Coronavirus Tweet Sentiment Analysis

Almabetter Capstone Project

Team: AETHER

Ankur Bhattacharjee

Bhabakrishna Talukdar

Mayank Tiwari

Md Suhel Ansari

Pratheek T M

Dated: 26/03/2022

Outline:

Coronavirus just hit the world by surprise. A pandemic with such great casualties was not predicted in any sort of futuristic researches. This pandemic affected people both physically and mentally. One of the best ways to look at what this pandemic brought into present society is through social media.

Our job is to analyse the sentiment on Coronavirus pandemic. Twitter is our main focus for this job now. We need to glance though the tweets of different sentiments for an understanding of how they affect the society and also in which way different parts of the society have received the pandemic effect.

There were different sentiments some of them were purely positive or negative and few were more kind of sentimentally neutral. We can use various Machine Learning Algorithms to analyse the sentiments and their impact on the society through the tweets.

Problem Statement:

- We build a classification model to predict the sentiment of COVID-19 tweets. The tweets have been pulled from Twitter and manual tagging has been done then.
- Performing some EDA on the dataset to understand various features and their importance.
- Preparing our dataset for Machine Learning models by cleaning the tweets like removing stopwords etc.
- Selection of suitable models for further improvements on such models.
- Analysing the performance using some Evaluation metrics.

Introduction:

Coronavirus Tweet Sentiment Analysis is an approach to analyse and predict sentiment of tweets related to COVID-19. The tweets can be analysed with the help of Classification and Natural Language Processing(NLP) techniques, as the tweets can be categorized into discrete number of sentiments and those tweets will be in human language which have to be made possible for the computer to analyse them.

The dataset consists of the tweets from around the world along with the coded usernames and screennames of the Tweeter, location from which it was tweeted, time of the tweet and the sentiment of the tweet. Usernames and screennames are coded to avoid privacy concerns.

The steps of this analysis include data cleaning by handling null values, removing stopwords, special characters, URL's and many other unnecessary elements from the tweets, data exploration, data preprocessing, vectorization, modeling and evaluating individual models.

Data Description:

The names and usernames have been given codes to avoid any privacy concerns.

You are given the following information:

- **Location: Location from where the tweet has been done.**
- **Tweet At: The time of the tweet.**
- **Original Tweet: The tweet.**
- **Label: The sentiment of the tweet.**

Steps Involved:

Step 1 - Data Exploration:

- Checking the initial information of the dataset, such as number of rows and columns, data-types of columns and checking for null-values.
- Checking the description of categorical features, to have an idea about their distributions.
- Dropping the columns 'Username' and 'Screenname' since these columns were just serial numbers and all were unique.

Step 2 - Exploratory Data Analysis:

- Plotting the distribution of various sentiments among the tweets
- Plotting the distribution of locations from where the tweets have been done.
- Generating Wordclouds to display top hash tags for each sentiment.
- Plotting the distribution of tweets with respect their tweet time

Step 3 - Feature Engineering:

- Removal of stopwords, special characters, links, @users.
- Conversion of all words to lowercase and tokenisation.
- Performing stemming to replace all words with their root words.
- Creating 'cleaned-tweets' column that contains the tweets from the above process.
- Performing vectorisartion using TFIDF vectors.

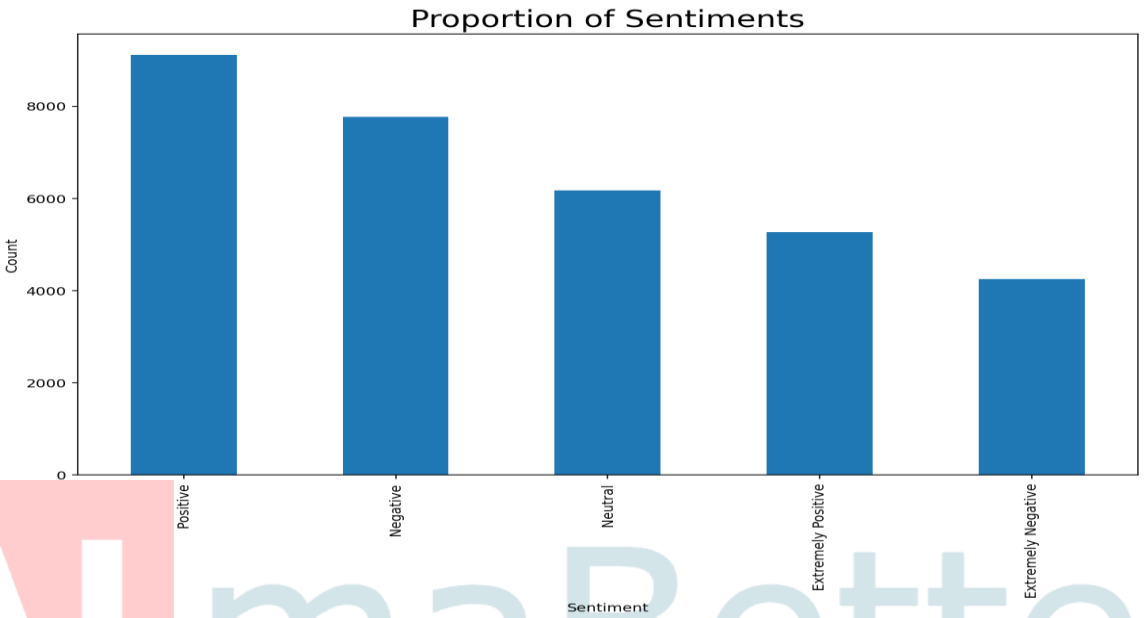
Step 4 - Model Training and Model Selection:

- Splitting the dataset into Training and Test sets.

