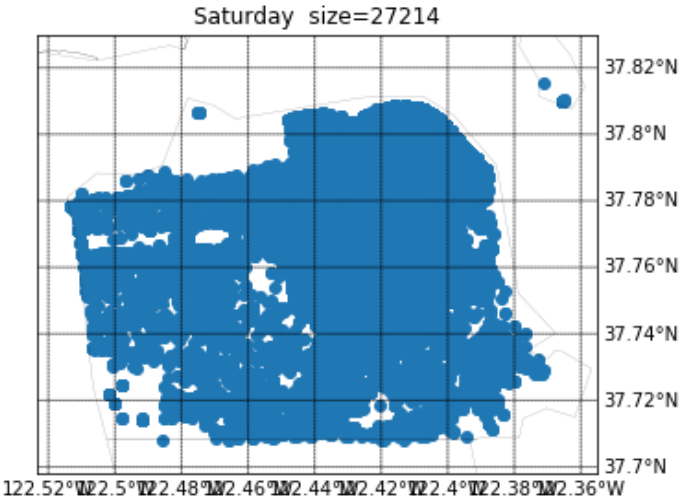


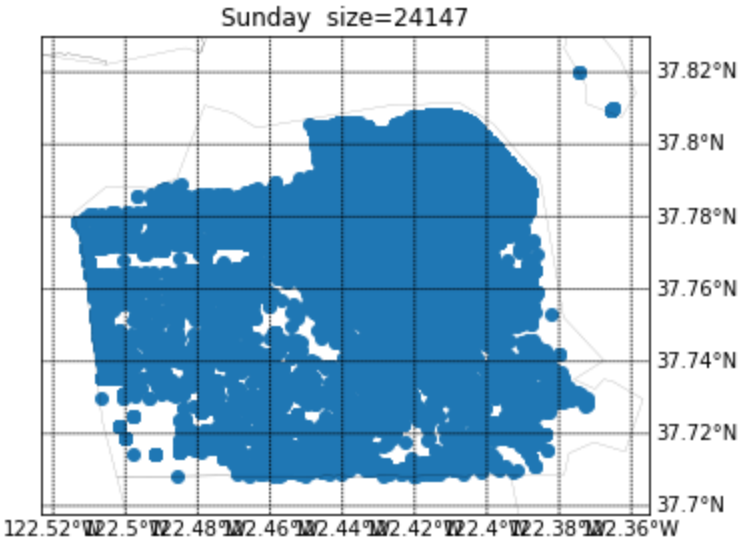












Number of occurrences for each day

Observation: As shown in plots above the crime pattern is likely to be equally distributed on months and days of week. Some months like `April` and `May` have more crime rate than others like `December` and `January`, and logically this may be due to the weather conditions that is good at spring and bad in winter. Also, days that before the week end is more than other days a little.

Algorithms and Techniques

Techniques used at data preprocessing are *Label Encoding* and *One-hot Encoding*

Label Encoding used for features with large different values as using One-hot Encoding for these features will produce large number of dimensions, One-hot Encoding used for other features.

This is a supervised learning classification problem. The following algorithms are used and compared to each other:

1. KNN – this algorithm
2. Decision Trees
3. ExtraTree Classifier
4. Neural networks - MLPClassifier
5. SVM
6. XGboost

Each algorithm will be compared using metrics and training, testing time.

Benchmark

The models and final results will be compared according to 3 categories

Random Threshold Score:

The results are compared to the random model score which produce:

F1-score = 0.027 log-loss = 34.12

The models should do better than this

Self-implemented models Score:

Score of each model is compared to other models to obtain the best from them

Kaggle scores

Some scores from Kaggle platform which indicate how our model is good

Examples: LiblineaR(score=2.5921) EDA and classification, score = 2.56180

Methodology:

Data Preprocessing

Removing redundant features and outliers

As mentioned above, there exist some samples that are considered as outliers, these samples are removed by checking the location of each point and if it not in San Francisco range, it is removed.

