**Link of the Dataset ->** <https://www.kaggle.com/datasets/eswarchandt/amazon-music-reviews>

Kaggle is an online community for data scientists and machine learning practioners.

**DATASET INFO:**

Web portals like Amazon get vast amount of feedback from the users. To go through all the feedback's can be a tedious job. You have to categorize opinions expressed in feedback forums. This can be utilized for feedback management system. The classification of individual comments and reviews is done to get the overall rating based on individual comments/reviews. So that company can get a complete idea on feedback's provided by customers and can take care on those particular fields. This makes more loyal Customers to the company, increase in business, fame, brand value, profits.

Content

This file has reviewer ID, User ID, Reviewer Name, Reviewer text, helpful, Summary (obtained from Reviewer text). Overall Rating on a scale 5, Review time  
Description of columns in the file:

1. reviewerID - ID of the reviewer, e.g. A2SUAM1J3GNN3B
2. asin - ID of the product, e.g. 0000013714
3. reviewerName - name of the reviewer
4. helpful - helpfulness rating of the review, e.g. 2/3
5. reviewText - text of the review
6. overall - rating of the product
7. summary - summary of the review
8. unixReviewTime - time of the review (unix time)
9. reviewTime - time of the review (raw)

**#AIM OF THE PROJECT:**

The aim of this project is to help the organization better about their customer feedback's So that they can concentrate on those issues customers are facing.

**Task :1**  
You have to categorize opinions expressed in feedback forums  
**Task :2**  
You have to classify individual comments/reviews and you have to determine overall rating based on individual  
comments/reviews.

Various data analysis libraries are used for Data analysis such as numPy, Pandas, Seaborn, etc.

Then, some Exploratory Data Analysis is done on order to get an idea of the frequency of occurrence of the classes of reviews in which they are classified, and also since this was an imbalanced dataset, SMOTE techniques are used for resampling for improving the accuracies. In the data pre-processing phase, the dependent categorical variables are LabelEncoded into numerical values. Text cleaning and Stemming is done in order to keep only the relevant words for applying NLP techniques. This cleaned text is stored into a list. Since, using CountVectorizer leads to only count the number of times a word appears in the document which results in biasing in favour of most frequent words, this ends up in ignoring rare words which could have helped in processing our data more efficiently. So, TfidfVectorizer is used to consider the overall document weightage of a word. It helps in dealing with the most frequent words. Using this, those words can be penalized since TfidifVectorizer weights the word counts by a measure of how often they appear in the document.

**SMOTE(used for resampling):**

The process of applying SMOTE is basically done to do random-oversampling of the minority classes until a Balancing ratio of 1 is achieved

BL = X(minority)/X(majority)

Random-oversampling has a drawback that it may lead to overfitting of the model, since the model may think that all the samples are the same or alternatively, they may belong to the other class.

Therefore, SMOTE is a much better preferred method, In this, the minority class is oversampled by creating synthetic data-points instead of oversampling the datapoints at random. New observation from the minority class will not be identical to the original ones.

Internal Working of SMOTE:

The algorithm works by selecting a data-point at random and then computing it‘s k-nearest neighbours it determines the distance of this selected point from its k nearest neighbours and finally interpolates new\_data points along these distance.

-SMOTE-NC could also have been used since it works for categorical variables

**# PROCESS OF EXECUTION, AND STEPS WHILE APPLYING NLP TECHNIQUES:**

1. **Text Pre-processing:-**

The process of text pre - processing is done in two phases.

The 1 phase includes the cleaning of the raw text data, which is done by applying various techniques such as stemming, removing the stopwords, and converting all the text into a lower case

