San Francisco Crime Classification using sparkML

- S SACHIN KUMAR (PES1PG21CS032)
- M MUKESH KUMAR (PES1PG21CS022)

Overview

This simulates a real world scenario where we can handle an enormous amount of data for predictive modelling. The data source is a stream and your application faces the constraint of only being able to handle batches of a stream at any given point in time.

Goals

Analyse large data streams and process them for machine learning tasks using Spark Streaming and Spark ML Lib to draw insights and deploy models for predictive tasks

Scope

We are interested in building a system that could classify crime discription into different categories. We create a system that could automatically assign a described crime to category which could help law enforcements to assign right officers to crime or could automatically assign officers to crime based on the classification.

We are using dataset from Kaggle on San Francisco Crime. Our responsibilty here is to train a model based on 39 pre-defined categories, test the model accuracy and deploy it into production. Given a new crime description, the system should assign it to one of 39 categories.

To solve this problem, we use a variety of feature extraction techniques along with different supervised machine learning algorithms in Pyspark.

Software/Languages to be used:

- Python
- Spark

Libraries Used:

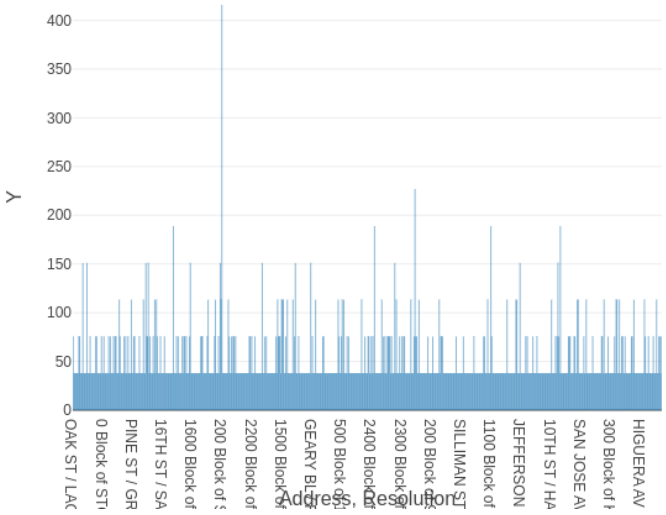
- Pyspark
- Numpy

Data fields

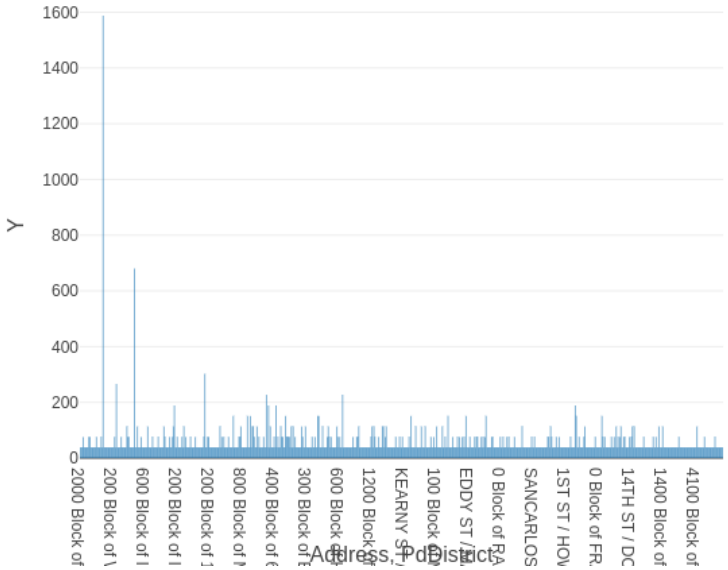
1. Dates - timestamp of the crime incident
2. Category - category of the crime incident (only in train.csv). This is the target variable that is going to be predicted.
3. Descript - detailed description of the crime incident (only in train.csv)

