Toxic Comment Classification Challenge

Problem Statement - You are provided with a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The types of toxicity are toxic, severe_toxic, obscene, threat, insult and identity_hate.You must create a model which predicts a probability of each type of toxicity for each comment.

Dataset Description - We are given with a dataset having the following dataset description -

- ID's of comments
- Text Comments
- Six output columns corresponding to each output class having binary class as '0' & '1'.
- The output columns are as follows:
    - - toxic
    - - severe_toxic
    - - obscene
    - - threat
    - - insult
    - - identity_hate
- The dataset has 159571 comment text data and corresponding output values.

Understanding the Problem Statement - We are given with the commented text and the corresponding labels and using supervised machine learning models on the training data we should be able to correctly predict the probabilities of the given comment lying in a particular output class.

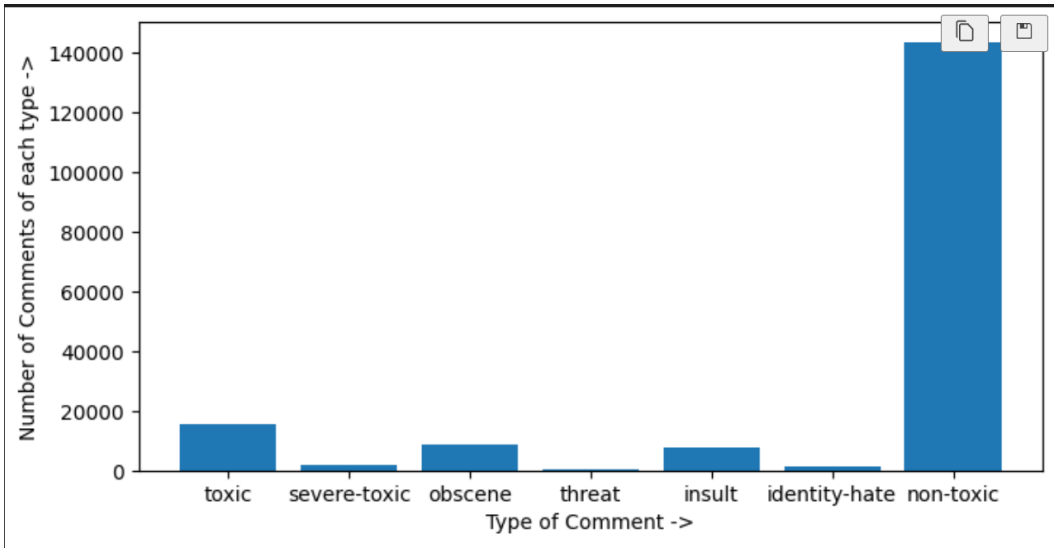
Proposed solution flow - On observing the dataset, we observe that we need to deal with the text data and to train the various models on it. To work further we can't directly continue with the given text data. So the workflow pipeline will be as follows:-