1. The first step includes the tokenization of the string(sentiments) into words for the machine to understand the words to classify them.
2. Stopwords are removed from the reviews to exclude unnecessary words. (NLTK provides a list or a custom list of stopwords could have been created)
3. Stemming:- It is a process of finding the base of a word – really fast(may not have any meaning). Therefore, to reduce the max\_features(top n highest number of times) of the Bag-of-words model this process is done. By reducing the features of the bag-of-words model the computational power can become less, and more accuracy can also be achieved. Another advantage is that unnecessary words can be removed.
4. To overcome the process of Stemming, Lemmatization could have been done. Through this, meaningful words could have been extracted but the process is very slow since it has to do a lot of comparisons.
5. In the next step, the words are converted into vectors (these are basically the numerical values of the words which the machine can understand) - This can be achieved using One-Hot Encoding (words are converted into word vectors based on the vocabulary – (advantage is that it is easy to implement, but one disadvantage is that it creates a sparse matrix – another disadvantage is that the size is not matching)) – The main disadvantage of a sparse matrix is that it contains a lot less information than a non-sparse matrix, since a vast majority of it’s entries must be equal to 0, not everything can be modelled by a sparse matrix.
6. For converting the cleaned-text data into numerical features – term frequency-inverse document frequency is used.

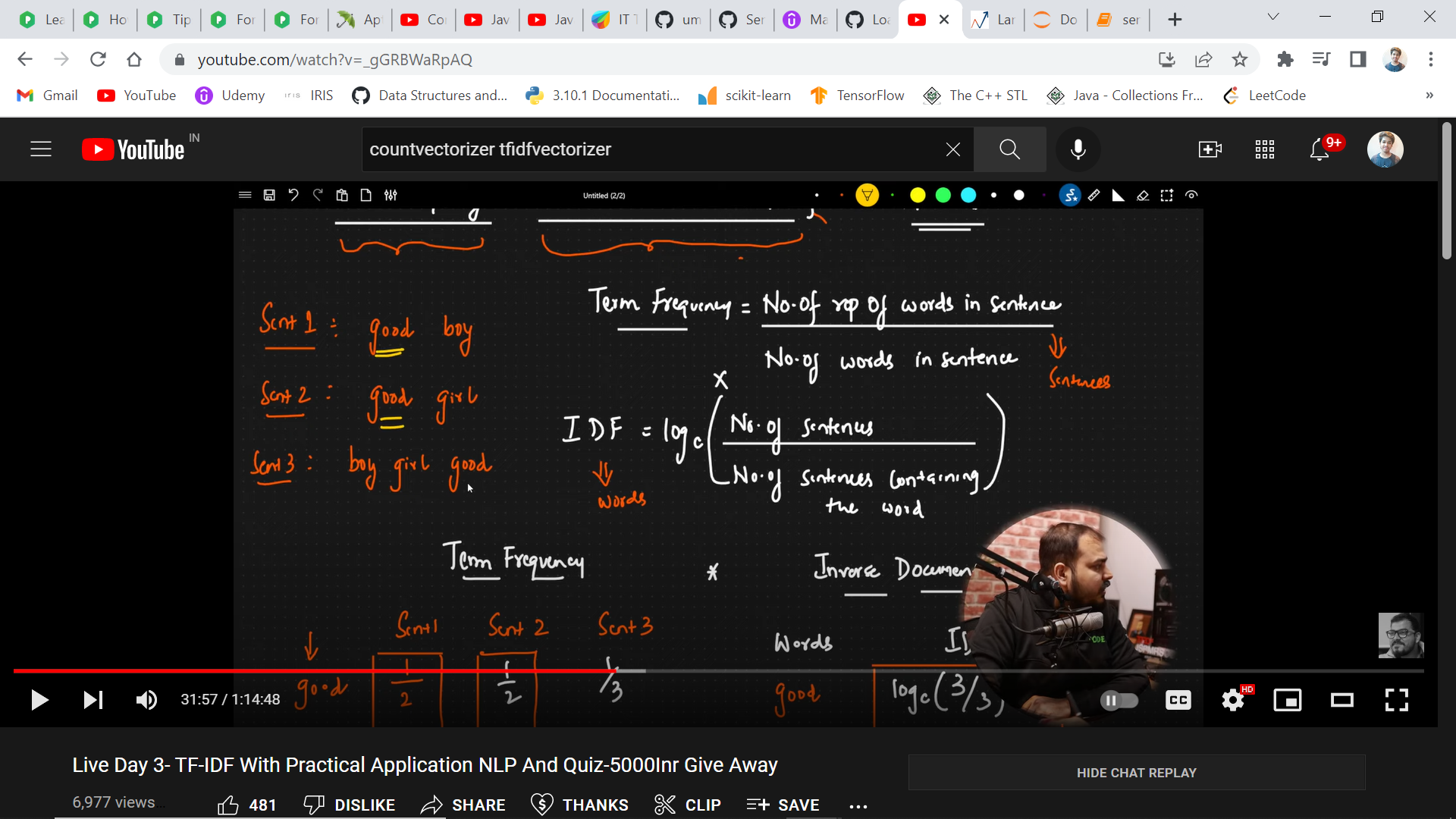
Term-frequency/inverse-document frequency signifies that, whichever words are present in the document less, should be given more weightage

