

**Coronavirus Tweets Sentiment Analysis**

**Debanjan Ganguly**

**Data Science Trainee**

**AlmaBetter, Bengaluru**

**CONTENTS**

**Page**

* **Introduction - - - 3**
* **Problem Statement - - - 3**
* **Dataset Analysis - - - 4**
* **Exploratory Data Analysis - - - 5**
* **Text Preprocessing - - - 8**
* **Text Normalization - - - 9**
* **Literature Survey - - - 10**
* **Experimental Analysis - - - 15**
* **Model Selection - - - 17**
* **Limitation - - - 20**
* **Scope of Improvement - - - 20**
* **Conclusion - - - 20**
* **References - - - 21**

**Introduction:**

NLP or Natural Language Processing is a new emerging hot topic in field of Data Science & Machine Learning. NLP is used to interpret human language and behavior. NLP combines the power of linguistics and computer science to study the rules and structure of language, and create intelligent systems (run on machine learning and NLP algorithms) capable of understanding, analyzing, and extracting meaning from text and speech.

Sentiment Analysis is one of the convenient applications of NLP. It does the task of classifying the polarity of a given text. For instance, a text-based tweet can be categorized into either "positive", "negative", or even "neutral" also. Given the text and accompanying labels, a model can be trained to predict the correct sentiment. Vader, TextBlob, Google Cloud Natural Language API are examples of some of the pretrained model for sentiment analysis.

This analysis focuses on the supervised ML-based approach, which is computationally fast and exhibits promising classification results. Aspect-based analysis has been performed using a text classifier model built from scratch. Statistical models like Naïve Bayes, SVM, DecisionTree has been used to train and evaluate performance of our model, all these algorithms are available in python [scikit-learn](https://scikit-learn.org/stable/) library. The model has been evaluated using standard metrics like balanced accuracy, f1 score, roc-auc score etc. The best model among those has been selected to predict the final result.

**Problem Statement:**

This challenge asks you to 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. The names and usernames have been given codes to avoid any privacy concerns.

**Dataset Analysis:**

We have a collection of 41157 tweets divided within 5 categories i.e. “Extremely Positive”, “Positive”, “Neutral”, “Negative” and “Extremely Negative”. Let us look through our features,

* UserName: a unique identifier for user
* ScreenName: a unique identifier
* Location: - location from where the tweets were posted
* TweetAt: date and time when the tweets were posted
* OriginalTweet: the default feature, text version of curated tweets
* Sentiment: the target, 5 distinct string categories

There is no missing or duplicate value present for our feature column “OriginalTweet” and target column “Sentiment”.

**Exploratory Data Analysis:**

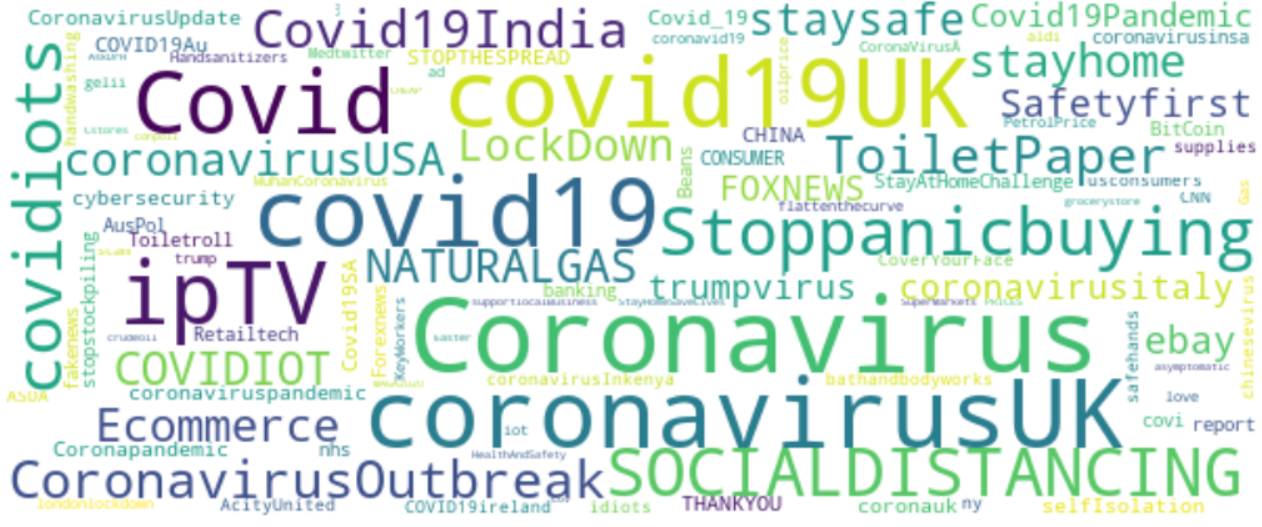
Exploratory Data Analysis or EDA perform a key role to become acquainted with data to drive intuition and begin to formulate testable hypothesis. This process typically makes use of descriptive statistics and visualizations.