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

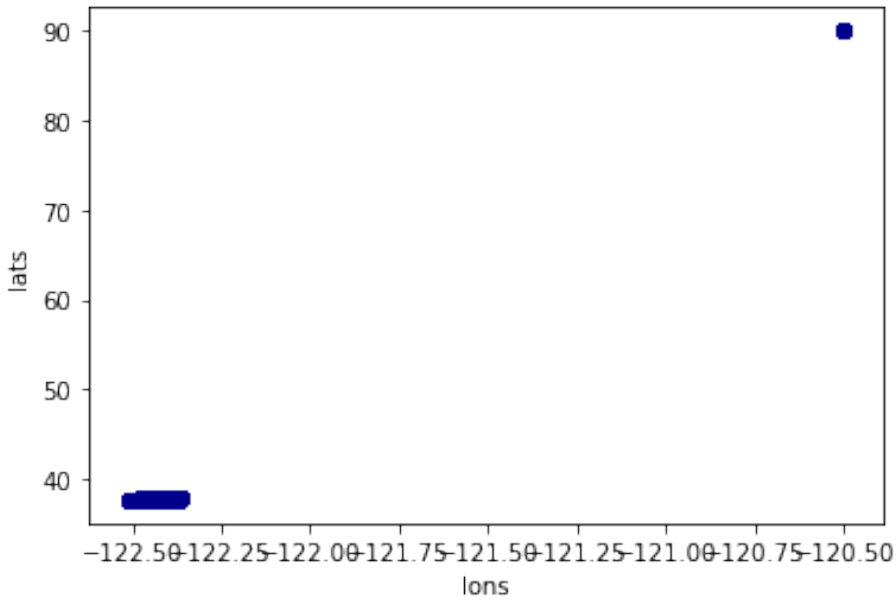
Visualisation of the Train Dataset



Visualisation of the Test Dataset



Longitude, latitude statistics:

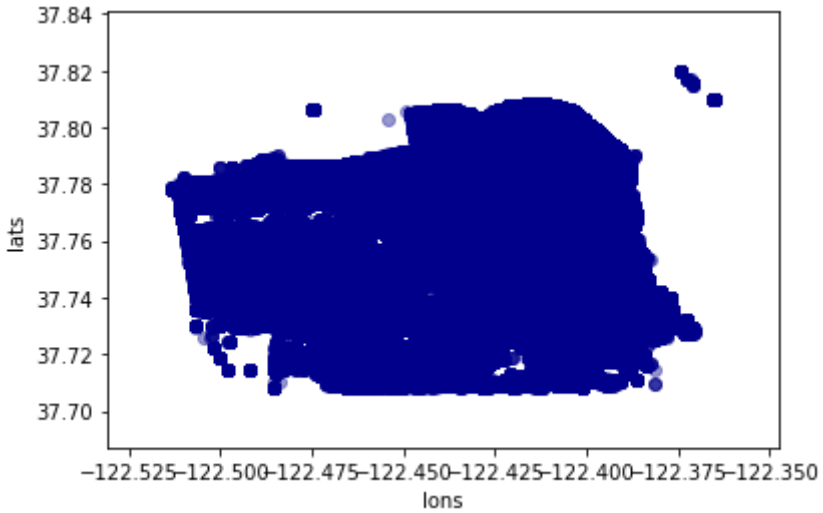


Obserivation:

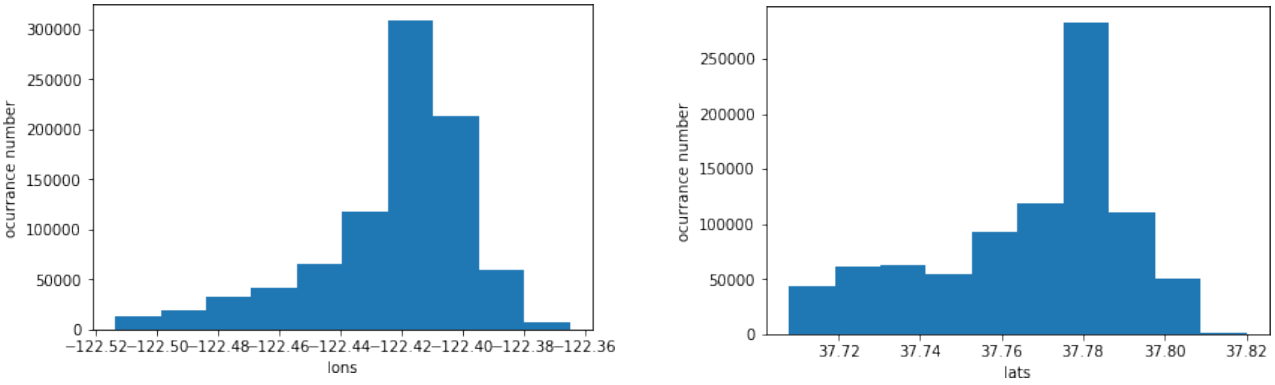
longitudes are between [-122.52, -120.5], each value different slightly from others
latitudes are between [37.708, 90] here as shown there exist some bad outlier values -i.e. close to 90-, the reasons that this is bad that:
- First, San Fransisco latitudes are between [37.707, 37.83269] ,reference: google 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