Also, there exist 2 columns – features – that are considered as redundancy. *Descript* and *Resolution* are these 2 columns as they don't exist in the testing values and also not a label required from the models, so they are removed.

Imbalanced data

The data given are imbalanced as it contains categories with different frequencies. So, to recover from this problem, 2 techniques are used:

1. Replicating data with small frequency – lower than 1000
2. Put more weights on the data with small frequency by putting the class_weight parameter = *balanced* which mean that the weights will be automatically adjusted inversely proportional to class frequencies in the input data as

$$n\_samples / (n\_classes * np.bincount(y))$$

Data Encoding

4 features from the data given are not numbers, so we need to convert them into numbers in order to be able to train the models on them. These features are:

- Dates
- DayOfWeek
- PdDistrict
- Address

We can use *Label Encoding* and *One-hot Encoding*.

One-hot Encoding may produce very high numbers of dimensions due to the many data labels in each feature, but it is better because that "the problem with *Label Encoding* is that it assumes higher the categorical value, better the category which produce more errors."

To see what of the features can be One-hot encoded, let's see the size of unique values for each of them:

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

Observation: *DayOfWeek* and *PdDistrict* can one-hot encoded. But *Address*, *Dates* should be encoded using label encoding.

Implementation

Training and testing pipeline

As the problem is solved using more than 1 model, we need a pipeline for training and testing such that each model go through it. This pipeline is implemented as 2 functions

```
def train_test_pipeline(learner, X_train, y_train, X_test, y_test)
```

where:

learner: is the model to train and test

X_train: is the data to train on

y_train: the validation set of training data

X_test: is the data to test on

y_test: the true value for test data

And the next function which call the previous one for given models

```
def train_test_models(models, names=None, n=len(y_train), m=len(y_test))
```

where:

models: is a list of models to train and test

names: the name of each given model, if None the model.class name will be taken

n: size of training data to use, default = all train data

m: size of testing data to use, default = all test data

This pipeline make it very easy to train and test many number of models and returns the results for each one. The results returned are:

train_time: the time required for trainig

test_time: the time required for testing

fscore_train: fscore for predicting training data

fscore_test: fscore for predicting testing data

logloss_train: logloss for training data

logloss_test: logloss for training data

Visualizing results

After getting the results of models training and testing, They are visualized by using

```
def visualize(results, random_results)
```

which take the results and random model results and visualizing them.

Random model Implementation

The random model is implemented as a python class which has 4 functions:

__getRandomValue(): returns always a random value from the categories

fit(X_train, y_train): which do nothing as the random model doesn't need to fit the data

predict(X): which predict the results for given list randomly

predict_proba(X): like predict but returns lists of probabilities for each class, the predicted class probability = 1, others = 0

Implementing other models

All previous 6 models are implemented as follow:

```
model_KNN = KNeighborsClassifier(n_jobs=-1, weights='distance')
model_tree = DecisionTreeClassifier(class_weight='balanced')
model_extraTrees = ExtraTreesClassifier(n_jobs=-1, class_weight='balanced'
)
model_NN = MLPClassifier(learning_rate='invscaling', shuffle=True)
model_SVC = SVC(probability=True, class_weight='balanced') # One-to-One
model_XGB = XGBClassifier(one_drop=1)
```

n_jobs means that the classifier should use all available CPU cores.

