FLIP ROBO

MALIGNANT COMMENTS CLASSIFIER PROJECT

Submitted by:

CHETHANA M

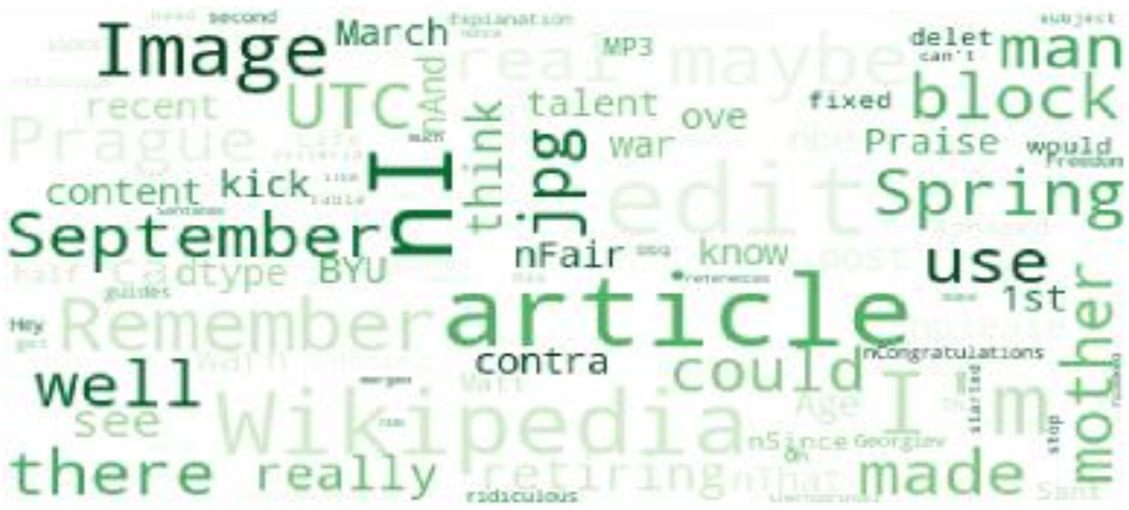
ACKNOWLEDGMENT

I express.my sincere gratitude to **Flip Robo Technologies** for giving me this opportunity to carry out the project work.

A special thanks to my mentor **Mohd Kashif** for guiding me in completing this project and being available to resolve my doubts whenever I raise any tickets.

I also would love to take this moment to thank **DataTrained** for giving me all the knowledge about how to build an effective machine learning model and providing this opportunity to work as intern in Flip Robo Technologies.

INTRODUCTION



The proliferation of social media enables people to express their opinions widely online. However, at the same time, this has resulted in the emergence of conflict and hate, making online environments uninviting for users. Although researchers have found that hate is a problem across multiple platforms, there is a lack of models for online hate detection.

Online hate, described as abusive language, aggression, cyberbullying, hatefulness and many others has been identified as a major threat on online social media platforms. Social media platforms are the most prominent grounds for such toxic behaviour.

There has been a remarkable increase in the cases of cyberbullying and trolls on various social media platforms. Many celebrities and influences are facing backlashes from people and have to come across hateful and offensive comments. This can take a toll on anyone and affect them mentally leading to depression, mental illness, self-hatred and suicidal thoughts.

Internet comments are bastions of hatred and vitriol. While online anonymity has provided a new outlet for aggression and hate speech, machine learning can be used to fight it. The problem we sought to solve was the tagging of internet comments that are aggressive towards other users. This means that insults to third parties such as celebrities will be tagged as unoffensive, but “u are an idiot” is clearly offensive.

Our goal is to build a prototype of online hate and abuse comment classifier which can used to classify hate and offensive comments so that it can be controlled and restricted from spreading hatred and cyberbullying.

Technical goals :

Given a number of comments, sentences or paragraphs being used as a comment by a user, our task is to identify the comment as whether it is a malignant comment or not. After that, when we have a collection of all the malignant comments, our main task is to classify the comments into one or more of the following categories

– Malignant, Highly Malignant, Rude, Threat, Abuse or Loathe. This problem thus comes under the category of multi-label classification problem.