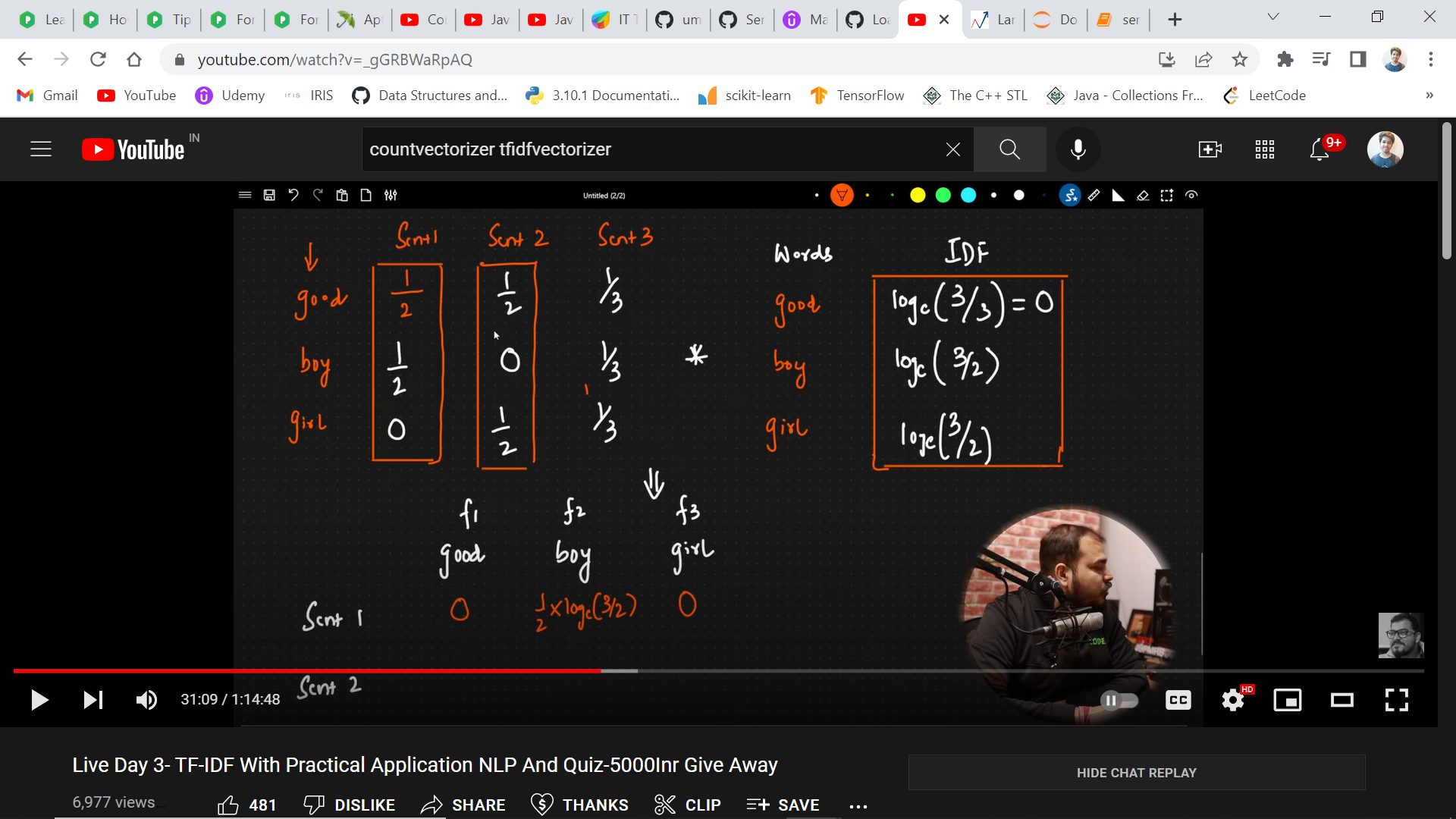
The rare-words will be captured by term-frequency and the words which are more common will be captured by inverse document frequency.