learning_rate='invscaling' is used to make the learning rate speed up with time

class_weight='balanced' is used as mentioned above due to imbalanced data

Refinement

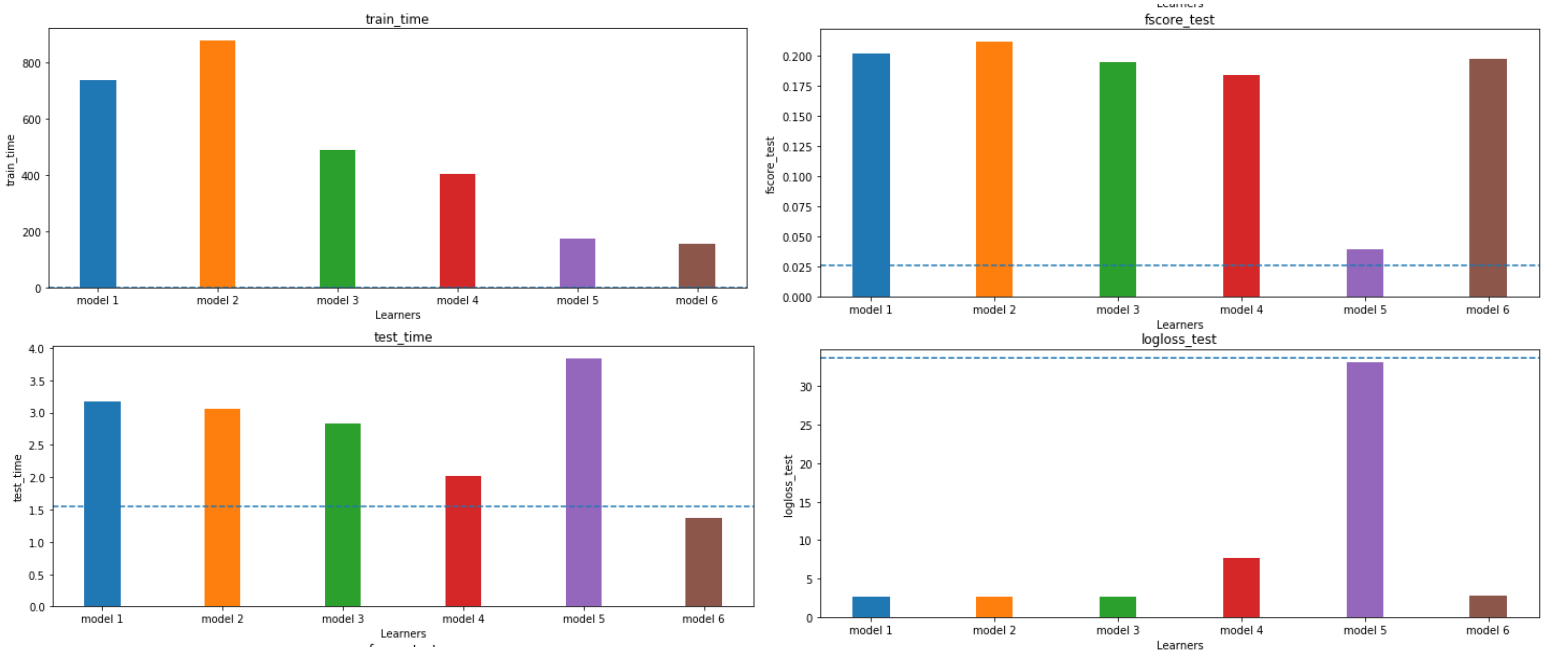
After showing results, the best model to use is MLPClassifier as will be discussed in results section. This model is refined by changing some of its' parameters and observing results using this code:

```
model_NN_tuned1 = MLPClassifier(learning_rate='invscaling', shuffle=True,
learning_rate_init=0.001,
                                hidden_layer_sizes=100, solver='adam')
model_NN_tuned2 = MLPClassifier(learning_rate='adaptive', shuffle=True, le
arning_rate_init=0.001,
                                hidden_layer_sizes=100, solver='adam')
model_NN_tuned3 = MLPClassifier(learning_rate='adaptive', shuffle=True, le
arning_rate_init=0.001,
                                hidden_layer_sizes=50, solver='adam')
model_NN_tuned4 = MLPClassifier(learning_rate='adaptive', shuffle=True, le
arning_rate_init=0.0001,
```

```
                                hidden_layer_sizes=200, solver='adam')
model_NN_tuned5 = MLPClassifier(learning_rate='adaptive', shuffle=True, le
arning_rate_init=0.001,
                                hidden_layer_sizes=100, solver='lbfgs')
model_NN_tuned6 = MLPClassifier(learning_rate='adaptive', shuffle=True, le
arning_rate_init=0.001,
                                hidden_layer_sizes=100, solver='sgd')

models = [model_NN_tuned1, model_NN_tuned2,model_NN_tuned3,model_NN_tuned4
,model_NN_tuned5,model_NN_tuned6]
names = ["model 1", "model 2", "model 3", "model 4", "model 5", "model 6"]
train_test_models(models, names=names)
```