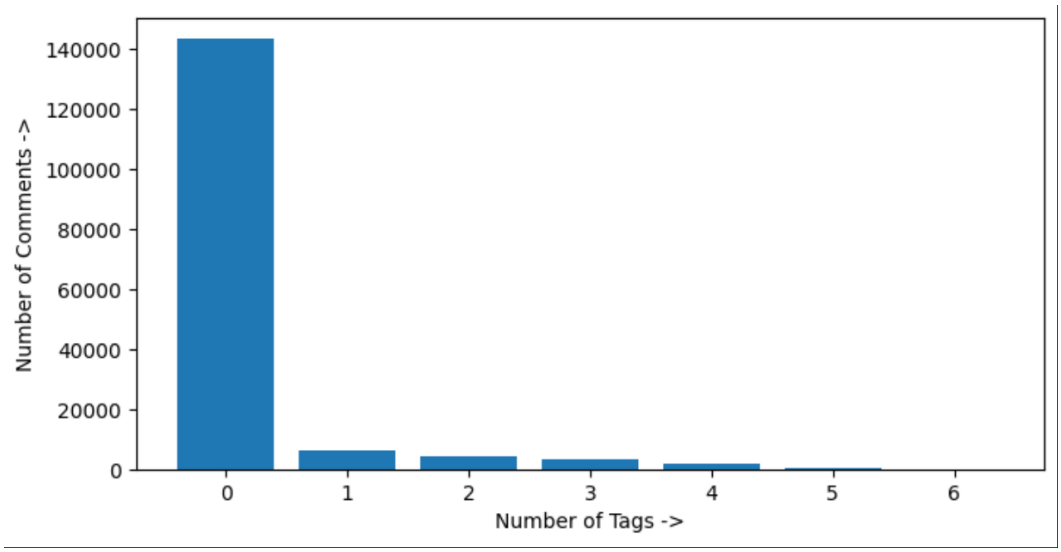
- ☐ Preprocess the text data.
- ☐ Balance out the whole data if required to remove biasness
- ☐ Train various ML models and get the evaluation reports
- ☐ Find the best suitable ML models and the best method to get the final predictions.
- ☐ Predict the probabilities of each comment to be in any particular class for test data.
- ☐ Create the final pipeline on the basis of a trained model so that any random comment can be classified.

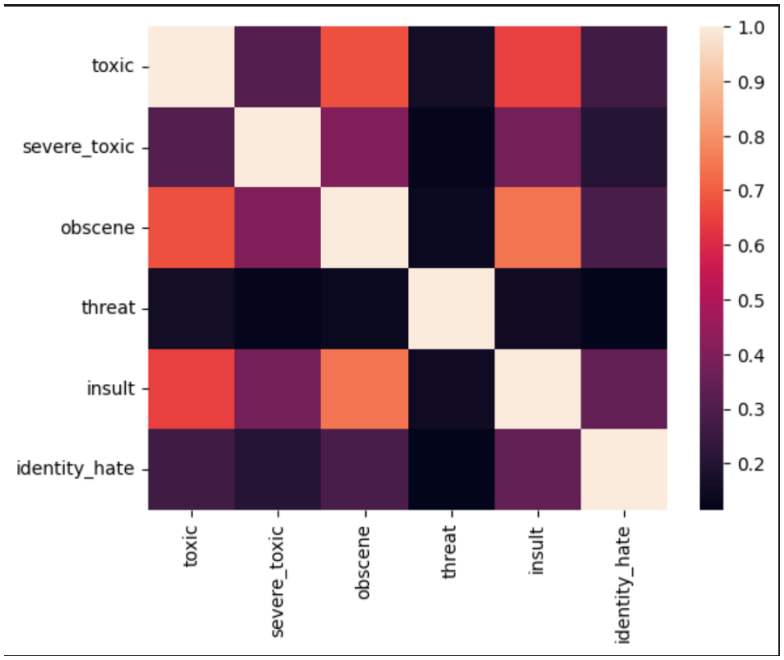
Visualizing and Analyzing the data

- We got the dataframe and checked for the **NULL values** in the complete dataframe and got the output that there are no NULL values in the given data.
- To analyze the data completely and get a complete idea about the dataframe, we obtained the counts for each output class of the number of ‘1’ values for a particular column which is the presence of that particular class output in the whole data.

```
Total comments =  159571
Total non_toxic comments =  143346
Count of each type of comment:
toxic            15294
severe_toxic      1595
obscene           8449
threat             478
insult            7877
identity_hate     1405
```


Number of Comments of each type ->
140000
120000
100000
80000
60000
40000
20000
0
toxic
severe-toxic
obscene
threat
insult
identity-hate
non-toxic
Type of Comment ->


Number of Comments ->
140000
120000
100000
80000
60000
40000
20000
0
0
1
2
3
4
5
6
Number of Tags ->



- We obtained the **confusion matrix** considering any two columns and obtained all the confusion matrices.