- Training a set of classification models on the dataset and evaluating their results.
- Selection of the best model based on Evaluation Metrics.

Exploratory Data Analysis

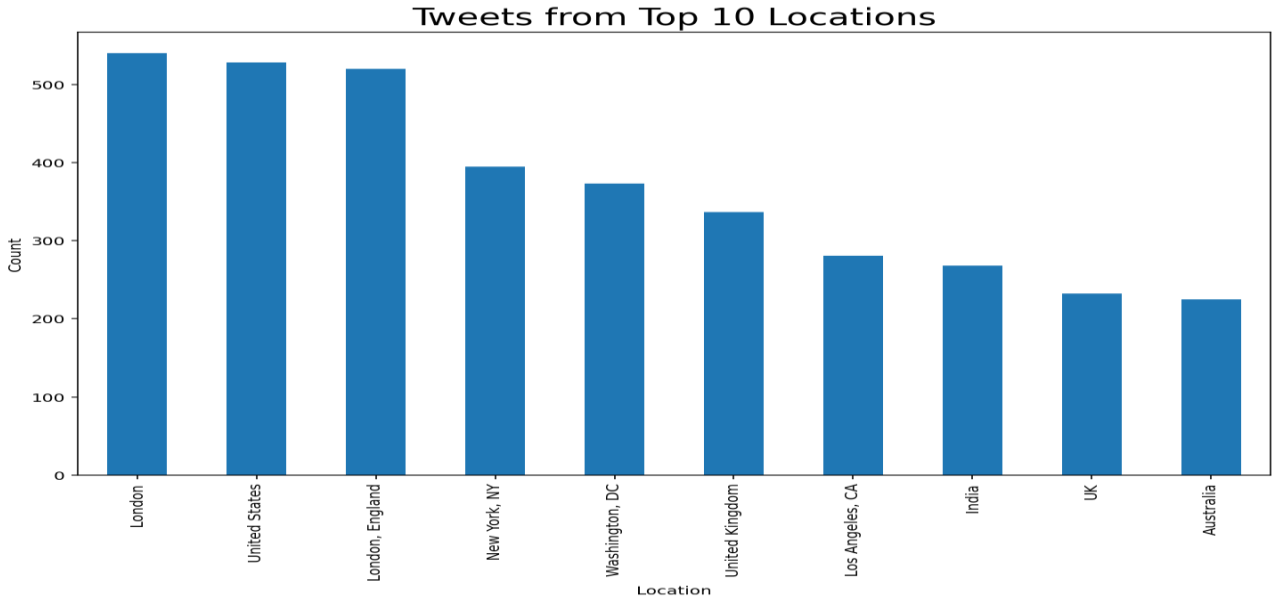
Proportion of Sentiments



Observation:

- **Maximum number of tweets are of Positive sentiment & Extremely Negative Sentiment are least in number.**

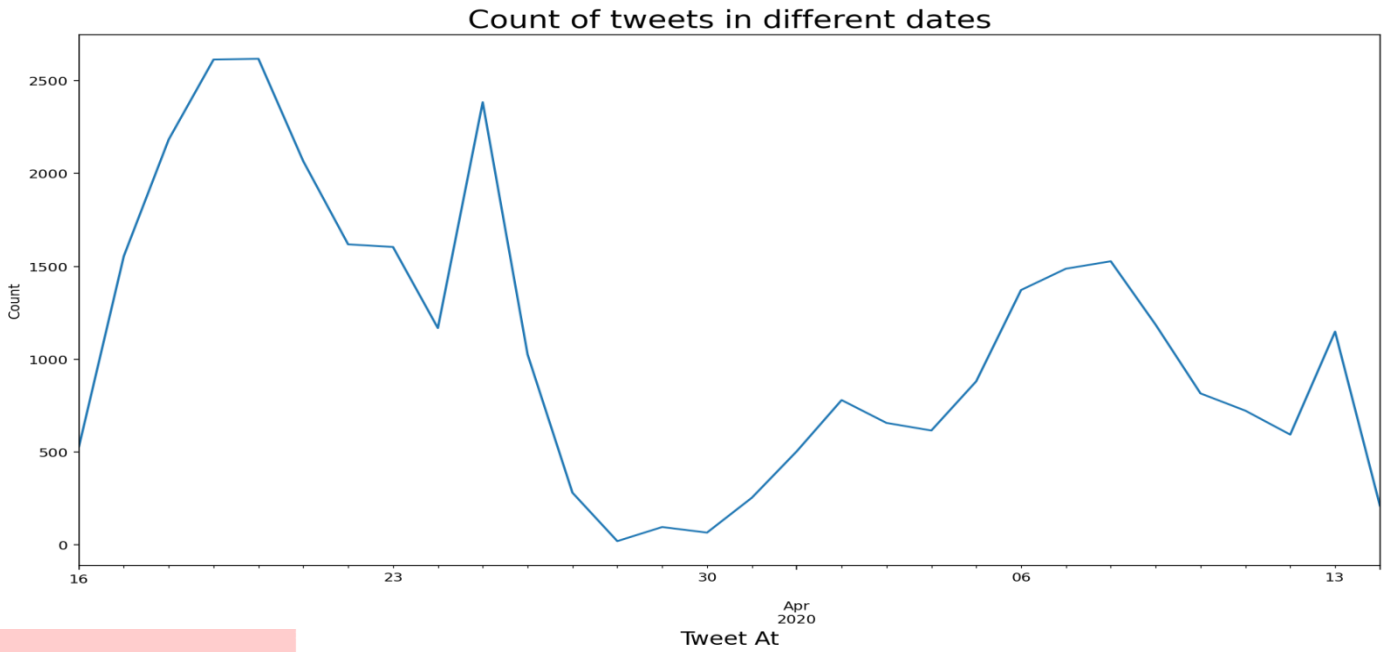
Tweets from Top 10 Locations.