This code returns the following results:



This mean that the best parameters to use are those that give the minimum logloss – i.e. model number 2. So, the final model used has paramaters:

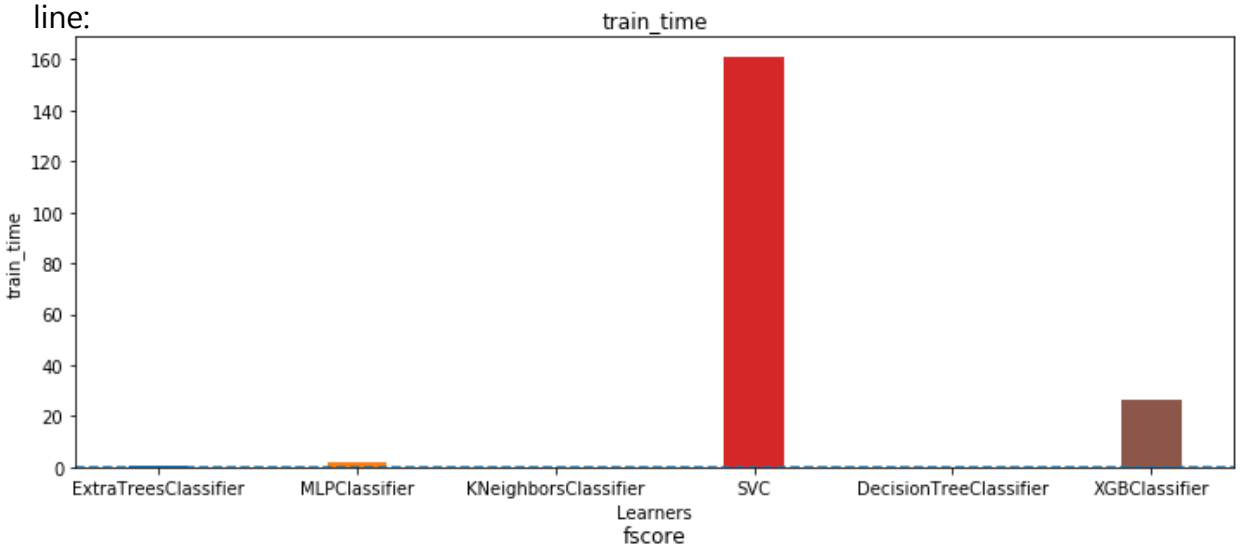
Learning_rate = adaptive epsilon = 1e-8 hidden_layer_sizes=100 solver=adam

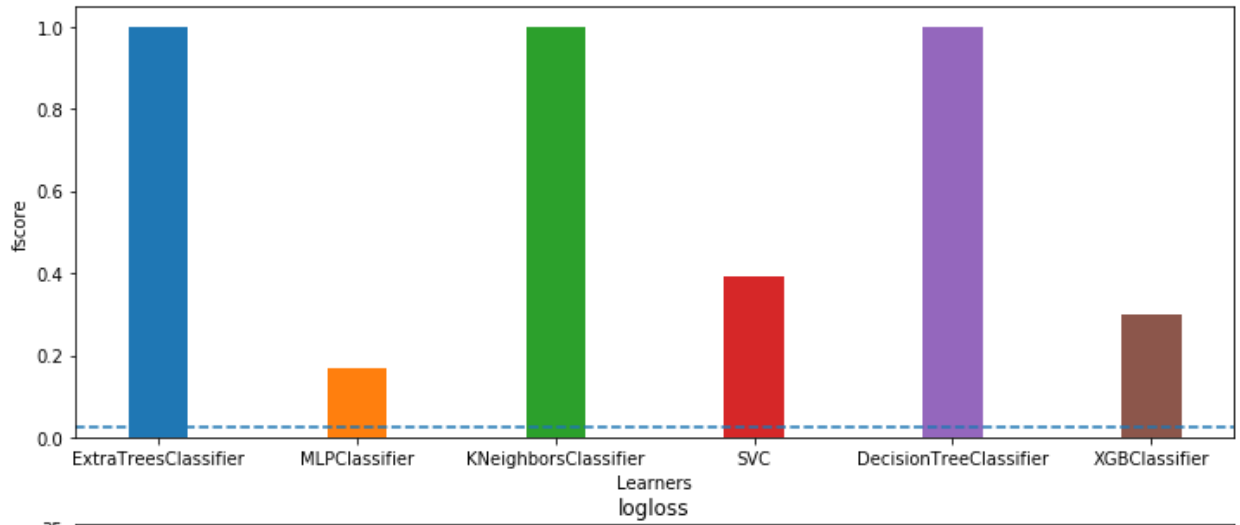
```
model_NN_tuned = MLPClassifier(learning_rate='adaptive', shuffle=True, eps
ilon=1e-8, activation='relu',hidden_layer_sizes=100, solver='adam', verbos
e=True)
```