Fig 1. Most Frequent Hashtags used

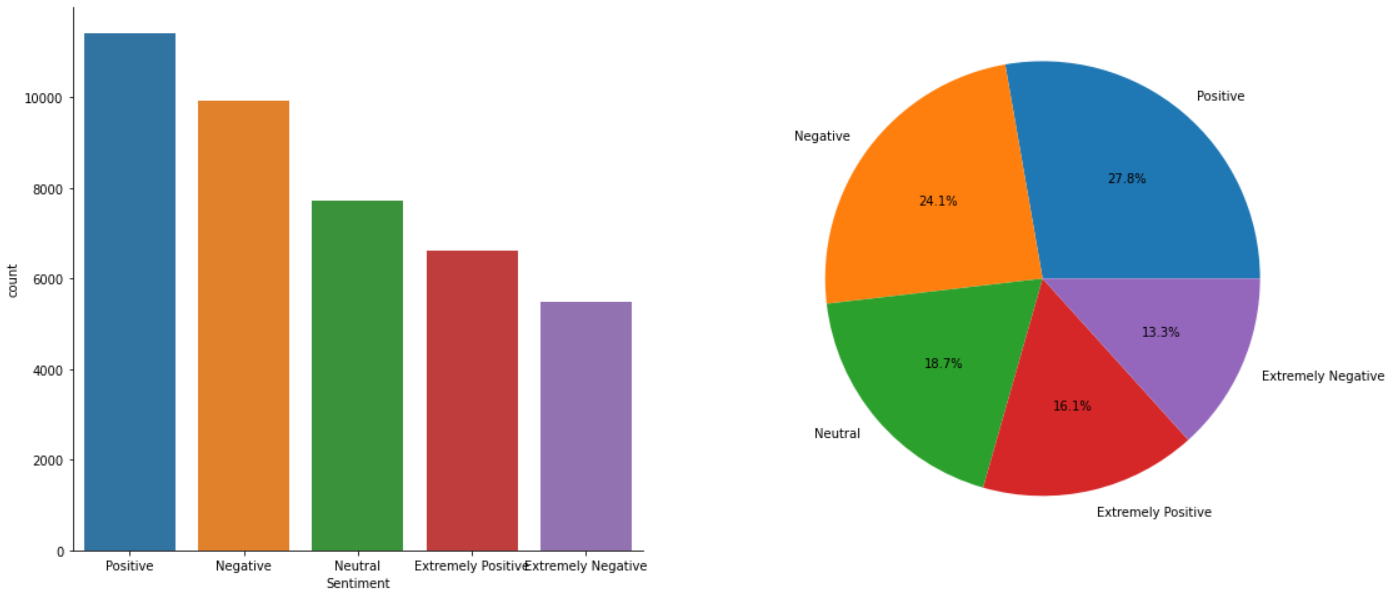
Hashtags (#) in tweets are an essential to get a broad overview of the topics of tweets. It can be viewed most of these hashtags are for coronavirus, very similar. But still there is some unusual or like present then situation hashtags like #Stoppanicbuying or #Ecommerce. We’ve seen a huge price hike for essential commodities and a boom in online ecommerce industries during lockdown days.

Fig 2. Proportion of Tweets Sentiment Labels

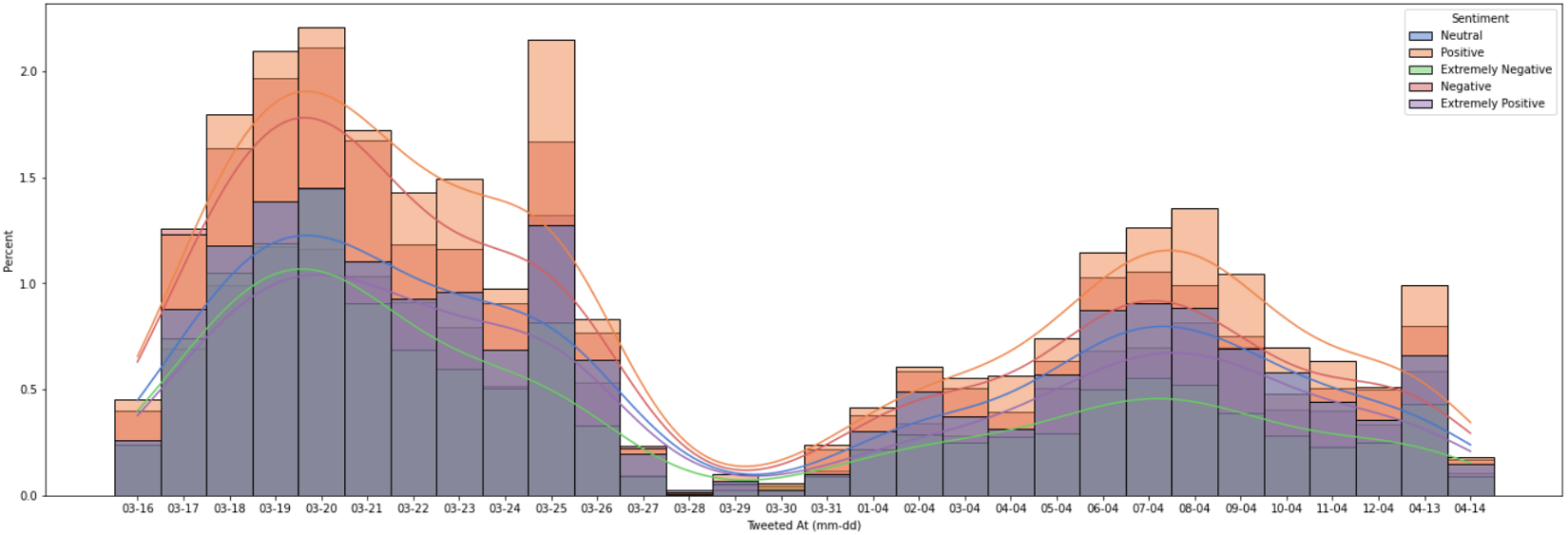
From the above fig 2, Positive tweets count are highest with 11422 samples. The ratio for target labels from left to right are respectively 1.208, 2.083, 1.407, 1.809, 1.0. Let us look at the distribution of tweets over time.

Fig 3. Distribution of Tweet Sentiments over time

It can be noticed that at every point of time, the rate of “Positive” labeled tweets are highest among all. “Extremely Negative” tweets are low in ratio for every point of time too. People never believed this is the new normal.

Following this, below are some visualizations for common and unique words in terms of various marked labels.

Fig 4. 200 Most Common words ([Link to Cell](https://colab.research.google.com/drive/19JBAqC_ZVS8nnbkRemPwXzK3_TNzguKM#scrollTo=GAoOP5H0YEf7&line=1&uniqifier=1))

Fig 5. 200 Most Common Negative words ([Link to Cell](https://colab.research.google.com/drive/19JBAqC_ZVS8nnbkRemPwXzK3_TNzguKM#scrollTo=zb4lUHd8Y4c8&line=1&uniqifier=1))

Fig 6. 200 Most Common Positive words ([Link to Cell](https://colab.research.google.com/drive/19JBAqC_ZVS8nnbkRemPwXzK3_TNzguKM#scrollTo=jiiUFLP7Y86P&line=2&uniqifier=1))



Fig 7. 200 Most Frequent Unique Positive Words ([Link to Cell](https://colab.research.google.com/drive/19JBAqC_ZVS8nnbkRemPwXzK3_TNzguKM#scrollTo=kZsCSbXKsBuK&line=2&uniqifier=1))