Analytical Problem Framing

The data set contains the training set, which has approximately 1,59,000 samples and the test set which contains nearly 1,53,000 samples. All the data samples contain 8 fields which includes ‘Id’, ‘Comments’, ‘Malignant’, ‘Highly malignant’, ‘Rude’, ‘Threat’, ‘Abuse’ and ‘Loathe’.

The label can be either 0 or 1, where 0 denotes a NO while 1 denotes a YES. There are various comments which have multiple labels. The first attribute is a unique ID associated with each comment.

The data set includes:

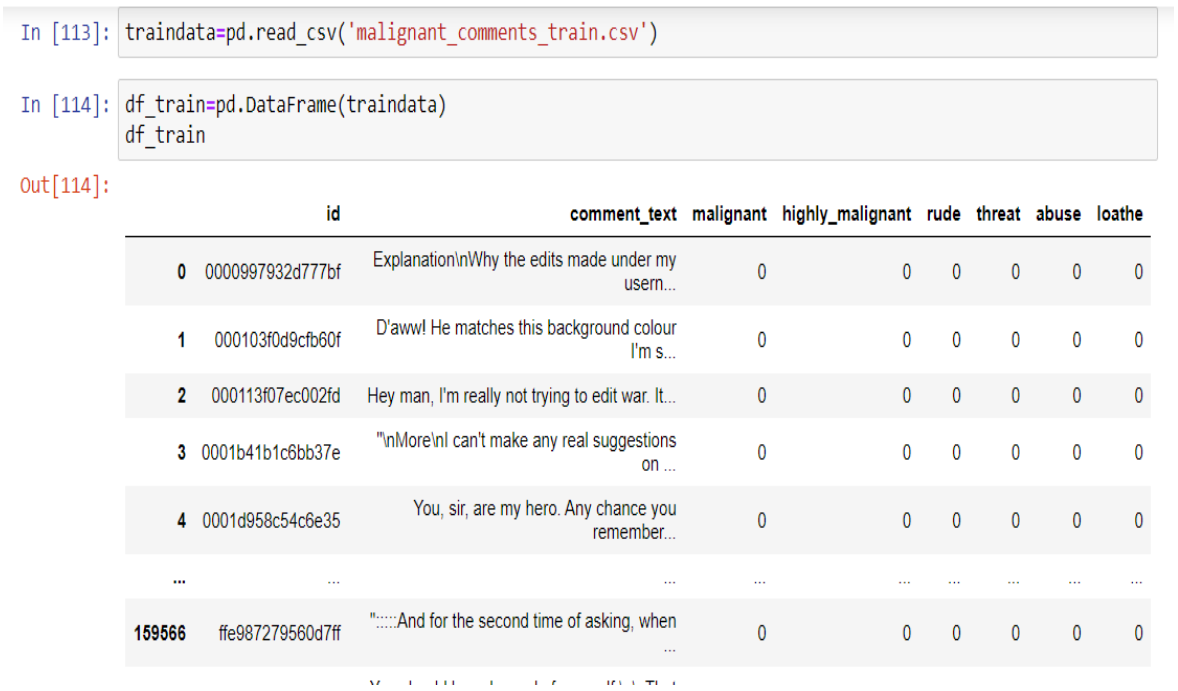
- Malignant: It is the Label column, which includes values 0 and 1, denoting if the comment is malignant or not.
- Highly Malignant: It denotes comments that are highly malignant and hurtful.
- Rude: It denotes comments that are very rude and offensive.
- Threat: It contains indication of the comments that are giving any threat to someone.
- Abuse: It is for comments that are abusive in nature.
- Loathe: It describes the comments which are hateful and loathing in nature.
- ID: It includes unique Ids associated with each comment text given.
- Comment text: This column contains the comments extracted from various social media platforms.

This project is more about exploration, feature engineering and classification that can be done on this data. Since the data set is huge and includes many categories of comments, we can do good amount of data exploration and derive some interesting features using the comments text column available.

Out of the above mentioned fields, the comment_text field will be preprocessed and fitted into the classifier to predict whether it belongs to one or more of the labels/outcome variables (i.e. ‘Malignant’, ‘Highly malignant’, ‘Rude’, ‘Threat’, ‘Abuse’ and ‘Loathe’).