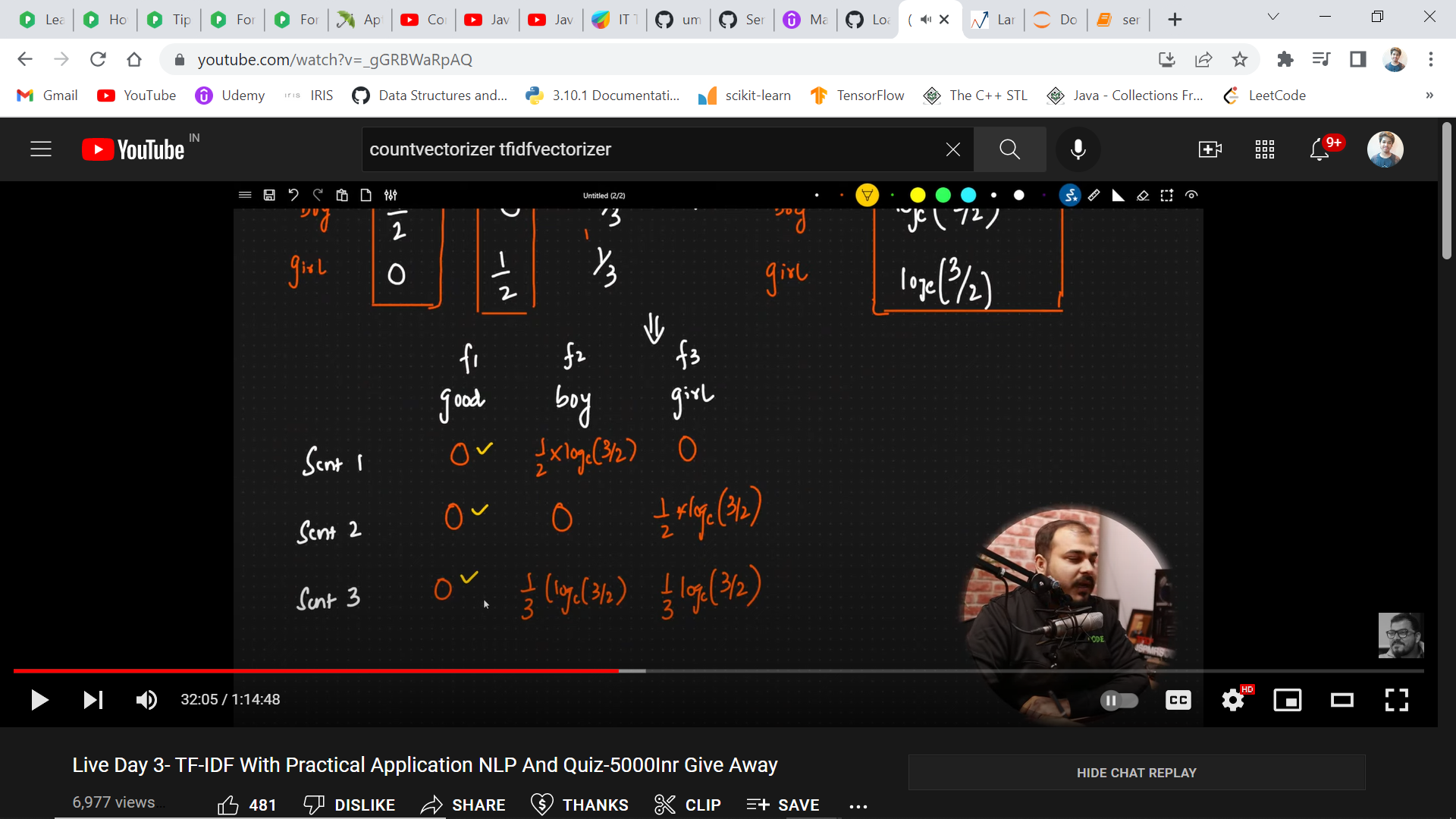
Term-Frequency = No. of repetition of words in sentence/No. of words in sentence

Inverse-Document Frequency = loge(No. of sentences/No sentences containing the word)

TF-IDF basically does a matrix multiplication of the TF, and IDF computed, and stores these values into the sparse matrix created therefore applying the bag-of-words model.