Analyze the comments

- From the given dataframe, we obtained the particular **column comments** belonging to each particular class, i.e class output is ‘1’.

Visualize the comments

- Using “World-Cloud”, a **visualization** method that displays how frequently words appear in a given body of text, by making the size of each word proportional to its frequency.

Frequent words in Non-toxic Comments

Frequent words in Toxic Comments

Frequent words in Severe-toxic Comments

Frequent words in Obscene Comments

Frequent words in Threat Comments

Frequent words in Insult Comments

Frequent words in Identity Hate Comments

Preprocessing

- **Cleaning Text -** First we have to clean our text comments so that our tokensization and vectorizer will work better, we have to remove the extra white spaces, punctuation signs, numbers, email addresses and all the unnecessary things in the text comments which is not affecting our classification.

- **Apply Tokenizer -** Tokenization is the process of breaking down a text document or a sentence into individual words or tokens. In other words, tokenization is a technique used to segment text into meaningful units called tokens. The tokens can be words, numbers, or punctuation marks. So we are applying the tokenizer on the text comments to get a tokenized set and further proceed with different cleaning techniques.

- **Removing stop words -** Stop words like "the", "a", "an", "and", "or", "of", "in" are removed from the tokenized data obtained using “nltk”, i.e. words in corpus of nltk.

Balancing out the data

We are generating a balanced dataset by augmenting the minority class in the original dataset. It follows the following steps:

1. Splits the original dataset into a training and testing set, with 80% of the data used for training and 20% for testing.
2. For each output column (toxic, severe_toxic, obscene, threat, insult, identity_hate), the code identifies the minority class and the number of samples in that class.
3. It calculates the number of samples needed to be added to balance the dataset, which is the difference between the number of samples in the minority and majority class divided by the number of output columns (to ensure a balanced dataset across all output columns).
4. The code generates augmented data for the minority class by randomly selecting samples from the majority class and flipping the output value for the minority class.
5. The augmented data is appended to the training set.
6. If the size of the training set exceeds the target size, the code randomly samples a subset of the data to reduce it to the target size.
7. The balanced dataset is saved to a new file and returned as a pandas dataframe.
8. Note that this code assumes that the input dataset is already preprocessed (e.g., tokenized, stop words removed, and vectorized). It only focuses on balancing the dataset by augmenting the minority class. And thus we are providing the dataframe to the code after dropping the other useful columns for re-sampling.

Vectorising the text data

Vectorizing text data refers to the process of converting raw text data into numerical form so that it can be processed by machine learning algorithms. We have used “TfidfVectorizer” which takes in a corpus of documents and generates a matrix of TF-IDF features, where each row represents a document and each column represents a word in the vocabulary. The vectorizer will give us the numerical features out of our text data, so that we can train our models on these numerical features.

Train-Test Split for final evaluation of the model

We now have obtained the two dataframes finally. The first one is the original dataframe and the second one is the data frame created by balancing and reducing the original dataframe.

Codeflow afterwards

Now we are applying different machine learning models on the obtained data frames and further on the basis of the evaluation reports provided by standard sklearn library, we will make the final decision for which we will go with the final dataframe and models for getting final predictions.

Function for models training and testing and saving:

Here we have used many models for classification for each output column, so for each output column we have called the particular model function to train the particular model on each output column using a loop the function to call the respective function and to train the model on each column. We

used the “pickle.dump” function to save the trained model by giving a unique string address which is generated by our function. We have also used the class weights parameter for increasing the recall of our model. This function will also give a classification report including precision, recall, accuracy on the test data which we are taking by using test train split on the original train.csv file.
And we have taken the hyperparameters of some models from a research paper*.

