San Francisco Crime Classification using sparkML
Advanced Big Data Analysis Project

-Deepak Purohit-PES1PG21CS012

NagaSuhas Shastry M-PES1PG21CS023

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 onlybeing 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 differentcategories. 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 is to train amodel 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 will use a variety of feature extraction techniques along withdifferent supervised machine learning algorithms in Pyspark.

Software/Languages to be used:

- Python
- Hadoop
- Spark

Libraries Used:

- Pyspark
- Numpy

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 thetarget variable that is going to be predicted.
3. Descript - detailed description of the crime incident (only in train.csv)

DayOfWeek - the day of the week
4.   PdDistrict - name of the Police Department District
5.   Resolution - how the crime incident was resolved (only in train.csv)
6.   Address - the approximate street address of the crime incident
7.   X - Longitude
8. Y – Latitude
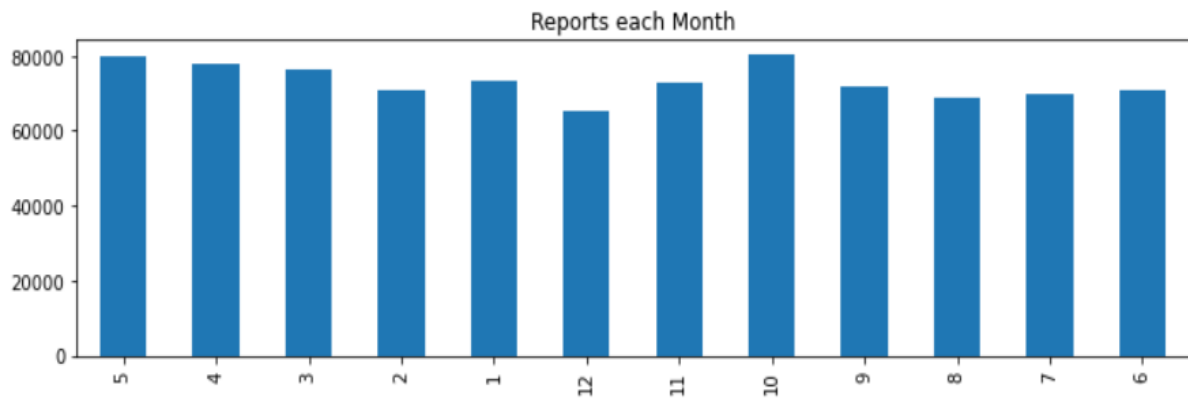
Exploratory Visualization:

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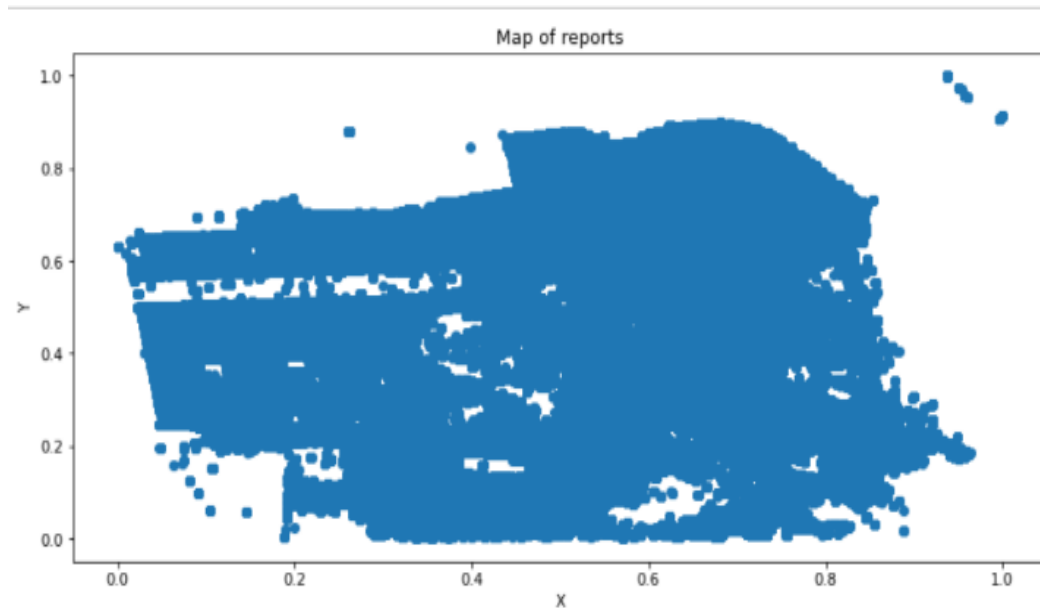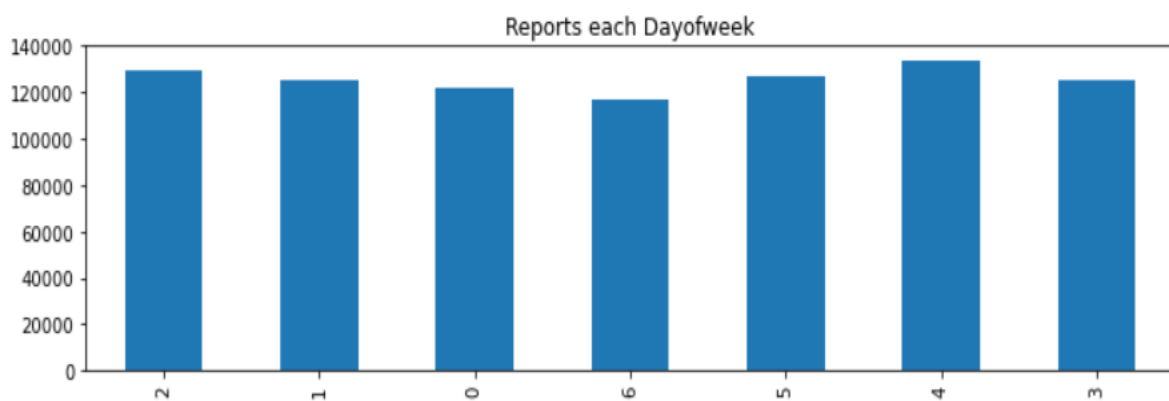