The Sparse Matrix becomes the entire dataset, and on this the model is trained

PS - (Lemmatization(sort so as to group together same forms of the word) could also have been applied which converts the stemmed word into it’s correct format)

The null values have been filled with ‘Empty’ in the data-preprocessing part.

Some other ways to handle missing data-

1. Deleting rows with missing data
2. Imputing missing values with continous features
3. Using only those algorithms which support missing values

##### 1. Logistic Regression is giving great results.

##### 2. Decision Tree was over - trained

##### 3. Random Forest gave great results

##### 4. KNN – which works well with imbalanced datasets, was performing the worst. (without resampling, KNN was performing much better with an accuracy of around 87% on the test set)

##### 5. SVC took too much time for training, because of the large dataset and a lot of features.

##### 6. MultinomialNB works well with features which assume vector values, therefore implemented MultinomialNB, other classes of naive bayes' performed poorly

##### 7. XG Boost was one of the best model on this dataset, although not preferred for NLP tasks.

XGBoost can be used when working with a large number of training samples, that means at least more than 1000 samples, and also when the features are very less.

**# Applying cross-validation to further evaluate the models in the most relevant way such that the training sample, doesn’t get lucky on a particular test set when predicting:**

Set the folds to 10, so the training set will be further sub divided into 10 samples of training, and test set, in which the models will be trained and simultaneously be evaluated on the testing set to compute the accuracies on these 10 folds, and further the mean of these accuracies will be calculated.

Standard deviation is also calculated to get an idea in which the training samples lie.

The combination of Hyperparameters will be evaluated through k-fold cross-validation and not on a single test-set, therefore, cv denotes the number of different train\_test folds when applying the k-fold cross validation for each of the combination.

**# Hyper-Parameter Tuning for further tuning each of the models to get the best version of each of the models :-**

Issues Faced:

1. Resampling Issues, model was becoming highly biased (basically overfitting), was not giving proper accuracy on the test set.
2. KNN – which works well with imbalanced datasets, was performing the worst. (without resampling, KNN was performing much better with an accuracy of around 87% on the test set)
3. Also, I did resampling after train\_test\_split, so that the test data doesn’t have duplicated data.
4. I used TfidfVectorizer, since because of CountVectorizer, the result was getting highly biased (overfitting).
5. SVC was taking too much time for training, I could have reduced the training data, but didn’t want to reduce the important features.
6. To overcome the problem of oversampling, means whether it should be done before or after train\_test\_split, I used SMOTE which doesn’t have the limitations which RandomOver–sampling has, means that It also prevents duplication, new observations from the minority class will not be identical to original ones.
7. I tried the SMOTE after the train\_test\_split, but it wasn’t giving better results, compared to the when I applied before train\_test\_split.
8. KNN was giving, much better accuracy this time.
9. Finally, I decided to do resampling before the train\_test\_split, since the performance was much better.
10. Was trying with GaussianNB, but since MultinomialNB works well with features which assume vector values, therefore implemented MultiNB.
11. While performing k-fold cross validation, I have set n\_jobs = -1 since was taking too much time to compute the accuracies, and same in the case of gridsearchCV.
12. It's a linear classification that supports logistic regression and linear support vector machines. The solver uses **a Coordinate Descent (CD) algorithm** that solves optimization problems by successively performing approximate minimization along coordinate directions or coordinate hyperplanes. 🡪 solver(in Log Reg)
13. **C*float, default=1.0***

Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

1. **Penalty *{‘l1’, ‘l2’, ‘elasticnet’, ‘none’}, default=’l2’***

Specify the norm of the penalty:

* 'none': no penalty is added;
* 'l2': add a L2 penalty term and it is the default choice;
* 'l1': add a L1 penalty term;
* 'elasticnet': both L1 and L2 penalty terms are added.