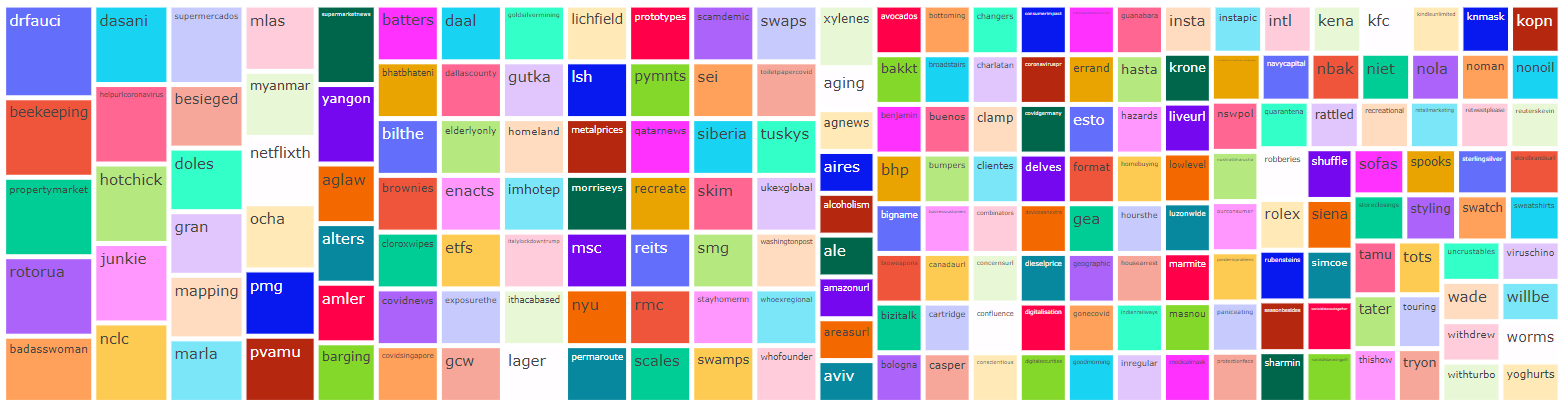
Fig 8. 200 Most Frequent Unique Negative Words ([Link to Cell](https://colab.research.google.com/drive/19JBAqC_ZVS8nnbkRemPwXzK3_TNzguKM#scrollTo=zPG6AtyXz5ft&line=3&uniqifier=1))

Fig 9. 200 Most Frequent Unique Neutral Words ([Link to Cell](https://colab.research.google.com/drive/19JBAqC_ZVS8nnbkRemPwXzK3_TNzguKM#scrollTo=mzAsudtM05Kn&line=3&uniqifier=1))

**Text Preprocessing:**

Text preprocessing is traditionally an important step for natural language processing (NLP) tasks. It transforms text into a more digestible form so that machine learning algorithms can perform better.

1. Lower Case Texts: This process converts all letters to its lowercase format. This helps to reduce dimension and it is a great way to deal with sparsity issues.
2. Tokenization: This process split the sequence of strings into words. It removes all the punctuations from the text data and gives words of text which is called tokens. Basically, it split the words on the basis of non-letter characters like space, commas, full stop or non utf-8 compliant whitespaces. In this step, all punctuation marks and whitespaces are removed.
3. Regular Expression Based Processing: Most of the tweets contain some mentioned user with “@username” mark. All of these mentioned user names has been replaced by a single common word “user”. Also, any of pasted or attached link has also been replaced with single common word “url”. Also, any 3 consecutive letters have been replaced with 2 letters of the same alphabet.
4. Filter Tokens by Length: This is to remove all irrelevant small length tokens which have no greater importance in our model. Any tokens having a length less than 3 has been removed from corpus.
5. Remove Stopwords: This process removes words from the document which does not play any important in giving intelligent pattern or information. Ex: the words like “as”, “by”, “are”, “then”, “with” etc.

After applying all above-mentioned preprocessing steps, a sample text would like below,

Sample Text:

Post Processed Text:

**Text Normalization:**

Text Normalization is a process that converts a list of words to a more uniform sequence. This is useful in preparing text for later processing and also it does improve text matching. Stemming and Lemmatization are two techniques used for text normalization. We’ve used the stemming to normalize tokens before vectorize it.

Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. We already know that a word has one root-base form but having different variations, for example, “play” is a root-base word and playing, played, plays are the different forms of a single word. So, these words get stripped out, they might get the incorrect meanings or some other sort of errors. Stemming occurs in such a way that depicting a group of relatable words under the same stem, even if the root has no appropriate meaning. It is a rule-based approach because it slices the inflected words from prefix or suffix as per the need using a set of commonly underused prefix and suffix, like “-ing”, “-ed”, “-es”, “-pre”, etc. It results in a word that is actually not a word. There are mainly two errors that occur while performing Stemming, Over-stemming, and Under-stemming. Over-steaming occurs when two words are stemmed from the same root of different stems. Under-stemming occurs when two words are stemmed from the same root of not a different stem. Depending on different stemming algorithm, a single word can have different post-normalization result. For example, based on algorithm PorterStemmer and SnowballStemmer the word “hourly” can results in “hourli” or “hour” respectively. We’ve used the latter mentioned techniques to normalize our data.

**Literature Survey:**

Sentiment analysis is a growing area of Natural Language Processing with research ranging from document level classification (Pang and Lee 2008) to learning the polarity of words and phrases (e.g., (Hatzivassiloglou and McKeown 1997; Esuli and Sebastiani 2006)). however, the informal and specialized language used in tweets, as well as the very nature of the microblogging domain make Twitter sentiment analysis a very different task. Researchers have also begun to investigate various ways of automatically collecting training data, explore the use of part-of-speech features, adding emoticons and hashtags of tweets as a part of feature for training. The analysis we’ve done is a 5-way which later has been converted to a 3-way polarity analysis. The techniques or algorithms we’ve used to vectorize the processed text and train the machine learning models, the definition of these processes is as below,

* **Bag of Words:** BOW is a text vectorization method to extract features from text documents. These features can be used for training machine learning algorithms. It creates a vocabulary of all the unique words occurring in all the documents in the training set. In simple terms, it’s a collection of words to represent a sentence with word count and mostly disregarding the order in which they appear. The BOW model is very simple as it discards all the information and order of the text and just considers the occurrences of the word, in short it converts a sentence or a paragraph into a bag of words with no meaning. It converts the documents to a fixed-length vector of numbers. A unique number is assigned to each word (generally index of an array) along with the count representing the number of occurrences of that word. This is the encoding of the words, in which we are focusing on the representation of the word and not on the order of the word. For example, to consider a list of two sentences,

‘Hello my name is james, this is my python notebook’

We have 8 unique words in the text and hence 8 different columns each representing a unique word in the matrix. The row represents the word count. Since the words ‘is’ and ‘my’ were repeated twice we have the count for those particular words as 2 and 1 for the rest. The vectorizer representation for these two sentences will look like following,

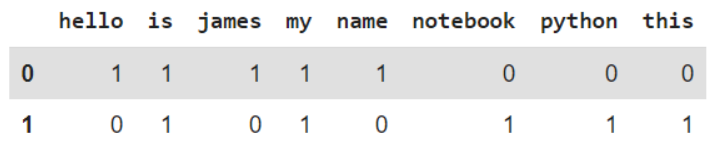


Fig 10. Representation of BOW

We’ve used CountVectorizer provided by scikit-learn to implement BOW techniques within our model. It automatically applies text lowercasing, removes stopwords and tokenize out of the box, and returns a sparse matrix where features are column and rows are the count of occurrence in every sample.