Observation:

- **Most of the tweets are from London & US.**

Count of tweets in different dates.



Observation:

- **All the tweets have been done in the month of March and April.**

Cleaning of Tweets

Removal of stopwords, special characters, links, @users:

- **Stopwords:** Stop words are a set of commonly used words in a language. Examples of stop words in English are “a”, “the”, “is”, “are” and etc. Stop words are commonly used in Text Mining and Natural Language Processing (NLP) to eliminate words that are so commonly used that they carry very little useful information.
- We have removed the special characters like, !, @, #,$,%,^,&,*,(,),_,+,?,/ etc, urls and usernames mentioned as they were of no use for our analysis on sentiments.

Tokenization:

Tokenization is the process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.

Feature Engineering

Vectorization:

Word Embeddings or Word vectorization is **a methodology in NLP to map words or phrases from vocabulary to a corresponding vector of real numbers which used to find word predictions, word similarities/semantics**. The process of converting words into numbers are called Vectorization

TF-IDF Vectorizer:

TF-IDF stands for Term Frequency Inverse Document Frequency of records. It can be defined as the calculation of how relevant a word in a series or corpus is to a text. The meaning increases proportionally to the number of times in the text a word appears but is compensated by the word frequency in the corpus (data-set).

Terminologies:

- **Term Frequency:** In document d, the frequency represents the number of instances of a given word t. Therefore, we can see that it becomes more relevant when a word appears in the text, which is rational. Since the ordering of terms is not significant, we can use a vector to describe the text in the bag of term models. For each specific term in the paper, there is an entry with the value being the term frequency.

The weight of a term that occurs in a document is simply proportional to the term frequency.

$$tf(t,d) = count\ of\ t\ in\ d\ /\ number\ of\ words\ in\ d$$

- **Document Frequency:** This tests the meaning of the text, which is very similar to TF, in the whole corpus collection. The only difference is that in document d, TF is the frequency counter for a term t, while df is the number of occurrences in the document set N of the term t. In other words, the number of papers in which the word is present is DF.

$$df(t) = occurrence\ of\ t\ in\ documents$$

- **Inverse Document Frequency:** Mainly, it tests how relevant the word is. The key aim of the search is to locate the appropriate records that fit the demand. Since tf considers all terms equally significant, it is therefore not only possible to use the term frequencies to measure the weight of the term in the paper. First, find the document frequency of a term t by counting the number of documents containing the term:

$$df(t) = N(t)$$

where,

df(t) = Document frequency of a term t

N(t) = Number of documents containing the term t

Term frequency is the number of instances of a term in a single document only; although the frequency of the document is the number of separate documents in which the term appears, it depends on the entire

corpus. Now let's look at the definition of the frequency of the inverse paper. The IDF of the word is the number of documents in the corpus separated by the frequency of the text.

$$idf(t) = N/\ df(t) = N/N(t)$$

The more common word is supposed to be considered less significant, but the element (most definite integers) seems too harsh. We then take the logarithm (with base 2) of the inverse frequency of the paper. So the *if* of the term *t* becomes:

$$idf(t)\ =\ log(N/\ df(t))$$

- **Computation:** Tf-idf is one of the best metrics to determine how significant a term is to a text in a series or a corpus. tf-idf is a weighting system that assigns a weight to each word in a document based on its term frequency (tf) and the reciprocal document frequency (tf) (idf). The words with higher scores of weight are deemed to be more significant.

Usually, the tf-idf weight consists of two terms-

1. **Normalized Term Frequency (tf)**
2. **Inverse Document Frequency (idf)**

$$tf - idf(t,d)\ =\ tf(t,d)\ *\ idf(t)$$

In python tf-idf values can be computed using *TfidfVectorizer()* method in *sklearn* module.

Model Training and Model Selection

Model Training:

In Model Training we split our cleaned dataset into two parts, 'train' and 'test', where we use the 'train' dataset for training different models(i.e. to learn the relationship between features and target classes) and 'test' dataset for evaluating the results of models(i.e. to check models learning accuracy).

Model Selection:

In Mode Selection out of the models we trained in Model Training, we choose the model that performed best using classification metrics like accuracy, precision, recall and f1-score.