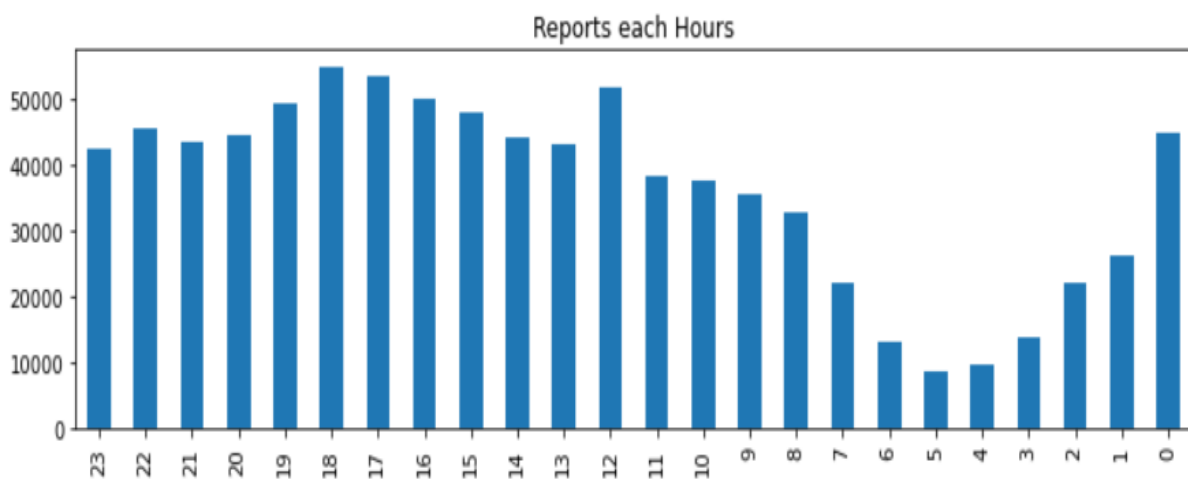Plotting the crimes occurrences according to the cases each year,
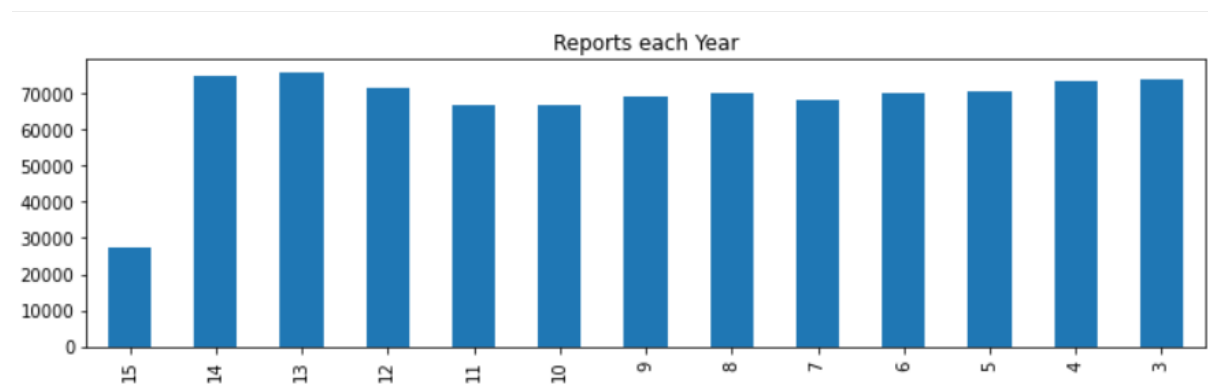
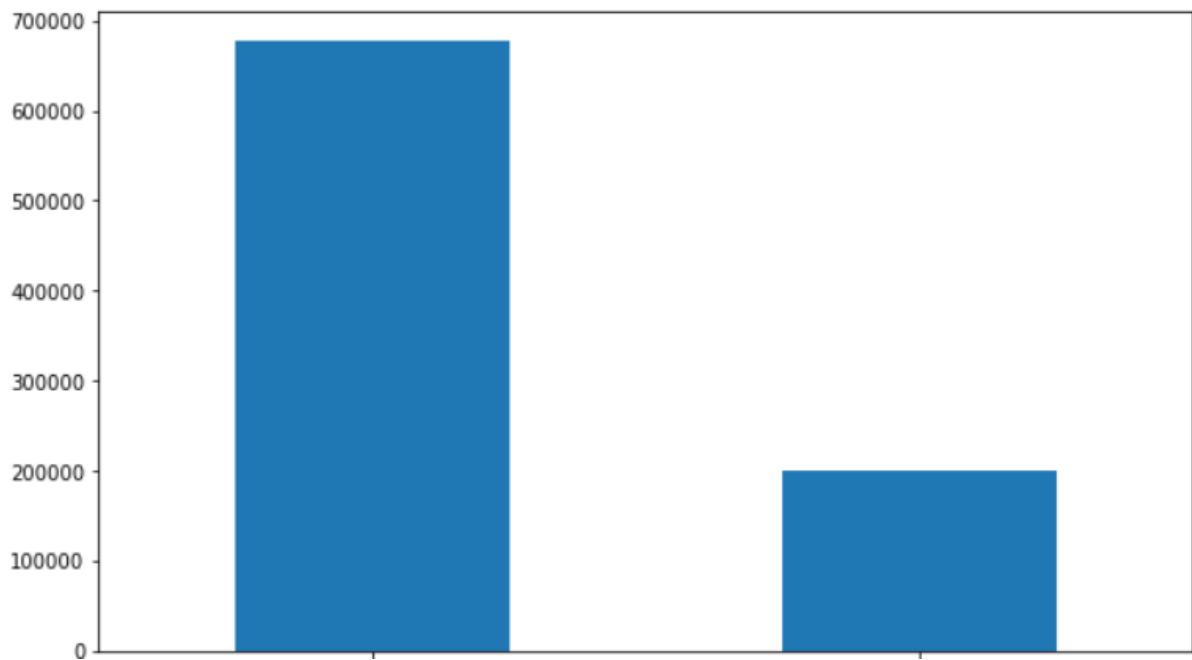Plotting the crimes occurrences according to the cases each month,