* **Naïve Bayes:** Naive Bayes falls under the umbrella of supervised machine learning algorithms that are primarily used for classification. This algorithm is called naïve because the classifier assumes that the input features that go into the model are independent of each other. Hence, changing one input feature won’t affect any of the others. It's therefore naive in the sense that this assumption may or may not be true, and it most probably isn't. Bayes Rule revolves around the concept of deriving a hypothesis (H) from the given evidence (E). It relates two notions: the probability of the hypothesis before getting the evidence, P(H), and the probability of the hypothesis after getting the evidence, P(H|E). In general, it’s given by the following equation:

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

From equations,

**P(H|E)** How often H happens given that E happens

**P(E|H)** How often E happens given that H happens

**P(H)** How likely H happens on its own

**P(E)**  How likely E happens on its own

In simple terms, it provides a way to calculate the probability of a hypothesis given the evidence. The Bayes Rule provides the formula to compute the probability of output (Y) given the input (X). In real-world problems, unlike the hypothetical assumption of having a single input feature, we have multiple X variables. When we can assume the features are independent of each other, we extend the Bayes Rule to what is called Naive Bayes. Consider a case where there are multiple inputs (X1, X2, X3 ... Xn). We predict the outcome (Y) using the Naive Bayes equation as follows:

**P(Y=k | X1...Xn) = ( P(X1 | Y=k) \* P(X2 | Y=k) \* P(X3 | Y=k) \* ....\* P(Xn | Y=k) ) \* P(Y=k) / P(X1)\*P(X2)\*P(X3)\*P(Xn)**

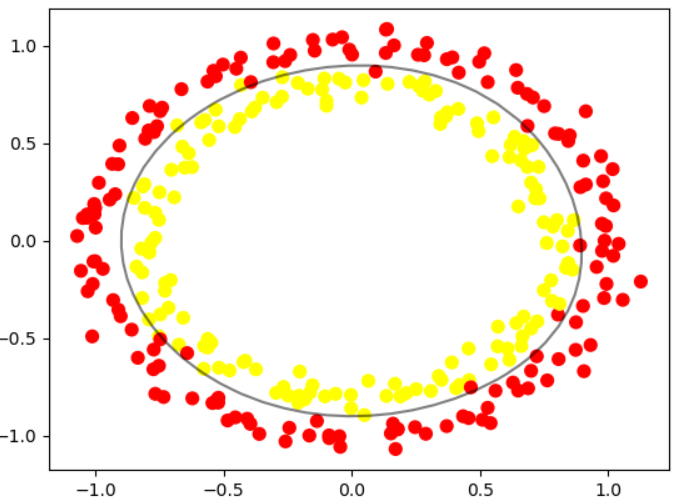
**P(Y=k | X1...Xn)** is called the Posterior Probability, which is the probability of an outcome given the evidence.

**P(X1 | Y=k) \* P(X2 | Y=k) \* ... P(Xn | Y=k)** is the probability of the likelihood of evidence.

**P(Y=k)** is the Prior Probability.

**P(X1)\*P(X2)\*P(Xn)** is the probability of the evidence.

Multinomial Naïve Bayes is used to predict multi-class classification. It considers a feature vector where a given term represents the number of times it appears or very often i.e., frequency. It has low computation cost, can work effectively on relatively large datasets. Also, for small sample sizes, it can outperform the most powerful alternatives. But it is very difficult to get the set of independent predictors for developing model.

* **Support Vector Machine:** Support vector machines are a set of supervised learning methods used for classification, regression, and outliers’ detection. All of these are common tasks in machine learning. A simple linear SVM classifier works by making a straight line between two classes. That means all of the data points on one side of the line will represent a category and the data points on the other side of the line will be put into a different category. This means there can be an infinite number of lines to choose from. The linear SVM algorithm better than some of the other algorithms, like k-nearest neighbors, is that it chooses the best line to classify your data points. It chooses the line that separates the data and is the furthest away from the closet data points as possible.

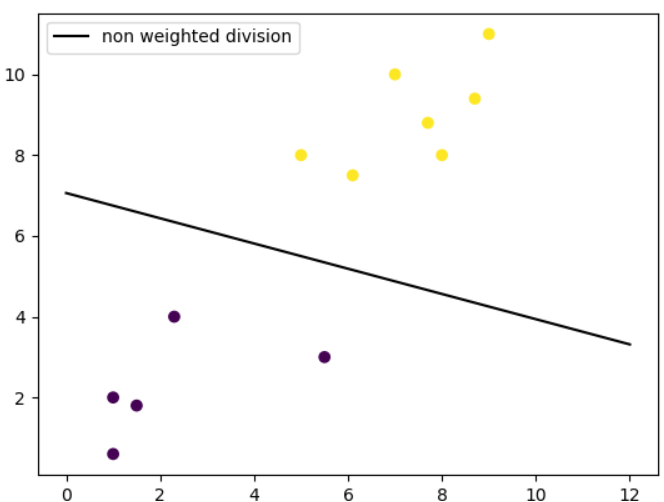


Fig 11. Linear SVM Fig 12. Non-linear SVM with RVF Kernel

SVM is Effective on datasets with multiple features or in cases where number of features is greater than number of data points

Train and test set has been standardized using StandardScaler from sklearn, and target variable has been log transformed. Let us plot the graphs of train and test set. It uses a subset of training points in the decision function called support vectors which makes it memory efficient. It has different kernel functions can be specified for the decision function. But SVMs don't directly provide probability estimates. Those are calculated using an expensive five-fold cross-validation.