We have 2 files – train.csv (the training set which contains comments with their binary labels) and test.csv (The test set for which the predictions are to be done. It includes id and comments_text)

We have a total of 159571 samples of comments and labelled data in train dataset, which can be loaded from train.csv file. The samples are as follows:



```
In [113]: traindata=pd.read_csv('malignant_comments_train.csv')
```

```
In [114]: df_train=pd.DataFrame(traindata)
          df_train
```

Out[114]:

	id	comment_text	malignant	highly_malignant	rude	threat	abuse	loathe
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
159566	ffe987279560d7ff	":::::And for the second time of asking, when ...	0	0	0	0	0	0

Process included :

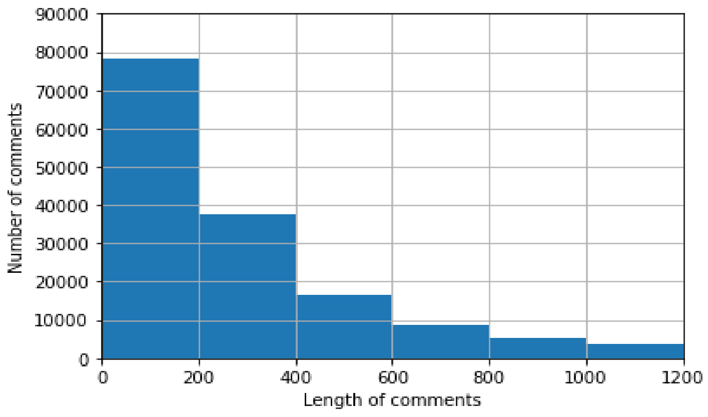
1. Calling the required libraries
2. Import necessary files
3. Read the train.csv file
4. EDA
5. Separate the comment field data and outcome labels
6. Data Visualisations
7. Remove excessive length comments
8. Preprocessing
9. Removing Punctuations and other special characters

    - Splitting the comments into individual words
    - Removing Stop Words
    - Stemming and Lemmatising
    - Applying Count Vectoriser

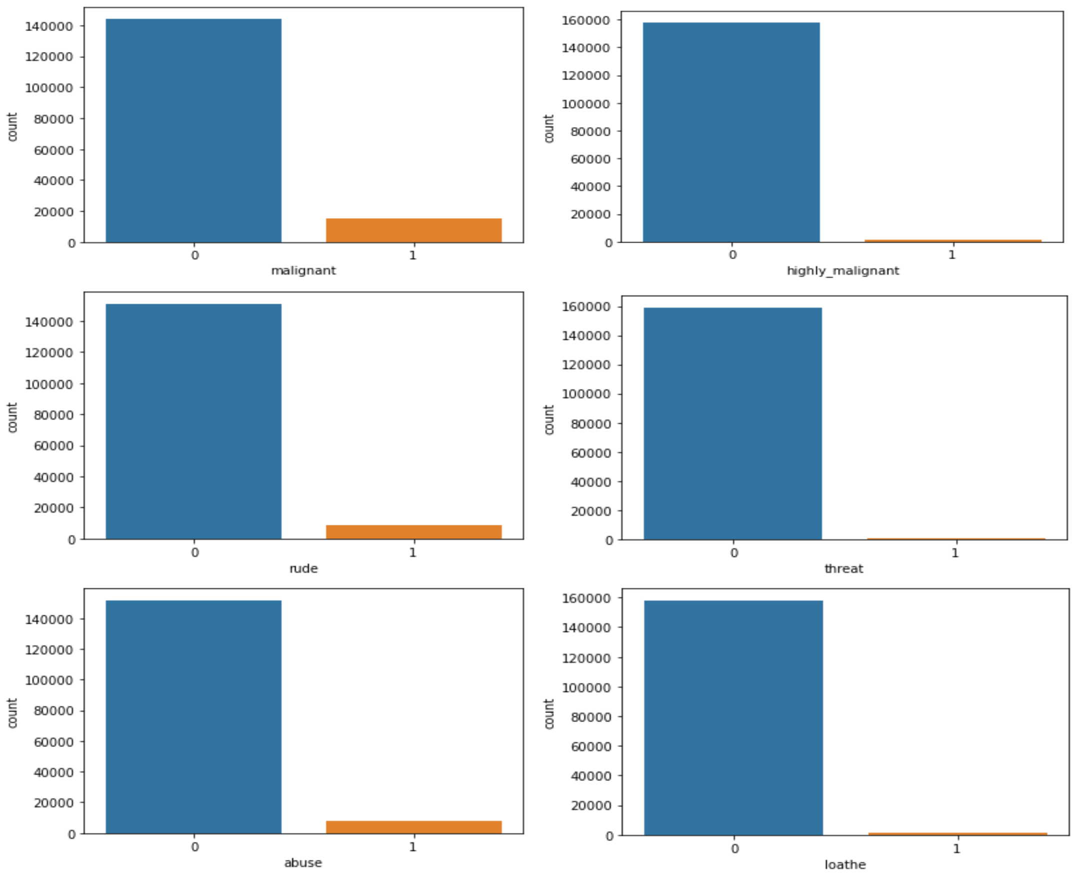
10. Similar process done on test data and preparing train and test data
11. Building the model