We made the this function for each model which includes:

- Random Forest
- Logistic Regression
- Support Vector Machine (SVM)
- K Nearest Neighbors (KNN)
- Gradient Boosting Classifier (GBM)
- Quadratic Discriminant Analysis (QDA)
- XGBoost Classifier
- Light GBM Classifier

Making the Models:

We have obtained the 2D array of model names and short codes which individual array will contain the names of models which are giving the best results for that particular output column. So by using the same address of the models which we saved earlier we called our saved models using the pickle.load() function.

The models for each output column that we have selected by analyzing the precision, recall, accuracy and f1-score are:

Severe -> Random Forest, Logistic Regression, Support Vector Machine (SVM), K Nearest Neighbors (KNN), XGBoost Classifier, Light GBM Classifier.

Severe_toxic -> Random Forest, Logistic Regression, Support Vector Machine (SVM), XGBoost Classifier, Light GBM Classifier.

Obscene -> Logistic Regression, Support Vector Machine (SVM), XGBoost Classifier, Light GBM Classifier.

Threat -> Logistic Regression, Support Vector Machine (SVM), XGBoost Classifier, Light GBM Classifier.

Insult -> Random Forest, Logistic Regression, Support Vector Machine (SVM), Gradient Boosting Classifier (GBM), XGBoost Classifier, Light GBM Classifier.

Indentity_hate -> Random Forest, Support Vector Machine (SVM), XGBoost Classifier.

Function to predict the probability of each output column

This function basically takes the **string input** and provides the **probability** of that particular comment text to be in a particular **targeted output column**.

We made a function pred_prob which takes a string input like as a comment on which we have to find the probability of each output column, and it is taking an output array obtained from the particular model for that particular output column which contains all the models which are the best. And then we have used the voting for all the outputs obtained for all the best models to find the probability of the particular column output.

For the prediction of the models, we have followed the standard preprocessing with text & get the predictions using the final trained models.

This function will give a single probability value of that particular column label, then we call this function for each column by passing the particular best models of this column and appending the probability to an array. Then find the output binary classes by using the probability we got, if probability is greater than equal to 0.5, then we will give the output for that particular output class ‘1’ otherwise 0.

Judgment on dataframes and its corresponding models

Now we have multiple dataframes and their corresponding trained models available. Now on the basis of the accuracy, precision, recall & f1 scores available for the models, we concluded to go further with the original data because the balanced data was synthetically created and may or not able to get the desired outputs every time.

Following are the evaluation results for various models on the augmented dataset :-