Linear Kernel is commonly recommended for text classification because most of these types of classification problems are linearly separable. The linear kernel works really well when there are a lot of features, and text classification problems have a lot of features. Linear kernel functions are faster than most of the others and you have fewer parameters to optimize. The LinearSVC module from sklearn library provides support for linear support vector machine. The function for linear kernel is as below,

**f(X) = wT \* X + b**

* **Decision Tree:** A decision tree is a tree-like graph with nodes representing the place where we pick an attribute and ask a question; edges represent the answers the to the question; and the leaves represent the actual output or class label. They are used in non-linear decision making with simple linear decision surface. Decision trees classify the examples by sorting them down the tree from the root to some leaf node, with the leaf node providing the classification to the example. Each node in the tree acts as a test case for some attribute, and each edge descending from that node corresponds to one of the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new nodes. A general algorithm for a decision tree can be described as follows:
* Pick the best attribute/feature. The best attribute is one which best splits or separates the data.
* Ask the relevant question.
* Follow the answer path.
* Go to step 1 until you arrive to the answer.

The best split is one which separates two different labels into two sets. The best attribute is the one with the highest information gain. Information gain is a statistical property that measures how well a given attribute separates the training examples according to their target classification. To define information gain precisely, we need to define a measure commonly used in information theory called entropy that measures the level of impurity in a group of examples. It is defined as,



 Now, given entropy as a measure of the impurity in a sample of training examples, information gain can be defined as a measure of the effectiveness of an attribute in classifying the training data. Information gain, Gain (S, A) of an attribute A, relative to a sample of examples S, is defined as:

Decision Tree is easy to use and understand, can handle both categorical and numerical data, is resistant to outliers, hence require little data preprocessing. It can be used to build larger classifiers by using ensemble techniques. But it is very prone to overfitting and hyper parameter tuning is very sensitive to metrics. In case of high imbalanced dataset, it can create biased trees which is also a form of overfitting.

**Experimental Analysis:**

In our analysis NumPy, pandas, nltk, scikit-learn, imblearn, matplotlib, seaborn, plotly, wordcloud libraries were used for experiments. Tokenization, text preprocessing, removal of stop words and stemming were performed by nltk. Vectorization and classification were accomplished by scikit-learn. Imblearn package is used for class resampling. Matplotlib, wordcloud, seaborn and plotly express packages were used to plot graphs & visualizations.

Once the texts were preprocessed, it is then passed through pipeline. A pipeline is a linear sequence of data preparation options, modeling operations, and prediction transform operations. It allows the sequence of steps to be specified and evaluated. As an atomic unit, the pipeline can be evaluated using a preferred resampling scheme such as a train-test split or k-fold cross-validation. Pipeline is beneficial to prevent data leakage, and to add reproducibility.

Initially, we’ve split our data to 4:1 ratio, this would separate a set of records to validate on pipelines. Four different pipelines were created, all of which has the predefined first step of text vectorization using BOW. To implement BOW technique, CountVectorizer was used. Next to it, two of our pipelines has an additional step of downsampling and oversampling respectively, other two doesn’t feature this step. Those pipelines were trained on imbalanced dataset, we will discuss about all the metrics obtained below, before that the last step of all pipelines was, classification with ML Algorithms discussed earlier. Once the all the steps were defined, the data are split using ShuffleSplit, which will randomly sample entire training dataset during each iteration to divide it further into train and test, which are to be used for cross validation using HalvingGridSearch supported by scikit-learn. Halving grid search over specified parameter values with successive halving. The search strategy starts evaluating all the candidates with a small number of resources and iteratively selects the best candidates, using more and more resources, thus results in most similar result with impressive lesser time than its closest GridSearchCV.

Different metrics were used to evaluate the performance of our pipelines.

* **Accuracy:** Accuracy is the quintessential classification metric. Accuracy is the proportion of true results among the total number of cases examined. It is easily suited for binary as well as a multiclass classification problem. Accuracy is a valid choice of evaluation for classification problems which are well balanced and not skewed or no/less class imbalance.

**Accuracy = (TP + TN) / (TP + TN + FP + FN)**

* **Precision (Macro):** Precision answers what proportion of predicted positives is truly positives. Precision is a valid choice of evaluation metric when we want to be very sure of our prediction.

**Precision Macro = (Sum of Precision for each individual class) / (No. of Classes)**

* **Recall (Macro):** Recall answers what proportion of actual Positives is correctly classified. Recall is a valid choice of evaluation metric when we want to capture as many positives as possible.

**Recall Macro = (Sum of Recall for each individual class) / (No. of Classes)**

* **F1 Score:** The F1 score is a number between 0 and 1 and is the harmonic mean of precision and recall. F1 score sort of maintains a balance between the precision and recall for classifier. If precision is low, the F1 is low and if the recall is low again F1 score is low. The F1 score manages the tradeoff.

**F1 Score = 2 \* (Precision \* Recall) / (Precision + Recall)**

* **AUC:** AUC is the area under the ROC curve. AUC ROC indicates how well the probabilities from the positive classes are separated from the negative classes. AUC is scale-invariant. It measures how well predictions are ranked, rather than their absolute values.

As specified earlier, 4 different pipelines were created. Let us look through those briefly.

1. **Pipeline created for 5 classes with no resampling done:**

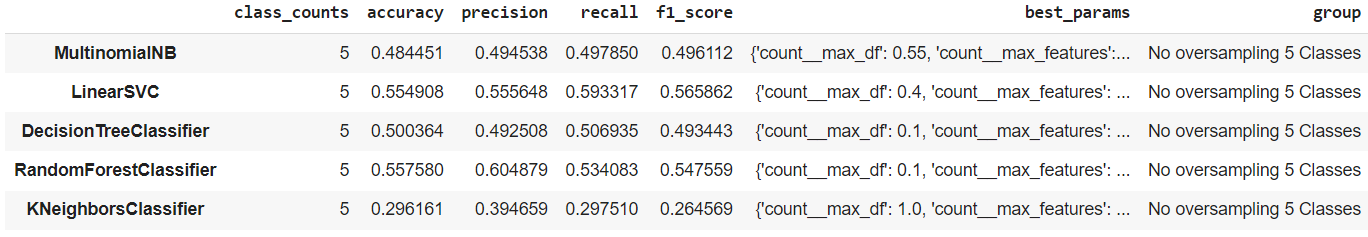
Steps involved are, vectorization (BOW) -> classification (shuffle split, halving grid search cv). 5 different algorithms have been evaluated in this pipeline. The results of classification are described as below,

Fig 13. Metrics for Imbalanced dataset (5 classes)

LinearSVC achieves highest performance (f1 score 0.56) followed by RandomForestClassifier (f1 score 0.547) but at a relatively lower computation cost. Though none of the results are satisfactory event with highest level of hyper parameter tuning. The reason behind this could be higher relevance between a class and its “Extreme” neighbor.