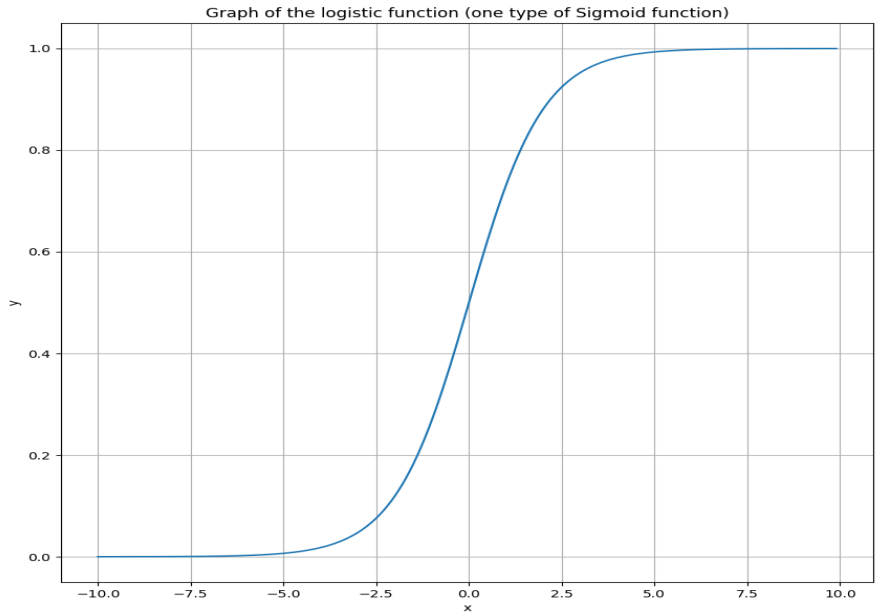
Models:

Logistic Regression:

Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable(or output), y, can take only discrete values for a given set of features(or inputs), X.

Contrary to popular belief, logistic regression IS a regression model. The model builds a regression model to predict the probability that a given data entry belongs to the category numbered as “1”. Just like Linear regression assumes that the data follows a linear function, Logistic regression models the data using the sigmoid function.

$$g(z) = \frac{1}{1 + e^{z}}$$



Logistic regression becomes a classification technique only when a decision threshold is brought into the picture. The setting of the threshold value is a very important aspect of Logistic regression and is dependent on the classification problem itself.
The decision for the value of the threshold value is majorly affected by the values of precision and recall.Ideally, we want both precision and recall to be 1, but this seldom is the case.

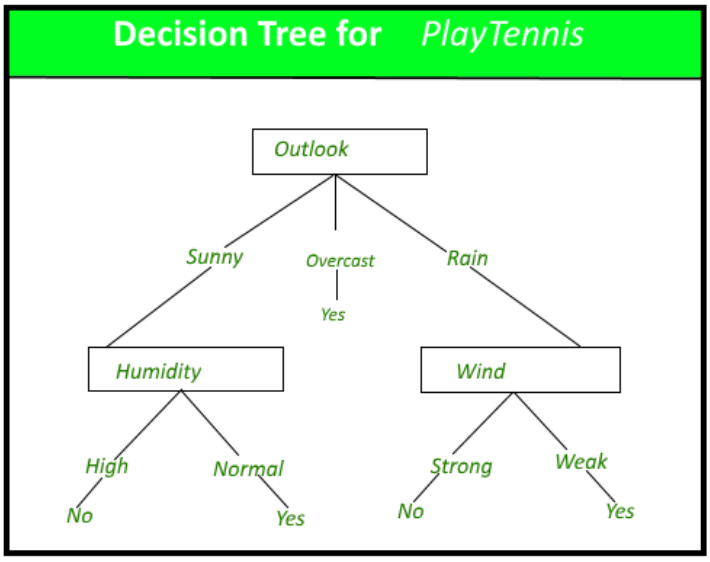
In the case of a Precision-Recall tradeoff, we use the following arguments to decide upon the threshold:-
1. Low Precision/High Recall: In applications where we want to reduce the number of false negatives without necessarily reducing the number of false positives, we choose a decision value that has a low value of Precision or a high value of Recall. For example, in a cancer diagnosis application, we do not want any affected patient to be classified as not affected without giving much heed to if the patient is being wrongfully diagnosed with cancer. This is because the absence of cancer can be detected by further medical diseases but the presence of the disease cannot be detected in an already rejected candidate.
2. High Precision/Low Recall: In applications where we want to reduce the number of false positives without necessarily reducing the number of false negatives, we choose a decision value that has a high value of Precision or a low value of Recall. For example, if we are classifying customers whether they will react positively or negatively to a personalized advertisement, we want to be absolutely sure that the customer will react positively to the advertisement because otherwise, a negative reaction can cause a loss

of potential sales from the customer.
Based on the number of categories, Logistic regression can be classified as:

1. **Binomial:** target variable can have only 2 possible types: “0” or “1” which may represent “win” vs “loss”, “pass” vs “fail”, “dead” vs “alive”, etc.
2. **Multinomial:** target variable can have 3 or more possible types which are not ordered(i.e. types have no quantitative significance) like “disease A” vs “disease B” vs “disease C”.
3. **Ordinal:** it deals with target variables with ordered categories. For example, a test score can be categorized as:“very poor”, “poor”, “good”, “very good”. Here, each category can be given a score like 0, 1, 2, 3.