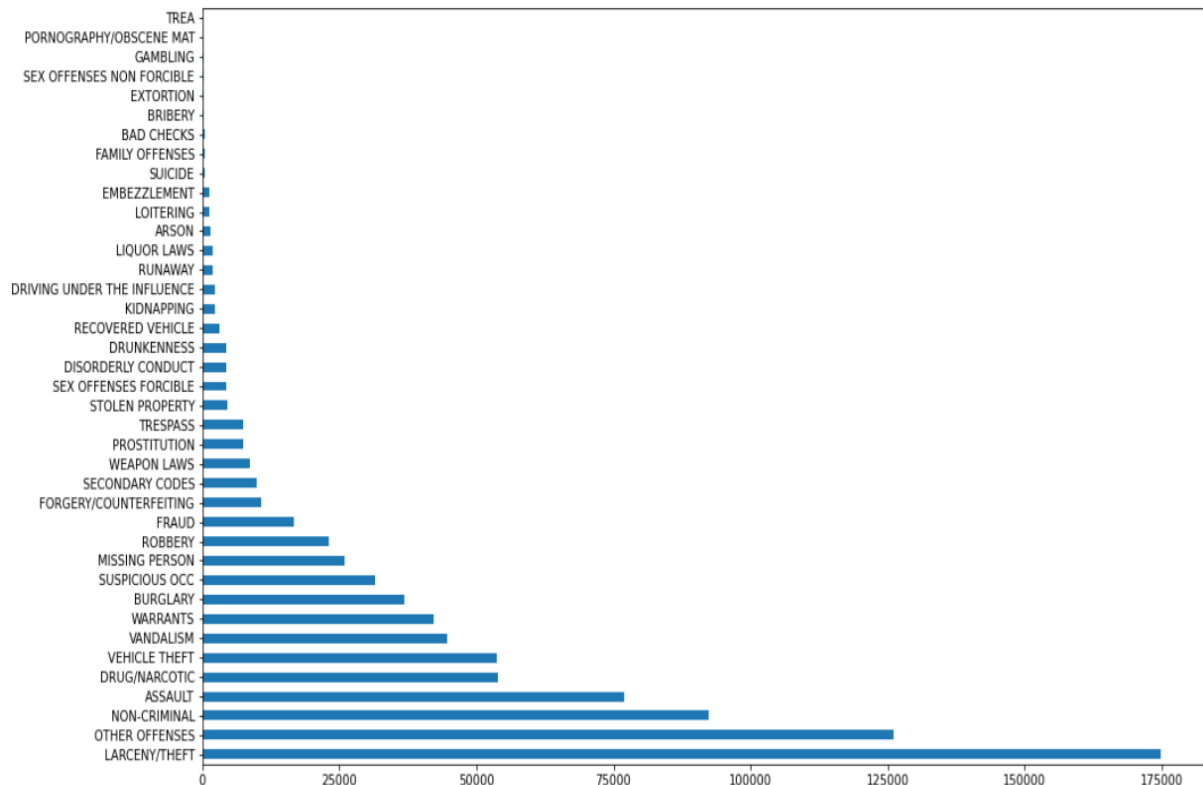Plotting the crimes occurrences according to the cases every day,