**Warning**

Some penalties may not work with some solvers. See the parameter solver below, to know the compatibility between the penalty and solver.

1. Each time the different accuracies of the model is computed in GridSearchCV, it is computed through k-fold cross validation.
2. XG Boost

gamma [default=0, alias: min\_split\_loss]

* 1. Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be.
  2. range: [0,∞]

booster [default= gbtree ]

* 1. Which booster to use. Can be gbtree, gblinear or dart; gbtree and dart use tree based models while gblinear uses linear functions.

1. D\_tree = **criterion*{“gini”, “entropy”}, default=”gini” – I tried changing the max\_depth parameter, but that led to underfitting.***

In decision trees, over-fitting occurs when the tree is designed so as to perfectly fit all samples in the training data set. Thus it ends up with branches with strict rules of sparse data. Thus, this effects the accuracy when predicting samples that are not part of the training set.

One of the methods used to address over-fitting in decision tree is called **pruning**which is done after the initial training is complete. In pruning, you trim off the branches of the tree, i.e., remove the decision nodes starting from the leaf node such that the overall accuracy is not disturbed. This is done by segregating the actual training set into two sets: training data set, D and validation data set, V. Prepare the decision tree using the segregated training data set, D. Then continue trimming the tree accordingly to optimize the accuracy of the validation data set, V.

The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

Gini impurity is an important measure used to construct the decision trees. Gini impurity is **a function that determines how well a decision tree was split**. Basically, it helps us to determine which splitter is best so that we can build a pure decision tree. Gini impurity ranges values from 0 to 0.5.

The information gained in the decision tree can be defined as **the amount of information improved in the nodes before splitting them for making further decisions**.

**max\_depth*int, default=None***

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.

1. R\_forrest = **n\_estimators*int, default=100 --> changing max\_depth led to underfitting***

The number of trees in the forest.

* Max\_depth = max\_depth = max number of levels in each decision tree

Random-forrest, and Decision-tree Regressor dosen’t require Feature-Scaling since the outcomes from these models are not based on some equations, but are come from the output of the nodes.

Decision-tree, and Random-forest works well with high-dimensional datasets.

Ensemble learning is very stable and powerful, since any changes in the dataset would lead to changing the outcome of decision trees, but to affect a forest of trees would be much harder.

1. MultinomialNB = **alpha*float, default=1.0***

Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).

Using Laplace smoothing, we can represent P(x'|positive) as, P(x'/positive)= **(number of reviews with x' and target\_outcome=positive + α) / (N+ α\*k)** Here, alpha(α) represents the smoothing parameter, K represents the dimensions(no of features) in the data, N represents the number of reviews with target\_outcome=positive.

1. KNN (slow process) = **n\_neighbors *int, default=5 🡪 took a lot of time to tune the hyperparameters, therefore reduced the folds to 3. (tried for larger n also), reducing the n\_neighbours speed up the process of grid\_search***

Number of neighbors to use by default for **[kneighbors](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html" \l "sklearn.neighbors.KNeighborsClassifier.kneighbors" \o "sklearn.neighbors.KNeighborsClassifier.kneighbors)** queries.

**Algorithm *{‘auto’, ‘ball\_tree’, ‘kd\_tree’, ‘brute’}, default=’auto’***

Algorithm used to compute the nearest neighbors:

* ‘ball\_tree’ will use **[BallTree](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html" \l "sklearn.neighbors.BallTree" \o "sklearn.neighbors.BallTree)**
* ‘kd\_tree’ will use **[KDTree](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html" \l "sklearn.neighbors.KDTree" \o "sklearn.neighbors.KDTree)**
* ‘brute’ will use a brute-force search.
* ‘auto’ will attempt to decide the most appropriate algorithm based on the values passed to [**fit**](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier.fit) method.

1. SVC :

[{'C':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9], 'kernel':['linear']},

{'C':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9], 'kernel':['rbf'], 'gamma':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]} 🡪 tried for both linear and rbf kernel, but took a lot of time, therefore implemented both separately

Reduced cv to 5.