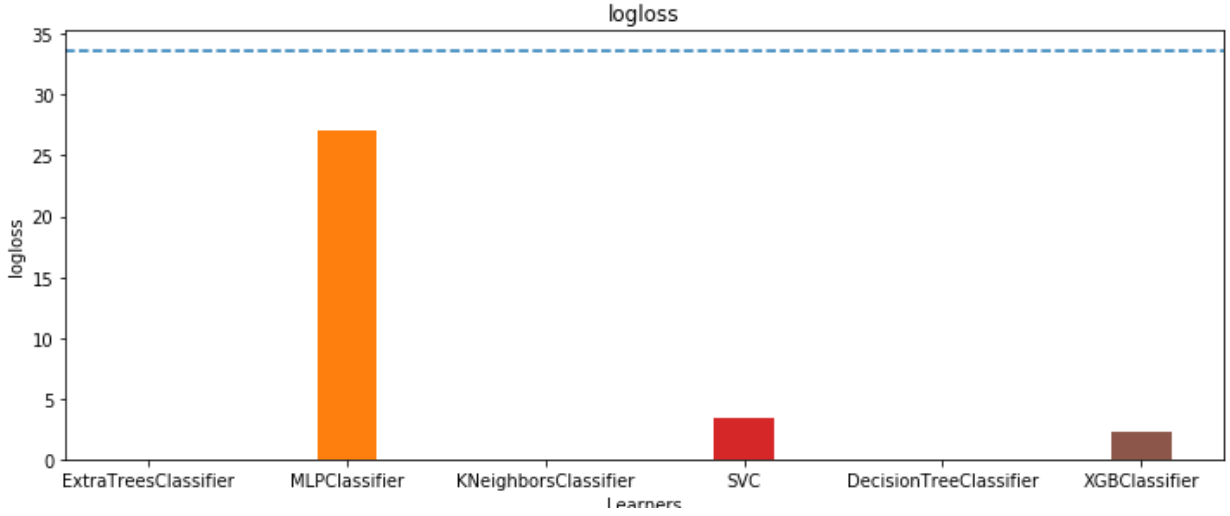
Results:

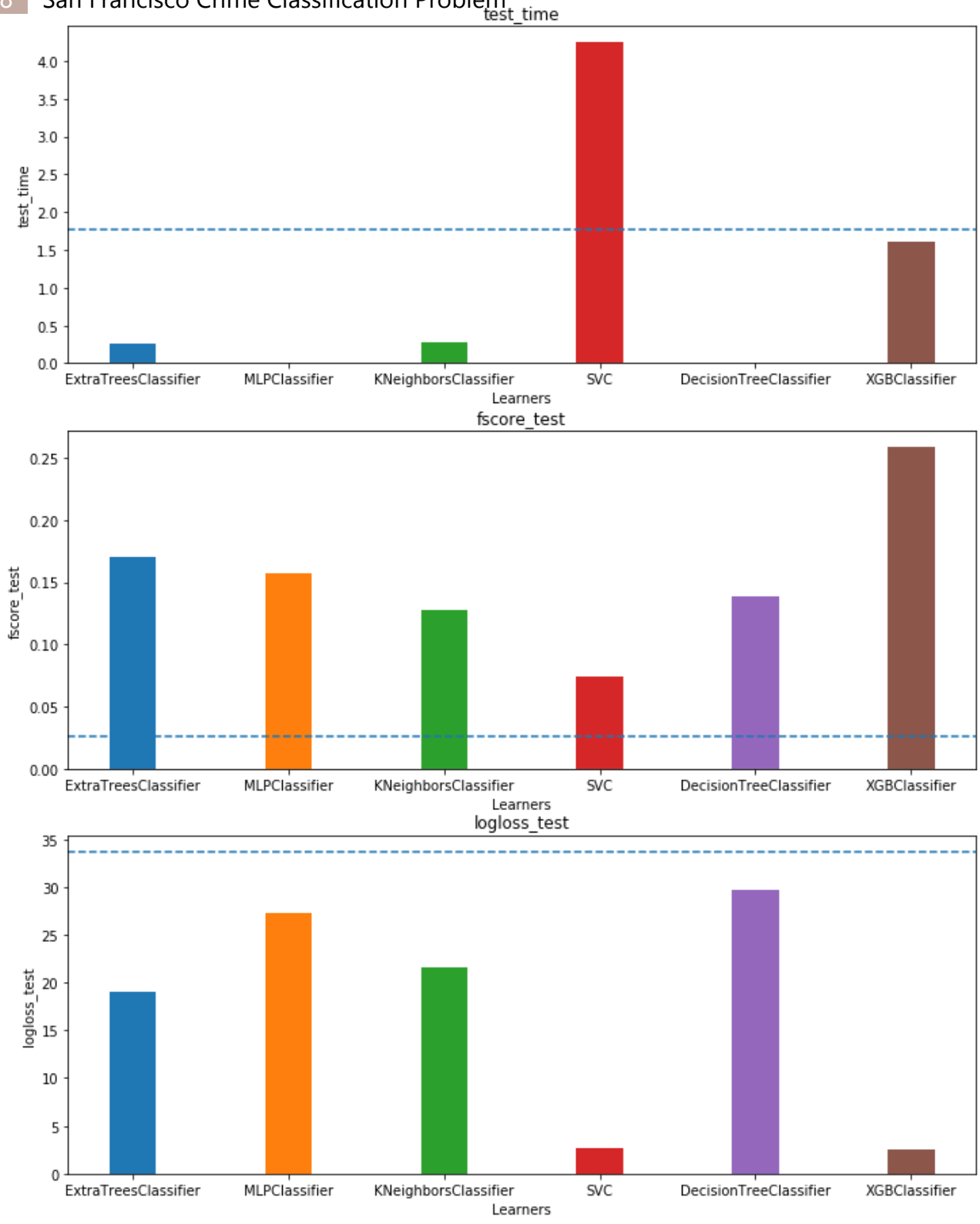
Model Evaluation and validation

First, all models are trained on 10000 samples for training and 2000 samples for testing which give the following results – random model results are expressed by blue dashed line:









Observations:

- SVC takes huge amount of time in training so It would be excluded while training on all data.

- ExtraTree take huge amount of memory which make the PC breakdown, so it would be excluded also.
- Trees and KNN are the best for predicting the training data
- SVC and XGB are the best for predicting the testing data

After excluding SVC due to it's high training time, XGB and others are trained on all data giving the following results:

```
model:          MLPClassifier
fscore:         0.200578658782
logloss:        2.6918295403
train time:     511.115532875
fscore_test:    0.2
logloss_test:   2.693622951
test_time:      3.00132

model: KNeighborsClassifier
fscore:         0.873238442702
logloss:        0.217253267588
train time:     6.08392095566
fscore_test:    0.236201
logloss_test:   22.32
test_time:      4.802236

model: DecisionTreeClassifier
fscore:         0.865415258244
logloss:        0.282391200185
train time:     23.2026150227
fscore_test:    0.236201
logloss_test:   26.5478
test_time:      0.422

model:          XGBClassifier
fscore:         0.257541264722
logloss:        2.48690374277
train time:     3017.42051196
fscore_test:    0.26241105
logloss_test:   2.6825
test_time:      197.5561
```