EDA and Visualization

average length of comment: 394.139



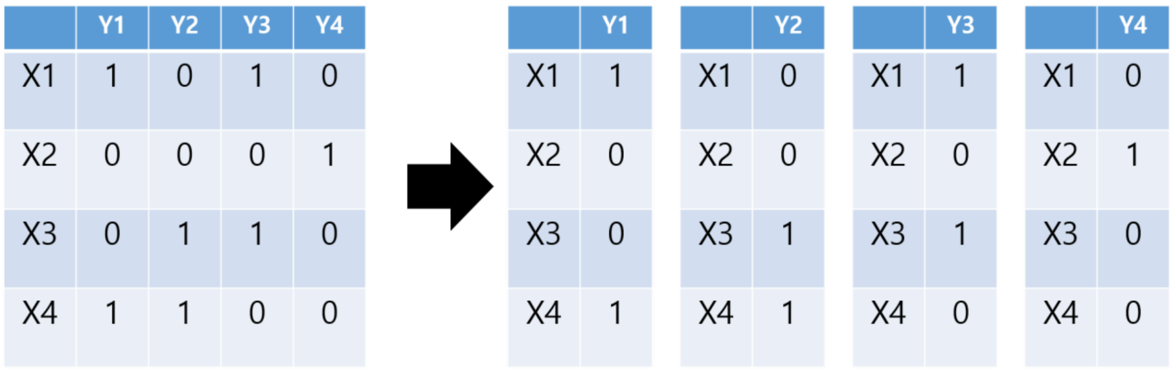
From the first visualization we can observe that comments have varying lengths from within 200 up to 1200. The majority of comments have length up to 200, and as we move towards greater lengths, the number of comments keep on falling. Since including very long length comments for training will increase the number of words manifold, it is important to set a threshold value for optimum results.



From the above visualizations, we can observe the number of words falling under the six different outcome labels malignant, highly_malignant, rude, threat, abuse, loathe.

Algorithm & Techniques :

Binary Relevance Method: This method does not take into account the interdependence of labels. Each label is solved separately like a single label classification problem. This is the simplest approach to be applied.