1. **Pipeline created for 3 classes with no resampling done:**

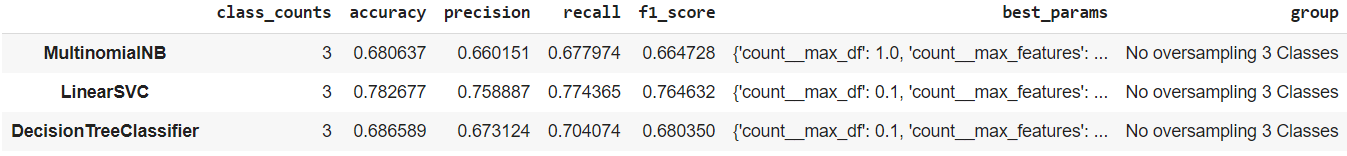
****Steps involved are, vectorization (BOW) -> classification (shuffle split, halving grid search cv). All the classes and its “Extreme” neighbor has been merged together. 3 different algorithms have been evaluated in this pipeline. The results of classification are described as below,

Fig 14. Metrics for Imbalanced dataset (3 classes)

A jump in performance can be seen, LinearSVC tops the score table with 0.76 f1 score

1. **Pipeline create for 3 classes with down sampling done:**

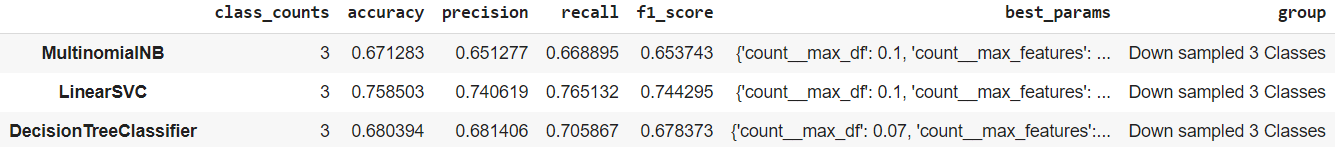
****Steps involved are, vectorization (BOW) -> Down Sampling (imblearn random down sampler) -> classification (shuffle split, halving grid search cv). The results of classification are described as below,

Fig 15. Metrics for Down sampled dataset (3 classes)

No impressive change can be observed but a slight decrease in scoring for under sampled pipeline, possibly owing to the fact that the resulting models cannot take full advantage of the whole training material available.

1. **Pipeline created for 3 classes with over sampling done:**

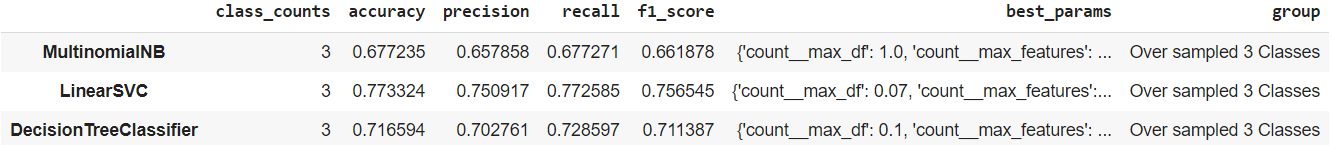
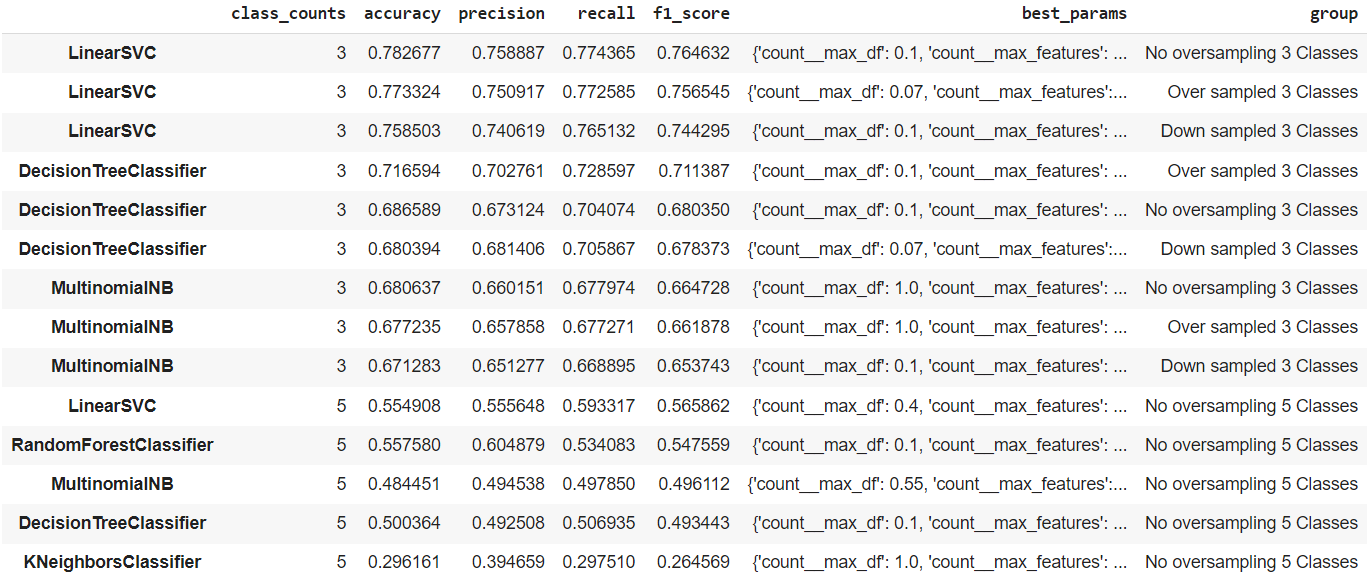
Steps involved are, vectorization (BOW) -> Over Sampling (imblearn random over sampler) -> classification (shuffle split, halving grid search cv). The results of classification are described as below,

Fig 16. Metrics for Over sampled dataset (3 classes)

DecisionTreeClassifier shows highest increase in performance. But other algo shows almost similar performance with no improvements at all. No improvements in performances can be explained by, there might have some bias attached with current working dataset.

We’ve trained a total of 14 models includes no resampled, down sampled and over sampled dataset. Below is the list of models’ performance ascending by their performance on the scale of f1 score.