1)Random forest

94.13097936299654

	precision	recall	f1-score	support
0	0.95	0.98	0.97	7293
1	0.84	0.68	0.75	1090

94.13097936299654

	precision	recall	f1-score	support
0	0.96	0.97	0.97	7338
1	0.78	0.73	0.76	1045

91.36347369676727

	precision	recall	f1-score	support
0	0.94	0.96	0.95	7076
1	0.75	0.67	0.71	1307

91.32768698556603

	precision	recall	f1-score	support
0	0.95	0.95	0.95	7341
1	0.66	0.64	0.65	1042

87.39114875342956

	precision	recall	f1-score	support
0	0.93	0.92	0.93	7070
1	0.60	0.60	0.60	1313

82.6434450673983

	precision	recall	f1-score	support
0	0.93	0.86	0.90	7297
1	0.39	0.58	0.47	1086

1)Random forest

94.13097936299654

	precision	recall	f1-score	support
0	0.95	0.98	0.97	7293
1	0.84	0.68	0.75	1090

94.13097936299654

	precision	recall	f1-score	support
0	0.96	0.97	0.97	7338
1	0.78	0.73	0.76	1045

91.36347369676727

	precision	recall	f1-score	support
0	0.94	0.96	0.95	7076
1	0.75	0.67	0.71	1307

91.32768698556603

	precision	recall	f1-score	support
0	0.95	0.95	0.95	7341
1	0.66	0.64	0.65	1042

87.39114875342956

	precision	recall	f1-score	support
0	0.93	0.92	0.93	7070
1	0.60	0.60	0.60	1313

82.6434450673983

	precision	recall	f1-score	support
0	0.93	0.86	0.90	7297
1	0.39	0.58	0.47	1086

2) logistic regression

92.30455898480339

	precision	recall	f1-score	support
0	0.98	0.93	0.96	28882
1	0.56	0.84	0.68	3033

96.63481121729595

	precision	recall	f1-score	support
0	1.00	0.97	0.98	31591
1	0.22	0.87	0.35	324

96.07081309728967

	precision	recall	f1-score	support
0	0.99	0.97	0.98	30215
1	0.59	0.88	0.70	1700

98.74980416731944

	precision	recall	f1-score	support
0	1.00	0.99	0.99	31826
1	0.15	0.72	0.24	89

94.37568541438195

	precision	recall	f1-score	support
0	0.99	0.95	0.97	30346
1	0.46	0.87	0.60	1569

95.56948143506189

	precision	recall	f1-score	support
0	1.00	0.96	0.98	31619
1	0.15	0.79	0.25	296

3) SVM

81.59370153882858

	precision	recall	f1-score	support
0	0.96	0.82	0.89	7293
1	0.39	0.76	0.52	1090

70.4759632589765

	precision	recall	f1-score	support
0	0.94	0.71	0.81	7338
1	0.25	0.69	0.37	1045

71.16783967553381

	precision	recall	f1-score	support
0	0.92	0.72	0.81	7076
1	0.30	0.65	0.41	1307

65.83561970654897

	precision	recall	f1-score	support
0	0.93	0.66	0.77	7341
1	0.21	0.64	0.32	1042

67.57723965167601

	precision	recall	f1-score	support
0	0.90	0.69	0.78	7070
1	0.27	0.61	0.37	1313

62.089943934152444

	precision	recall	f1-score	support
0	0.91	0.63	0.74	7297
1	0.19	0.57	0.28	1086

```
4) knn
92.77455741814194
              precision    recall  f1-score   support

           0       0.93      0.99      0.96     28882
           1       0.79      0.33      0.46      3033

98.92527024909917
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     31591
           1       0.42      0.16      0.23       324

96.08021306595644
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     30215
           1       0.80      0.35      0.49      1700

99.70233432555224
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     31826
           1       0.38      0.10      0.16        89

95.97681341062196
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     30346
           1       0.70      0.32      0.44      1569

99.12266959110137
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     31619
           1       0.64      0.12      0.20       296
```

```
4) KNN

90.87438864368364
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      7293
           1       0.75      0.45      0.56      1090

92.92616008588811
              precision    recall  f1-score   support

           0       0.96      0.96      0.96      7338
           1       0.73      0.69      0.71      1045

89.01347966121914
              precision    recall  f1-score   support

           0       0.93      0.94      0.94      7076
           1       0.66      0.62      0.64      1307

89.97972086365263
              precision    recall  f1-score   support

           0       0.93      0.96      0.94      7341
           1       0.63      0.46      0.53      1042

85.24394608135512
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      7070
           1       0.54      0.39      0.45      1313

84.65942979840153
              precision    recall  f1-score   support

           0       0.90      0.92      0.91      7297
           1       0.39      0.33      0.36      1086
```

```
5)  XGB classifier
95.40968196772678
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     28859
           1       0.90      0.59      0.71      3056

98.98793670687765
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     31608
           1       0.44      0.20      0.27       307

97.84114052953157
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     30204
           1       0.88      0.69      0.78      1711

99.74620084599718
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     31828
           1       0.59      0.23      0.33        87

97.11107629641235
              precision    recall  f1-score   support

           0       0.98      0.99      0.98     30320
           1       0.81      0.55      0.66      1595

99.23546921510261
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     31650
           1       0.62      0.21      0.31       265
```

```
7) XGB

91.86448765358463
              precision    recall  f1-score   support

           0       0.92      0.99      0.96      7349
           1       0.87      0.40      0.55      1034
89.01347966121914
              precision    recall  f1-score   support