Exploratory Visualization

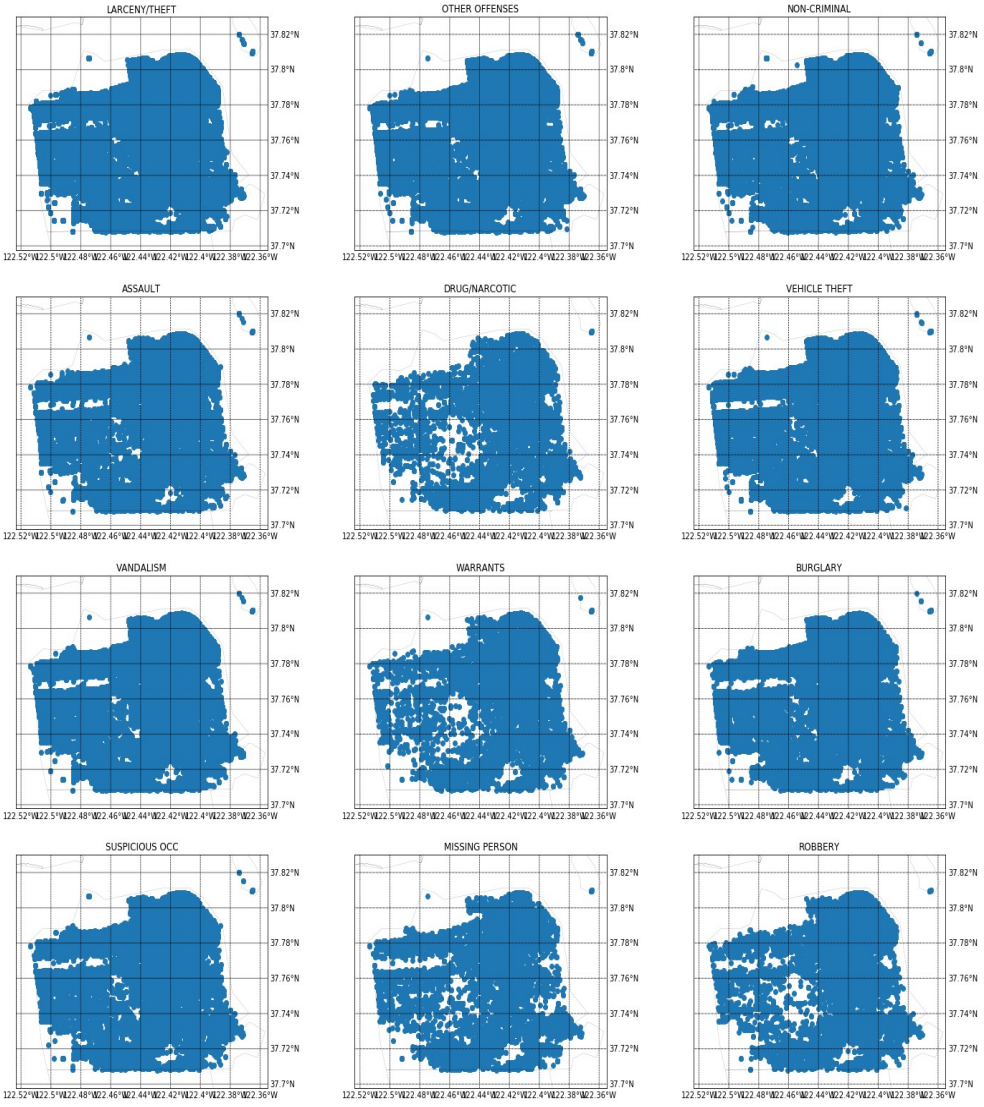
After removing the outliers, the samples will be distributed on the map as follow

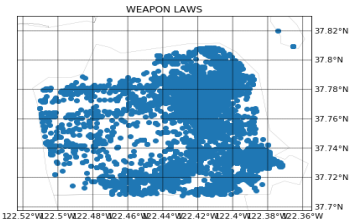


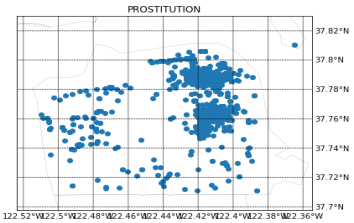
Plotting the crimes occurrences according to longitudes and latitudes