Decision Trees

Decision Tree: Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.



A decision tree for the concept Play Tennis.

Construction of Decision Tree:
A tree can be *“learned”* by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called *recursive partitioning*. The recursion is completed when the subset at a node all has the same value of the target variable, or when

splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

Decision Tree Representation:

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute as shown in the above figure. This process is then repeated for the subtree rooted at the new node.

The decision tree in above figure classifies a particular morning according to whether it is suitable for playing tennis and returning the classification associated with the particular leaf.(in this case Yes or No).

K-Nearest Neighbours:

K-Nearest Neighbours is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.

It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data).

We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

Support Vector Machines:

Support Vector Machine (SVM) is a relatively simple **Supervised Machine Learning Algorithm** used for classification and/or regression. It is more preferred for classification but is sometimes very useful for regression as well. Basically, SVM finds a hyper-plane that creates a boundary between the types of data. In 2-dimensional space, this hyper-plane is nothing but a line.

In SVM, we plot each data item in the dataset in an N-dimensional space, where N is the number of features/attributes in the data. Next, find the optimal hyper-plane to separate the data. So by this, you must have understood that inherently, SVM can only perform binary classification (i.e., choose between two classes). However, there are various techniques to use for multi-class problems.

XGBoost:

XGBoost stands for Extreme Gradient Boosting, which was proposed by the researchers at the University of Washington. It is a library written in C++ which optimizes the training for Gradient Boosting. In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

Random Forest:

Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data and hence the output doesn't depend on one decision tree but multiple decision trees. In the case of a classification problem, the final output is taken by using the majority voting classifier. In the case of a regression problem, the final output is the mean of all the outputs. This part is Aggregation.

The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

Extra Trees Regressor:

Extremely Randomized Trees Classifier (Extra Trees Classifier) is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a "forest" to output it's classification result. In concept, it is very similar to a Random Forest Classifier and only differs from it in the manner of construction of the decision trees in the forest.

Each Decision Tree in the Extra Trees Forest is constructed from the original training sample. Then, at each test node, Each tree is provided with a random sample of k features from the feature-set from which each decision tree must select the best feature to split the data based on some mathematical criteria (typically the Gini Index). This random sample of features leads to the creation of multiple de-correlated decision trees.

To perform feature selection using the above forest structure, during the construction of the forest, for each feature, the normalized total reduction in the mathematical criteria used in the decision of feature of split (Gini Index if the Gini Index is used in the construction of the forest) is computed. This value is called the Gini Importance of the feature. To perform feature selection, each feature is ordered in descending order according to the Gini Importance of each feature and the user selects the top k features according to his/her choice.

CatBoost Classifier

Many machine learning algorithms require data to be numeric. So, before training a model, we need to convert categorical data into numeric form. There are various categorical encoding methods available. Catboost is one of them. Catboost is a target-based categorical encoder. It is a supervised encoder that encodes categorical columns according to the target value. It supports binomial and continuous targets.

Target encoding is a popular technique used for categorical encoding. It replaces a categorical feature with average value of target corresponding to that category in training dataset combined with the target probability over the entire dataset. But this introduces a target leakage since the target is used to predict the target. Such models tend to be overfitted and don't generalize well in unseen circumstances.

A CatBoost encoder is similar to target encoding, but also involves an ordering principle in order to overcome this problem of target leakage. It uses the principle similar to the time series data validation. The values of target statistic rely on the observed history, i.e, target probability for the current feature is calculated only from the rows (observations) before it.

Categorical feature values are encoded using the following formula:

$$\frac{TargeSum + Prior}{FeatureCount + 1}$$

***TargetCount*:** Sum of the target value for that particular categorical feature (upto the current one).

***Prior*:** It is a constant value determined by (sum of target values in the whole dataset)/(total number of observations (i.e. rows) in the dataset)

***FeatureCount*:** Total number of categorical features observed upto the current one with the same value as the current one.

With this approach, the first few observations in the dataset always have target statistics with much higher variance than the successive ones. To reduce this effect, many random permutations of the same data are used to calculate target statistics and the final encoding is calculated by averaging across these permutations. So, if the number of permutations is large enough, the final encoded values obey the following equation:

$$\frac{TargeSum + Prior}{FeatureCount + 1}$$

***TargetCount*:** Sum of the target value for that particular categorical feature in the whole dataset.

***Prior*:** It is a constant value determined by (sum of target values in the whole dataset)/(total number of observations (i.e. rows) in the dataset)

***FeatureCount*:** Total number of categorical features observed in the whole dataset with the same value as the current one.

For Example, if we have categorical feature column with values,

color=[“red”, “blue”, “blue”, “green”, “red”, “red”, “black”, “black”, “blue”, “green”] and target column with values, target=[1, 2, 3, 2, 3, 1, 4, 4, 2, 3]

Then, prior will be *25/10 = 2.5*

For the “red” category, TargetCount will be 1+3+1 = 5 and FeatureCount = 3

And hence, encoded value for “red” will be (5+2.5) /(3+1)=1.875

Naïve Bayes:

Naive Bayes is among one of the very simple and powerful algorithms for classification based on **Bayes Theorem** with an assumption of independence among the predictors. The Naive Bayes classifier assumes that the presence of a feature in a class is not related to any other feature. Naive Bayes is a classification algorithm for binary and multi-class classification problems.