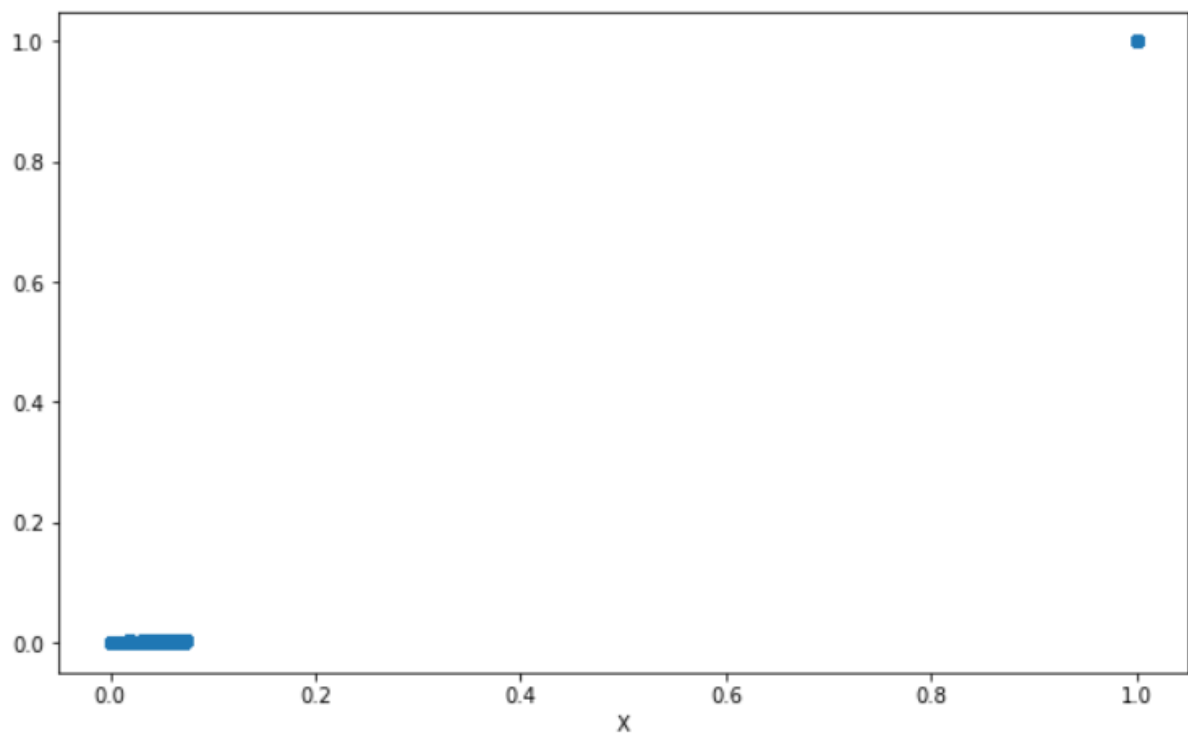Plotting the crimes occurrences according to the cases every hour,
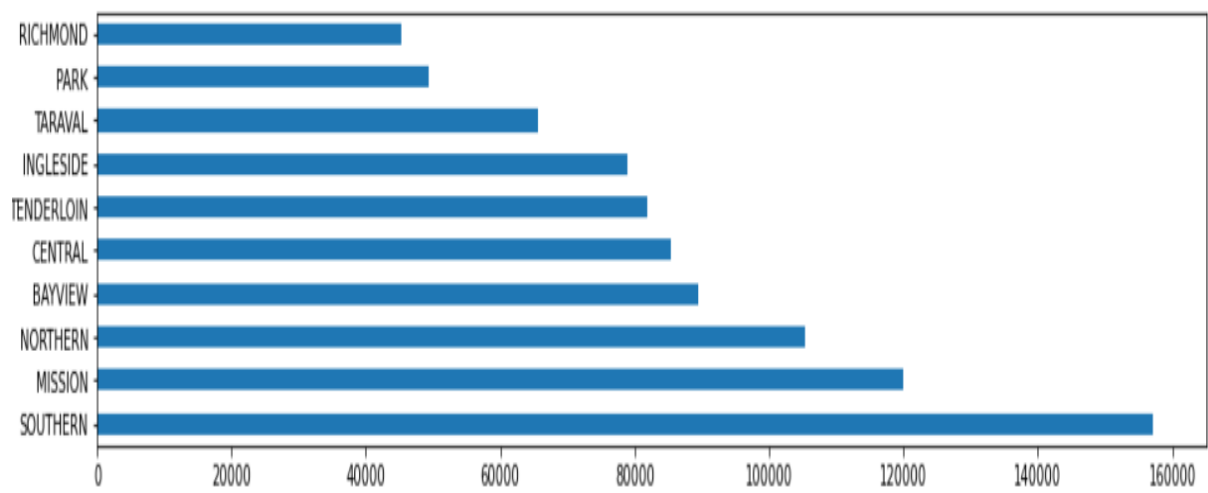
Distribution Over Roadtype:

Analysis based on Different offenses:

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

The Process of cleaning the dataset involves:

Define tokenization function 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. Alternatively, users can set parameter "gaps" to false indicating the regex "pattern" denotes "tokens" rather than splitting gaps, and find all matching occurrences as the tokenization result.