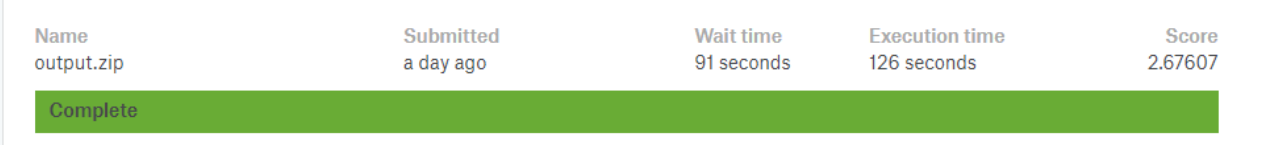
Observations:

- XGB and MLP are almost the same for testing and training data
- MLP is almost 6 times faster than XGB
- KNeighbors and DecisionTree are very good for training data but the worst for the testing data
- DecisionTree is the fastest in terms of testing time
- KNeighbors is the fastest in terms of training time

After tuning MLP classifier it gives the final results:

```
model: MLPClassifier
fscore:         0.209142808753
logloss:        2.66669112374
train time:     653.946779966
fscore_test:            0.208582203304
logloss_test:   2.66587393992
test time:      3.74381899834
```

Final Kaggle score is 2.67607



Justification

Random model

The final result is better than random model 12.75 times

Self-implemented models Score

The MLP final result compared to other models are shown in the next table

Model	Logloss Ratio = model / MLP
DecisionTree	9.977
KNN	8.39
XGB	1.008

As shown, MLP is better than KNN and DecisionTree a lot but almost equal to XGBoost classifier.

Kaggle scores

My score: 2.676

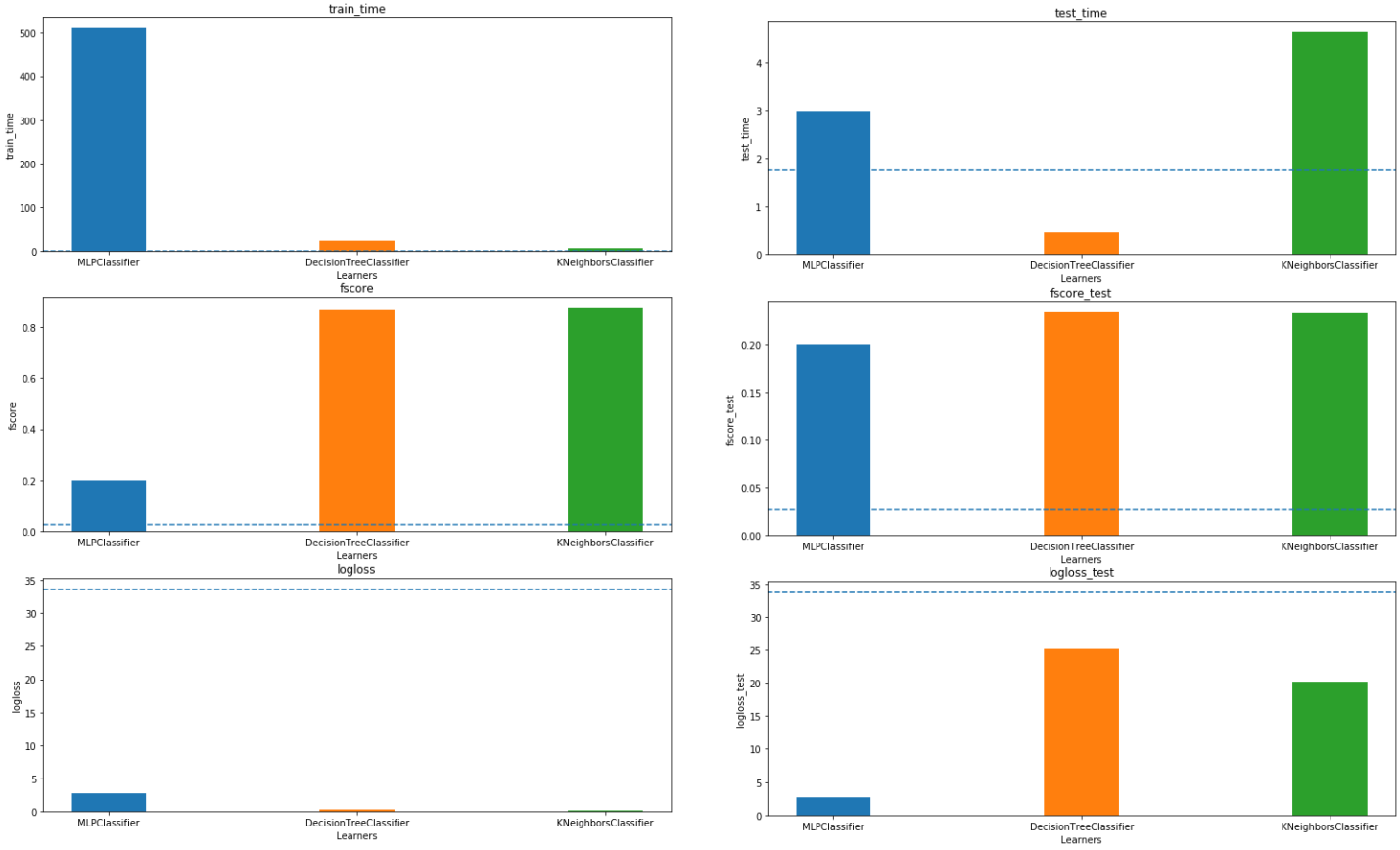
Almost equal to mentioned examples - i.e. 2.59 and 2.56