           0       0.89      1.00      0.94      7336
           1       0.94      0.13      0.23      1047

87.79673148037695
              precision    recall  f1-score   support

           0       0.88      0.99      0.93      7036
           1       0.88      0.28      0.42      1347

88.48860789693427
              precision    recall  f1-score   support

           0       0.89      1.00      0.94      7358
           1       0.83      0.07      0.13      1025

85.22008827388763
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      7038
           1       0.70      0.14      0.23      1345
87.31957533102708
              precision    recall  f1-score   support

           0       0.88      0.99      0.93      7340
           1       0.36      0.02      0.04      1043
```

```
6) LGBM
 93.56415478615071
              precision    recall  f1-score   support

           0       0.98      0.95      0.96     28860
           1       0.63      0.80      0.70      3055

96.36221212595957
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     31612
           1       0.19      0.88      0.31       303

96.550211499295
              precision    recall  f1-score   support

           0       0.99      0.97      0.98     30217
           1       0.63      0.87      0.73      1698

98.51167162776125
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     31815
           1       0.15      0.77      0.24       100

94.90835030549898
              precision    recall  f1-score   support

           0       0.99      0.95      0.97     30324
           1       0.49      0.84      0.62      1591

95.5851480495065
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     31644
           1       0.14      0.80      0.24       271
```

```
8) LGBM

89.7411427889777
              precision    recall  f1-score   support

           0       0.97      0.92      0.94      7349
           1       0.56      0.77      0.65      1034

87.68937134677323
              precision    recall  f1-score   support

           0       0.97      0.89      0.93      7336
           1       0.50      0.82      0.62      1047

85.98353811284743
              precision    recall  f1-score   support

           0       0.96      0.87      0.91      7036
           1       0.54      0.79      0.64      1347

84.46856733866159
              precision    recall  f1-score   support

           0       0.96      0.86      0.91      7358
           1       0.42      0.76      0.54      1025

80.67517595133008
              precision    recall  f1-score   support

           0       0.94      0.82      0.88      7038
           1       0.44      0.73      0.55      1345

77.5020875581534
              precision    recall  f1-score   support

           0       0.95      0.79      0.86      7340
           1       0.31      0.68      0.43      1043
```

```
7) Decision tree

93.56415478615071
              precision    recall  f1-score   support

           0       0.98      0.95      0.96     28860
           1       0.63      0.80      0.70      3055

96.36221212595957
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     31612
           1       0.19      0.88      0.31       303

96.550211499295
              precision    recall  f1-score   support

           0       0.99      0.97      0.98     30217
           1       0.63      0.87      0.73      1698

98.51167162776125
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     31815
           1       0.15      0.77      0.24       100

94.90835030549898
              precision    recall  f1-score   support

           0       0.99      0.95      0.97     30324
           1       0.49      0.84      0.62      1591

95.5851480495065
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     31644
           1       0.14      0.80      0.24       271
```

```
6)  Decision Tree Classifier
90.14672551592508
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      7312
           1       0.62      0.61      0.61      1071
86.85434808541095
              precision    recall  f1-score   support

           0       0.92      0.93      0.92      7305
           1       0.49      0.47      0.48      1078
86.55612549206728
              precision    recall  f1-score   support

           0       0.92      0.92      0.92      7054
           1       0.58      0.57      0.57      1329
85.81653346057497
              precision    recall  f1-score   support

           0       0.93      0.91      0.92      7391
           1       0.41      0.45      0.43       992
83.41882381009185
              precision    recall  f1-score   support

           0       0.89      0.91      0.90      7077
           1       0.46      0.41      0.44      1306
44.733389001550755
              precision    recall  f1-score   support

           0       0.95      0.39      0.55      7309
           1       0.17      0.86      0.28      1074
```

Final Output

Then we called the predict_prob function for the initial 1000 of the test.csv file to find the corresponding labels and probabilities of the test.csv comments because the test.csv file has around 1.5 lakh comments and it is taking very much time to predict all the 1.5 lakh comments, and then we made the final dataframe to show the output which includes our id column, comment_text column and the output label probabilities and classes. And then at the end we save our dataframe to the google drive to download my result.csv file to upload that.