Define stop remover function using StopWordsRemover:
A feature transformer that filters out stop words from input. StopWordsRemover can filter out multiple columns at once by setting the inputCols parameter.

Define bag of words function for Descript variable using CountVectorizer:

CountVectorizer can be used as an estimator to extract the vocabulary, and generates a CountVectorizerModel. The model produces sparse representations for the documents over the vocabulary, which can then be passed to other algorithms like LDA. During the fitting process, CountVectorizer will select the top vocabSize words ordered by term frequency across the corpus. An optional parameter minDF also affects the fitting process by specifying the minimum number (or fraction if < 1.0) of documents a term must appear in to be included in the vocabulary.

Define function to Encode the values of category variable using StringIndexer:

StringIndexer encodes a string column of labels to a column of label indices. The indices are in (0, numLabels), ordered by label frequencies, so the most frequent label gets index 0. In our case, the label colum(Category) will be encoded to label indices, from 0 to 38; the most frequent label (LARCENY/THEFT) will be indexed as 0.

Define a pipeline to call these functions:

ML Pipelines provide a uniform set of highlevel APIs built on top of DataFrames that help users create and tune practical machine learning pipelines.

5. Build a multi-classification

The stages involve to perform multi-classification include:
1. Model training and evaluation
   Build baseline model

-Logistic regression using CountVectorizer features We will build a model to make predictions and score on the test sets using logistics regression using the dataset we transformed using count vectors. And we will see the top 10 predictions from the highest probability from our model, accuracy and other metrics to evaluate our model.

2. Build secondary models
   - Naive Bayes

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.

- Logistic regression and Naive Bayes using TF -IDF Features

Term frequency-inverse document frequency (TF-IDF) is a feature vectorization method widely used in text mining to reflect the importance of a term to a document in the corpus. Denote a term by t, a document by d, and the corpus by D. Term frequency TF(t,d) is the number of times that term t appears in

document d, while document frequency DF(t,D) is the number of documents that contains term t. If we only use term frequency to measure the importance, it is very easy to overemphasize terms that appear very often but carry little information about the document, e.g. "a", "the", and "of". If a term appears very often across the corpus, it means it doesn't carry special information about a particular document. Inverse document frequency is a numerical measure of how much information a term provides:

$$IDF(t,D)=log(\frac{|D|+1}{DF(t,D)+1}),$$

where |D| is the total number of documents in the corpus. Since logarithm is used, if a term appears in all documents, its IDF value becomes 0. Note that a smoothing term is applied to avoid dividing by zero for terms outside the corpus. The TF-IDF measure is simply the product of T and IDF:

$$TFIDF(t,d,D)=TF(t,d) \cdot IDF(t,D)$$

There are several variants on the definition of term frequency and document frequency. In MLlib, we separate TF and IDF to make them flexible. 3. Apply Logistic Regression Using Word2Vec features Word2Vec is an Estimator which takes sequences of words representing documents and trains a Word2VecModel. The model maps each word to a unique fixed-size vector. The Word2VecModel transforms each document into a vector using the average of all words in the document; this vector can then be used as features for prediction, document similarity calculations, etc.

Results:

In Logical Regression ,

Training data accuracy**:** 25.39137%

Testing data Accuracy: 25.37733%

In Random Forest,

Training Data accuracy: 19.195%

Testing Data Accuracy: 23.0025%

In Light Gradient Boosting,

Training Data Accuracy: 69.2721%

Testing data Accuracy: 68.628%

Conclusion:

Light Gradient Boosting proves to be better algorithm for text analysis than Log
-istic regression and Naives Bayes.