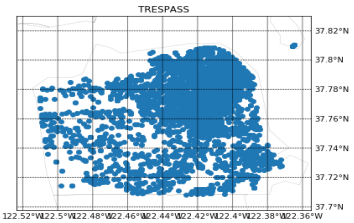


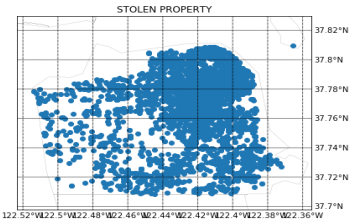
Plotting each crime occurrences according to locations:

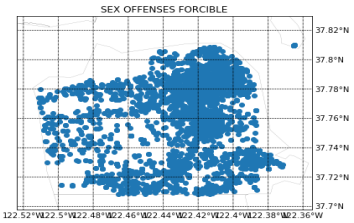


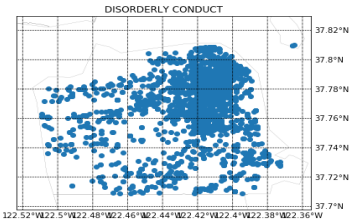

WEAPON LAWS

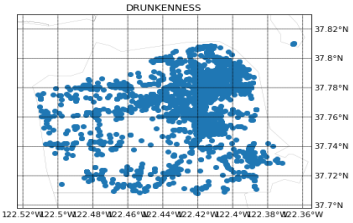

PROSTITUTION

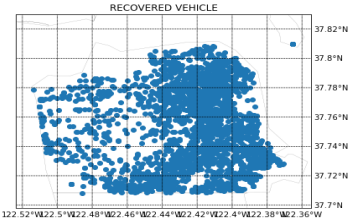

TRESPASS

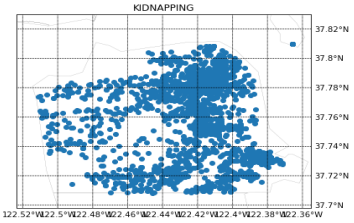

STOLEN PROPERTY

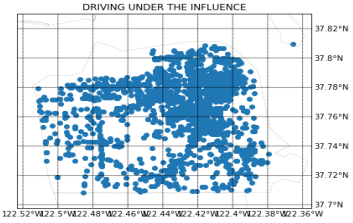

SEX OFFENSES FORCIBLE

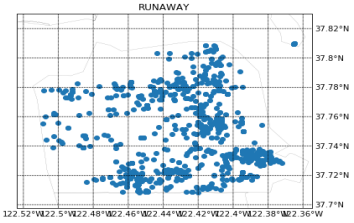

DISORDERLY CONDUCT

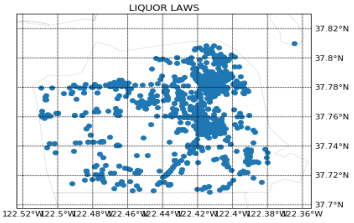

DRUNKENNESS

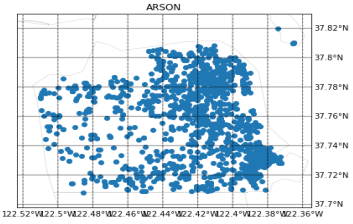

RECOVERED VEHICLE

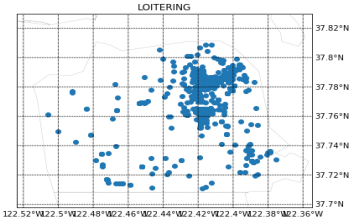

KIDNAPPING

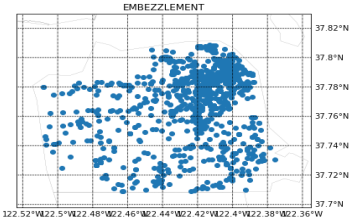

DRIVING UNDER THE INFLUENCE

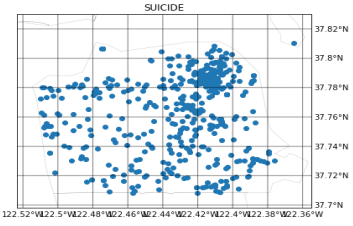

RUNAWAY

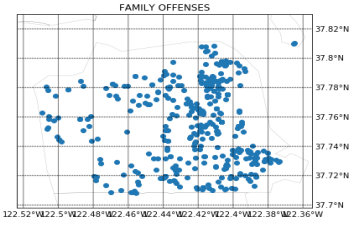

LIQUOR LAWS

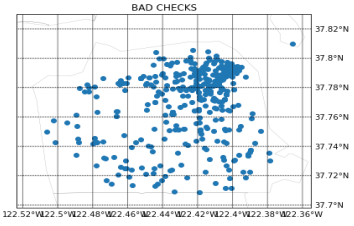

ARSON

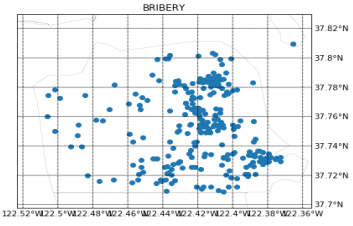

LOITERING

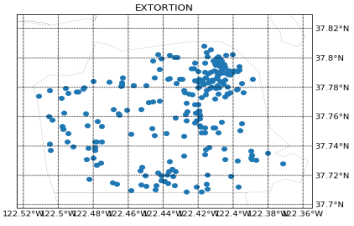

EMBEZZLEMENT

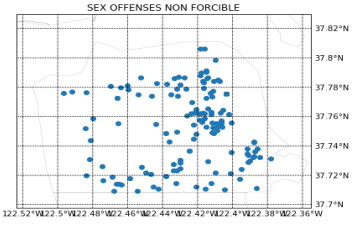

SUICIDE

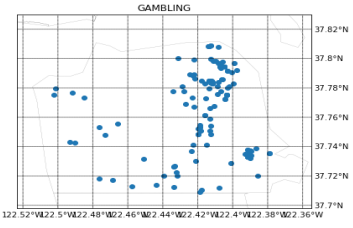

FAMILY OFFENSES

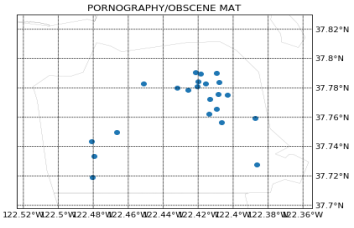

BAD CHECKS

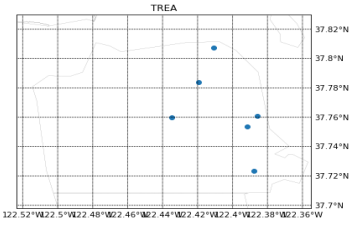

BRIBERY


EXTORTION


SEX OFFENSES NON FORCIBLE


GAMBLING


PORNOGRAPHY/OBSCENE MAT


TREA

Methodology:

1.Set up Spark and load Libraries
2.Data Extraction
3.Partition the dataset int train and test.
4. Define a structure to build a pipeline
5. Building a multi-classification

The Process of cleaning the dataset involves:

Tokenization using RegexTokenizer: RegexTokenizer allows more advanced tokenization based on regular expression (regex) matching. By default, the parameter "pattern" (regex, default: "\s+") is used as delimiters to split the input text.