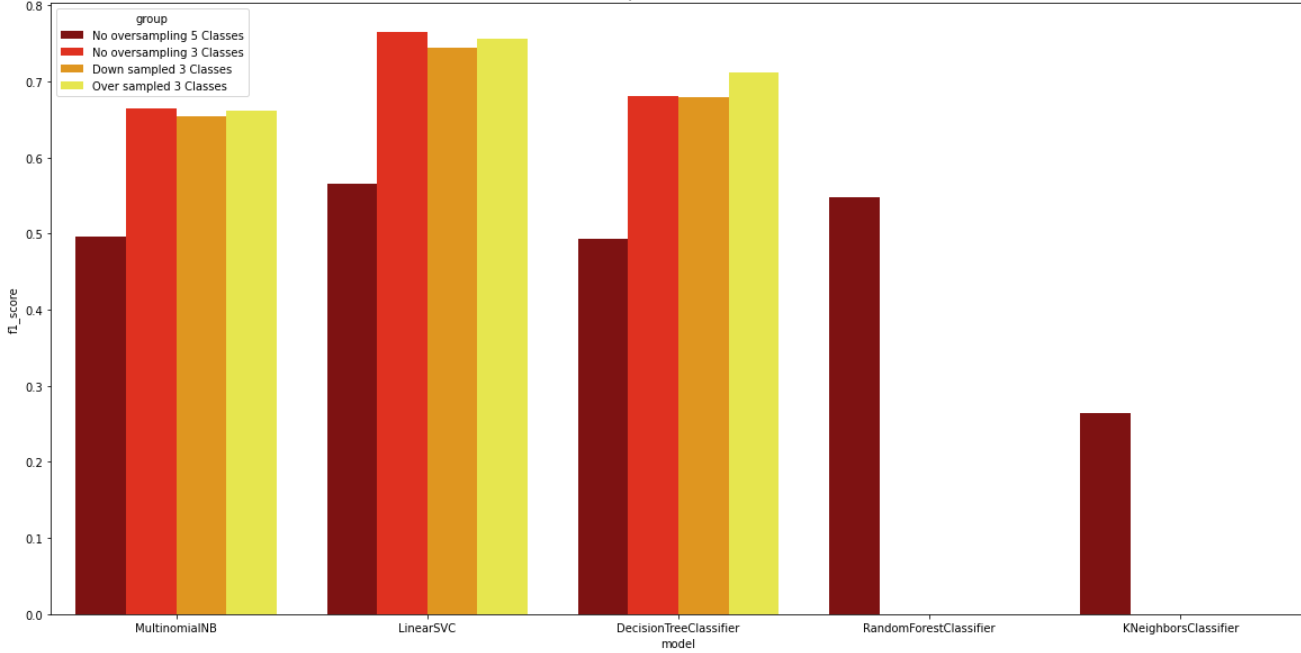
Fig 17. List of all evaluated metrics on all models

Fig 18. Comparison of F1 Score

Highest performance achieved by LinearSVC on imbalanced dataset with 3 classes. All the top 3 spots are hold by LinearSVC followed by Decision Trees ranked 4th on over sampled dataset. The poor performance behind Kneighbors could be higher level of dimensions.

**Model Selection:**

Final model has been built on LinearSVC, with hyper parameter tuning done on both bag of words and ml model. The parameters are,