Conclusion:

Free-Form Visualization

The good quality of the project is represented in comparing various model with each other and observing the results. For example the following plot demonstrates the differences between MLPClassifier, KNneighbors and DecisionTreeClassifier.

The plot shows that the best for predicting training data is KNN and Decision Tree but they do very bad at testing data. On the other hand, MLP takes more time on training but gives good results on both training and testing data.



Reflection

This project demonstrates the differences in 6 models while handling data.

By exploring and visualizing the data we discovered the outliers in lons and lots, and that the data is imbalanced which lead us to replicate some data and give it more weights in training.

The crimes are distributed almost equally on days of weeks and months which is logical.

Also, the data was needed for some preprocessing such as Encoding features using 2 methods – label encoding and one-hot encoding – and also the need of removing outliers and unnecessary features.

After visualizing and encoding the data, we trained and tested our models using the train test pipeline and visualizing the results gave us important aspects of each model.

The results lead us to choose the most suitable model which was MLP for this problem as it has reasonable training time and amount of needed resources.

There exist some difficulties which were represented in visualizing huge amount of data and training this huge amount of data using heavy models such as ExtraTrees and SVC which lead to crash the device several times.

The interesting parts of the project were while training the last 4 models which seemed to be unexpected that KNN do bad on the data, but this seems to be due to the imbalanced and non-normal distribution of crimes on locations. Also while tuning the MLP final models parameters which seem that it is already tuned giving the best results than any other changing in its parameters.

Improvement

Of-course there exist better solutions for the problem. The part that I think should be has more improvement is the implementation of SVC, If for example the library offers somehow the SVC to be able to run on all CPU cores like KNN, I think It would make it faster and better. Also, the visualization of data may be better if we preprocessed the data first and if we used contour plots on all data plots for example.

References:

https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f

https://www.kaggle.com/c/sf-crime/

https://www.kaggle.com/rudikruger/liblinear/code

https://www.kaggle.com/vivekyadav/sfo-rmd-kaggle

https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/

https://stats.stackexchange.com/questions/113301/multi-class-logarithmic-loss-function-per-class

https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/