Bayes Theorem

- Based on prior knowledge of conditions that may be related to an event, Bayes theorem describes the probability of the event
- conditional probability can be found this way
- Assume we have a Hypothesis(*H*) and evidence(*E*),
  According to Bayes theorem, the relationship between the probability of Hypothesis before getting the evidence represented as *P(H)* and the probability of the hypothesis after getting the evidence represented as *P(H|E)* is:

$$P(H|E) = P(E|H) * P(H)/P(E)$$

- **Prior probability** = *P(H)* is the probability before getting the evidence
  Posterior probability = *P(H|E)* is the probability after getting evidence
- In general,

$$P(class|data) = (P(data|class) * P(class)) / P(data)$$

Bayes Theorem Example

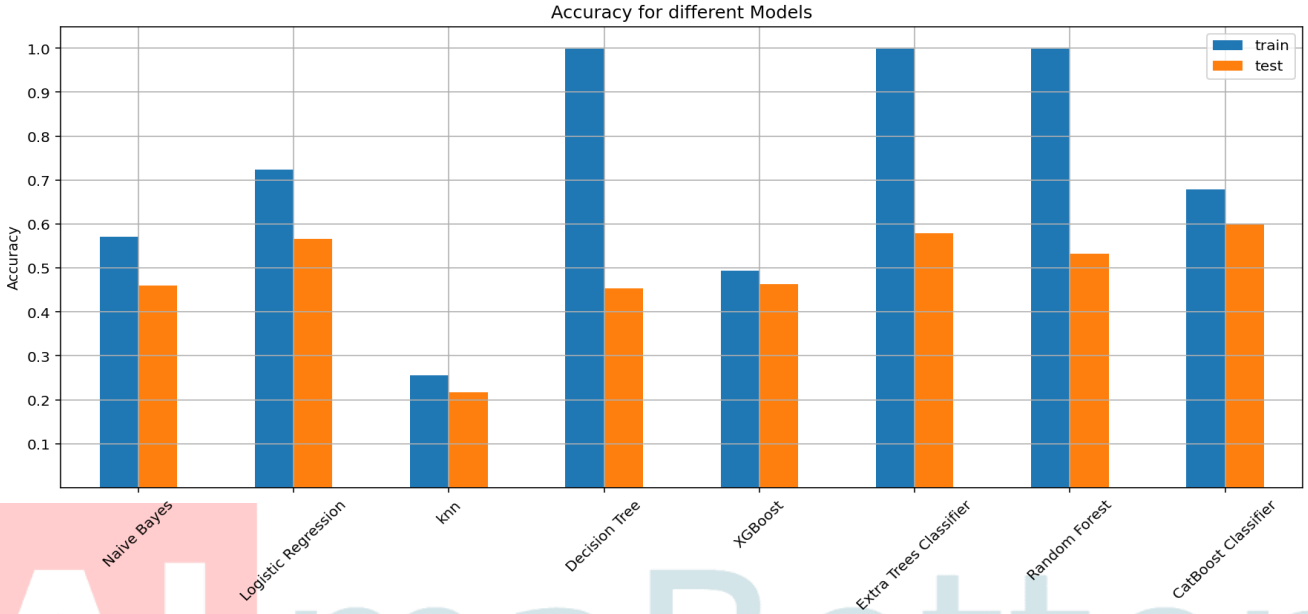
Assume we have to find the probability of the randomly picked card to be king given that it is a face card.
There are *4* Kings in a Deck of Cards which implies that *P(King) = 4/52*
as all the Kings are face Cards so *P(Face|King) = 1*
there are *3* Face Cards in a Suit of *13 cards* and there are *4 Suits* in total so *P(Face) = 12/52*
Therefore,

$$P(King|face) = P(face|king) * P(king)/P(face) = 1/3$$

Combined Evaluation metric results of above models:

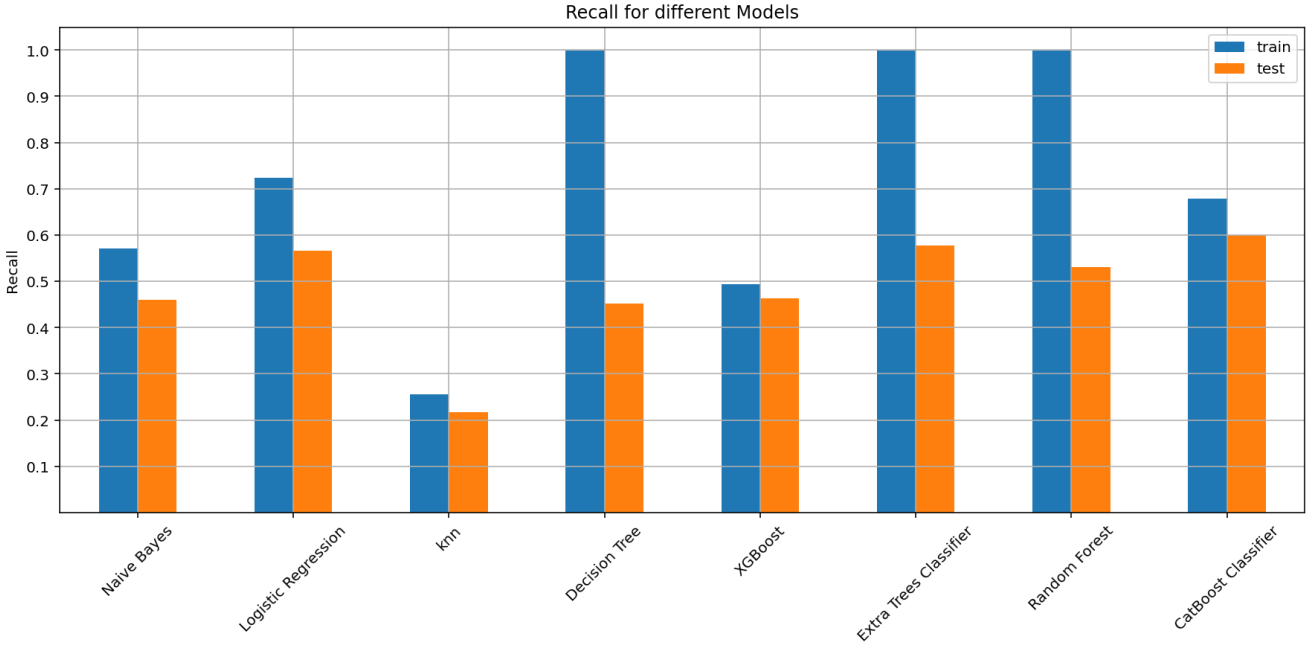
Accuracy

	Evaluation	Naive Bayes	Logistic Regression	knn	Decision Tree	XGBoost	Extra Trees Classifier	Random Forest	CatBoost Classifier
train	Accuracy	0.570227	0.723264	0.256174	0.999565	0.493019	0.999565	0.999565	0.679238
test	Accuracy	0.459873	0.566191	0.216684	0.452952	0.462966	0.579149	0.531512	0.600353



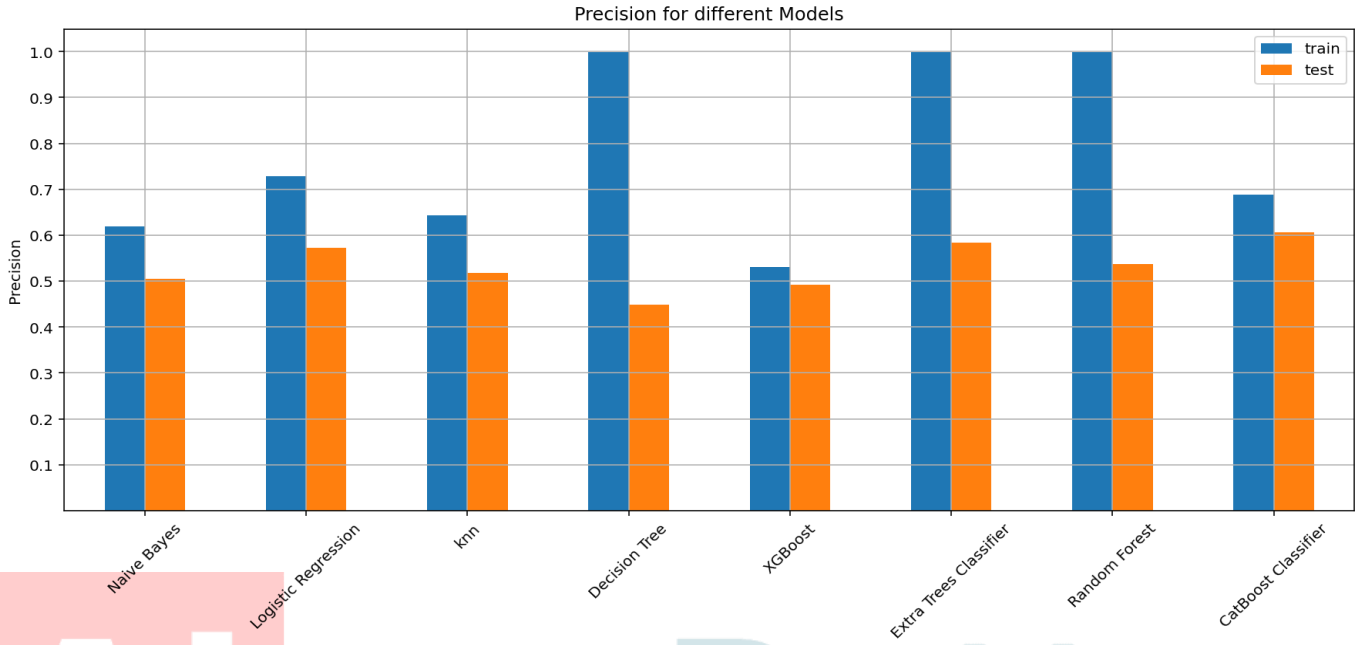
Recall

	Evaluation	Naive Bayes	Logistic Regression	knn	Decision Tree	XGBoost	Extra Trees Classifier	Random Forest	CatBoost Classifier
train	Recall	0.570227	0.723264	0.256174	0.999565	0.493019	0.999565	0.999565	0.679238
test	Recall	0.459873	0.566191	0.216684	0.451848	0.462966	0.577382	0.530997	0.600353