**1. CountVectorizer:** max\_df = 0.7 ; max\_features = 7500

**2. LinearSVC:** multi\_class = ovr ; class\_weight = balanced ; max\_iter = 100000, C = 0.1

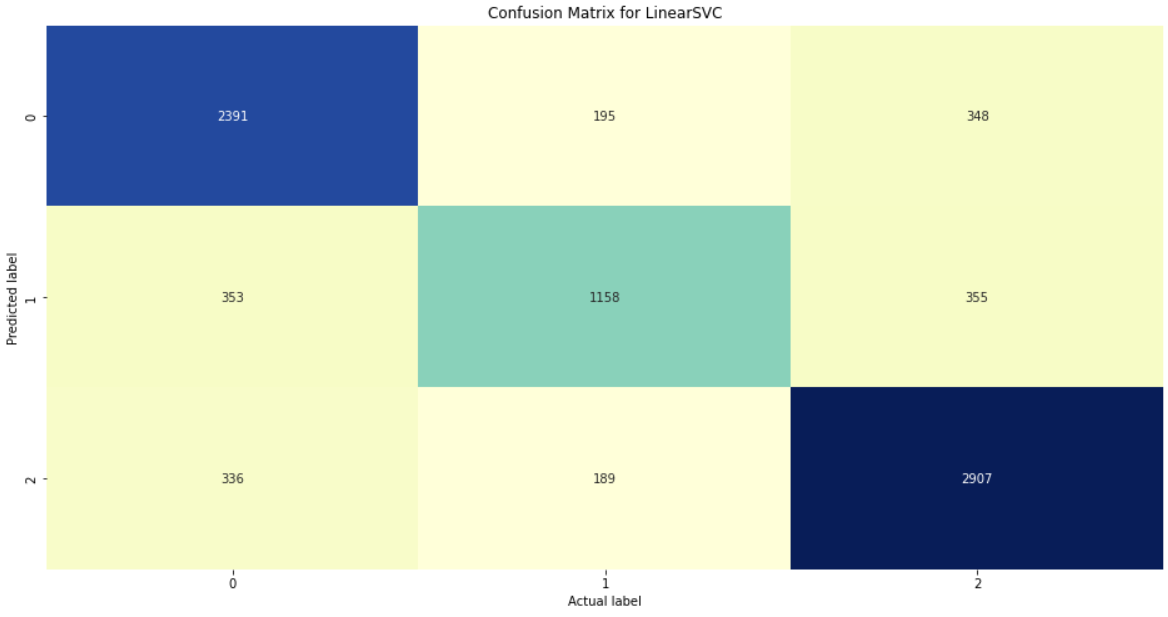
**Confusion Matrix:**

Fig 19. Confusion Matrix

**Classification Report:**



Fig 20. Classification Report

**Balanced Accuracy Score: 0.7776**

**Average Weighted F1 Score: 0.7868**

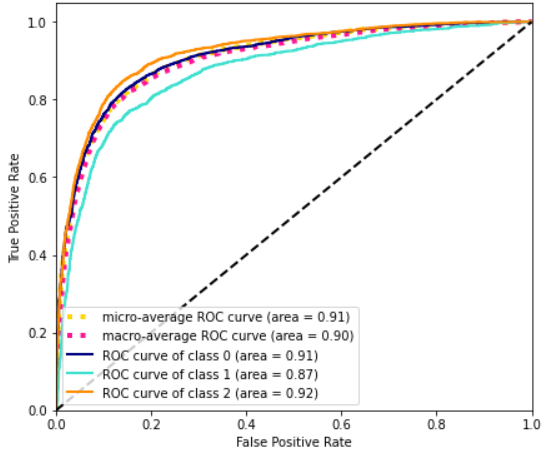
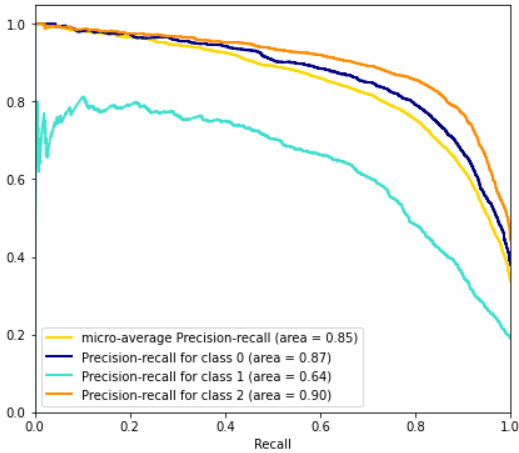
**ROC AUC Graph:**

Fig 21. Extension of ROC to multi-class Fig 22. Extension of Precision-Recall to multi class

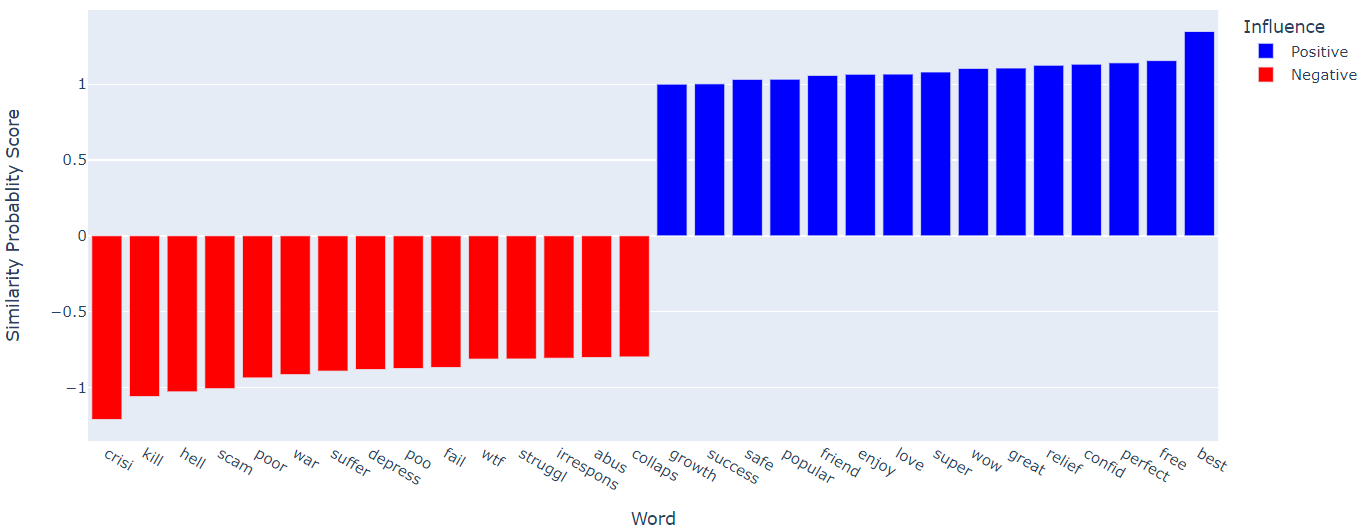
**Top 15 Influential Words:**

Fig 23. Top 15 Influential Words

**Limitation:**

As a multi-class classification problem with imbalanced dataset, we can assume dataset has some bias added within it. Also, we’ve some limited no of vocabulary available in terms of records, as our dataset focused on covid related tweets only Model performance will degrade while analyzing sentiment out of its known context or topic.

On trying some ensemble techniques, the computation cost and time hold us back as we’re trying this experiment on google colab runtime.

**Scope of Improvement:**

In general, we’ve huge scope of improvements. We could try to add more data to our set to generalize and also can use this from covid related to some healthcare field sentiment analysis easily. Moreover, we should balance the data, balancing could result in significant performance improvement. Also, this could be a tough, but we should check for document misclassification, if not document level, then at least word level, we can choose top importance significant number of words and analyze if the same word looks odd within its specified document label.

We’ve used BOW methods to vectorize the text, we can try tf-idf methods too. Some advanced complex word embedding techniques can also be applied like Glove, doc2vec but those work greatly on larger document corpus, we don’t think with such limited data we’ve those complex methods could perform any better. In terms of modelling, only standard ml algorithms been used in our analysis. Some complex high computation ensemble algorithms like XGBoost, BaggingClassifier, ExtraTreesClassifier can also be tried. We can also move on to try the same analysis with deep learning methods like LSTM etc.

**Conclusion:**

We focused on sentiment analysis for sentence labelling. We described the preprocessing steps, pipeline steps within which text normalization and model cross validation is included, performance has been measured using balanced accuracy, f1 score etc. We used “Stemming” instead of Lemmatization to reduce dimensions, for the same reason we haven’t tried tf-idf or term frequency vectorizer. We concentrated on feeding our model with word count information. Though we can assume bias is present in our dataset, but as we’ve used BOW, we think we haven’t triggered it. Some notable performance increase can be observed when class is reduced from 5 to 3. We assume, in case of binary classification we can further improve this score.

**References:**

* <https://towardsdatascience.com/comprehensive-guide-to-multiclass-classification-with-sklearn-127cc500f362>
* <https://www.techscience.com/cmc/v69n1/42767/html>
* <https://www.kaggle.com/code/venkatkrishnan/nlp-multiclass-onevsrestclassifier/notebook>
* <https://www.kaggle.com/code/venkatkrishnan/nlp-multiclass-onevsrestclassifier/notebook>
* <https://medium.com/@aneesha/visualising-top-features-in-linear-svm-with-scikit-learn-and-matplotlib-3454ab18a14d>
* <http://michael-harmon.com/blog/NLP1.html>

THE END