Stop remover function using StopWordsRemover: StopWordsRemover takes as input a sequence of strings (e.g. the output of a Tokenizer) and drops all the stop words from the input sequences. The list of stopwords is specified by the stopWords parameter.

Estimator to extract the vocabulary using CountVectorizer: CountVectorizer can be used as an estimator to extract the vocabulary, and generates a CountVectorizerModel.

Encode the values of category variable using StringIndexer: StringIndexer encodes a string column of labels to a column of label indices.

Define a pipeline to call these functions: ML Pipelines provide a uniform set of high-level APIs built on top of DataFrames that help users create and tune practical machine learning pipelines.

Model training and evaluation

Build baseline model

-Logistic regression using CountVectorizer features

We build a model to make predictions and score on the test sets using logistics regression using the dataset we transformed using count vectors.

1. Models Implemented

a) Random Forest Using (Count Vector and TF-IDF)

Random forest is a commonly-used machine learning algorithm trademarked by Leo Breiman and Adele Cutler, which combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

b) Naive Bayes Using (Count Vector and TF-IDF)

Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. The spark.ml implementation currently supports both multinomial naive Bayes and Bernoulli naive Bayes.

c) Logistic Regression Using (Count Vector and TF-IDF)

Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1.

Results:

	Count Vector	TF-IDF
Random Forest	69.4 %	36.1 %
Logistic Regression	97.2 %	97.2 %
Naive Bayes	99.3 %	99.4 %

Conclusion:

From the above, we can say that TF-IDF proves to be best vectoriser for this dataset,but not so in the case of the Rand Forest, as Count Vector, is the preffered extraction method.

While Naive Bayes proves to be better algorithm for text analysis than Random Forest and Logistic regression.