Link for our model results → https://drive.google.com/drive/folders/1Z_-jV9Dz237ocM7lz2fjNiNIdFGb8Mws?usp=share_link

Individual Contributions:

Raunak Garg (B21EE056)

My work was mainly associated with coding the project. I wrote the functions from scratch for the model training, testing and saving them into the google drive.

Then I made the array of models for each column that are best according to the precision, recall and accuracy. I called the models by providing and generating the same string address which I used for training my models and saving them to that particular address for each model for each output column.

Then I wrote the function predict_proba() which will find the probabilities of each output column on the basis of voting of the best models for each column which I made as described above.

Then I wrote the code for finding the binary labels or classes by looking at the probabilities given by the predict_proba() function.

Then I wrote the code which will call my function for generating the probabilities and labels for the test.csv file comments, and then i made the final ‘result’ dataset for displaying the final output labels and probabilities and then saving the dataset into the google drive from where we can download the csv final result file.

I also worked on the visualization and preprocessing of the data using Wordcloud to visualize the common words more clearly.
I mainly worked on the coding part of the project, and the more research part is done by other team members.

Samarth Sudhirkumar Bhalerao (B21EE060)

I have worked on finding different methods to deal with the text data for further proceeding with it. Also I worked on making the synthetic data after looking at various ways to create the synthetic a way that the data has a significant number of output labels as ‘1’ & ‘0’ respectively.

I also worked on the part to deal with the dataframes having multiple multiple target column outputs and researching for the algorithm so that the data dealing and also new data creation is possible.

The main purpose behind the creation of the synthetic data is to make the model learn on significant unbiased data. I have used two methods to create the data, the first one is making an undersampled and equiproportionate data & the second one is to create a smaller similar version of the larger data using the k-means algorithm inside to create the similar structure & get the outputs.

Also I searched for one more nicer alternative to go with which was augmented data which created an overall dataframe in which the data was significantly balanced corresponding to the outputs.
I also worked on selecting the appropriate models & writing the codes for the models.
Also, after getting the models trained, I worked on what basis to judge and compare the model performance with others. Also implemented the same evaluation matrix and also made the observations and conclusions by taking a look at the output.
Simultaneously, I worked decisively to implement the selected models trained on the available data frames and finally make the judgment to decide whether which particular dataframe and simultaneously which particular model to go with the particular model.
I also worked on setting up the final pipeline of the code set and implementing the pipeline correctly.

Kartik Sanjay Narkhede (B21EE041)

I read a research paper about the toxic comment classification problem. From that I got information about how to handle string data and how to extract features from it. Using that information ,we cleaned the data .
We used that cleaned data for tokenizing and vectorizing it.I wrote code for that.

I also refer to google for removing biases of given dataset. I found methods like SMOTE , which oversamples / undersamples given dataset
To form a new dataset which is a balanced one. Also read very practical examples that are affected by data biasness .

Also I searched for another alternative to SMOTE which was augmented data which created an overall dataframe in which the data was significantly balanced corresponding to the outputs.

I referred to Natural Language Processing blogs for understanding how to work on string data , and provided that information to teammates.

I have worked on the model selection part of the pipeline , which is which model to use. I also researched different pre-processing techniques to deal with the string data and contributed with the preprocessing of the data, especially string data.

I also worked on making different models, making the code compatible for training and testing and then training the models on multiple training datasets available after synthetic creation and original ones and saving the models in google drive for further analysis and use .
Using that , we can easily call the trained model for predicting the output values.
I worked on the selected models trained on the available data frames and eventually made the decision on which specific dataframe and concurrently which particular model to proceed with the particular model.