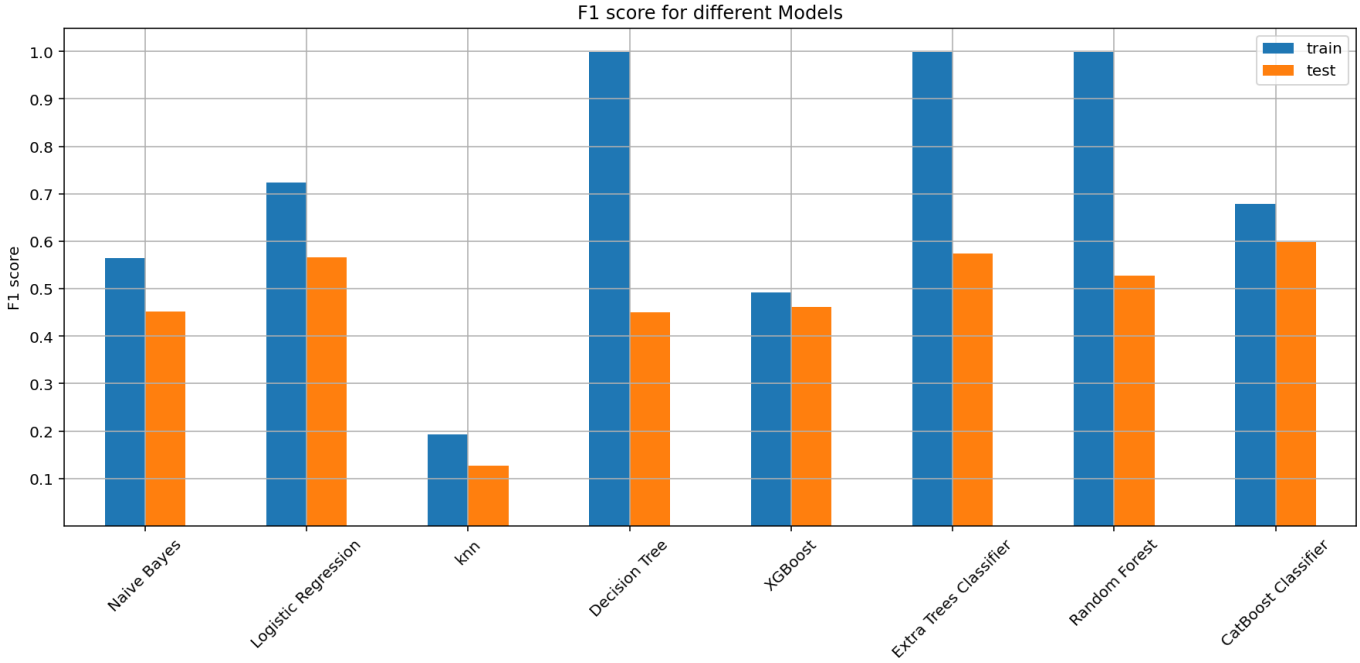
Precision

	Evaluation	Naive Bayes	Logistic Regression	knn	Decision Tree	XGBoost	Extra Trees Classifier	Random Forest	CatBoost Classifier
train	Precision	0.619172	0.729276	0.643623	0.999565	0.530422	0.999565	0.999565	0.688561
test	Precision	0.504285	0.572416	0.517649	0.449340	0.491385	0.583210	0.536499	0.605882



F1-score

	Evaluation	Naive Bayes	Logistic Regression	knn	Decision Tree	XGBoost	Extra Trees Classifier	Random Forest	CatBoost Classifier
train	F1_score	0.564672	0.723557	0.192066	0.999565	0.492796	0.999565	0.999565	0.678671
test	F1_score	0.451229	0.566128	0.127157	0.450009	0.460968	0.574254	0.527791	0.597790



Observations:

Visualizing all the graphs of different metrics....

- **Naive Bayes, Logistic Regression and Cat Boost** are performing better than other algorithms.
- The tree based algorithms like **Decision tree, Extra Tree Classifier and Random Forest** are more likely to over-fit.
- Among all the algorithms **Cat Boost** is performing best

Conclusion:

From initial explorations, columns 'UserName' and 'ScreenName' are serials and did not contribute to the analysis. Positive sentiment is the highest in proportion and Extremly Negative is the lowest among five sentiments. Highest number of tweets is recorded from London. All tweets are recorded in the months of March and April 2020.

Word Clouds of different sentiments were not so distinct, which indicated that the 'OriginalTweet' column has a lot of unnecessary information such as stop-words, URL's, @users, punctuations, special characters, numerical elements etc. So we removed them, converted every word to lowercase, performed tokenization and stemming.

Finally, after performing vectorization on the data, a set of classification models were built, and evaluated using metrics accuracy, recall, precision and F1-score. Models Naïve Bayes, Logistic Regression and CatBoost classifier performed comparatively better. Tree based models like Decision Tree, Random Forest, Extra-Trees Classifier and XGBoost were overfitting.

CatBoost Classifier with F1-score of 0.69 on the training set and 0.60 on the test set, is the best generalisation from training data to testing data and it gives the best balance between overfitting and underfitting over other models.

References:

i. GeeksForGeeks
ii. Towardsdatascience