	Y1	Y2	Y3	Y4
X1	1	0	1	0
X2	0	0	0	1
X3	0	1	1	0
X4	1	1	0	0

	Y1
X1	1
X2	0
X3	0
X4	1

	Y2
X1	0
X2	0
X3	1
X4	1

	Y3
X1	1
X2	0
X3	1
X4	0

	Y4
X1	0
X2	1
X3	0
X4	0

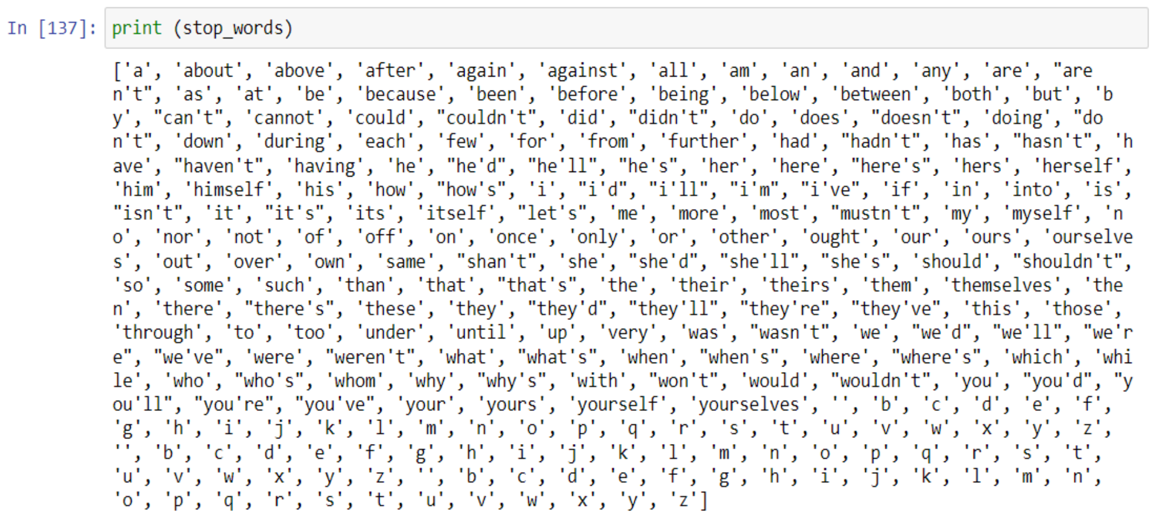
By plotting graphs as shown above we can observe that all the variables are skewed, the same can be removed in the further steps.

Data Preprocessing :

A string without all punctuations to be prepared:

1. The string library contains punctuation characters. This is imported and all numbers are appended to this string. Our comment_text field contains strings such as won't, didn't, etc. which contain apostrophe character('). To prevent these words from being converted to wont or didn't, the character ' represented as \' in escape sequence notation is replaced by empty character in the punctuation string.
2. Make_trans(intab, outtab) function is used. It returns a translation table that maps each character in the intab into the character at the same position in the outtab string.

Updating the list of stop words:



```
In [137]: print (stop_words)
```

```
['a', 'about', 'above', 'after', 'again', 'against', 'all', 'am', 'an', 'and', 'any', 'are', "are
n't", 'as', 'at', 'be', 'because', 'been', 'before', 'being', 'below', 'between', 'both', 'but', 'b
y', "can't", 'cannot', 'could', "couldn't", 'did', "didn't", 'do', 'does', "doesn't", 'doing', "do
n't", 'down', 'during', 'each', 'few', 'for', 'from', 'further', 'had', "hadn't", 'has', "hasn't", 'h
ave', "haven't", 'having', 'he', "he'd", "he'll", "he's", 'her', 'here', "here's", 'hers', 'herself',
'him', 'himself', 'his', 'how', "how's", 'i', "i'd", "i'll", "i'm", "i've", 'if', 'in', 'into', 'is',
"isn't", 'it', "it's", 'its', 'itself', "let's", 'me', 'more', 'most', "mustn't", 'my', 'myself', 'n
o', 'nor', 'not', 'of', 'off', 'on', 'once', 'only', 'or', 'other', 'ought', 'our', 'ours', 'ourselve
s', 'out', 'over', 'own', 'same', "shan't", 'she', "she'd", "she'll", "she's", 'should', "shouldn't",
'so', 'some', 'such', 'than', 'that', "that's", 'the', 'their', 'theirs', 'them', 'themselves', 'the
n', 'there', "there's", 'these', 'they', "they'd", "they'll", "they're", "they've", 'this', 'those',
'through', 'to', 'too', 'under', 'until', 'up', 'very', 'was', "wasn't", 'we', "we'd", "we'll", "we'r
e", "we've", 'were', "weren't", 'what', "what's", 'when', "when's", 'where', "where's", 'which', 'whi
le', 'who', "who's", 'whom', 'why', "why's", 'with', "won't", 'would', "wouldn't", 'you', "you'd", "y
ou'll", "you're", "you've", 'your', 'yours', 'yourself', 'yourselves', '', 'b', 'c', 'd', 'e', 'f',
'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
'', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
'u', 'v', 'w', 'x', 'y', 'z', '', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

1. Stop words are those words that are frequently used in both written and verbal communication and thereby do not have either a positive or negative impact on our statement like “is, this, us, etc.”.
2. Single letter words if existing or created due to any preprocessing step do not convey any useful meaning and so they can be directly removed. Hence letters from b to z, will be added to the list of stop words imported directly.

Stemming and Lemmatizing:

1. The process of converting inflected/derived words to their word stem or the root form is called stemming. Many similar origin words are converted to the same word e.g. words like "stems", "stemmer", "stemming", "stemmed" as based on "stem".
2. Lemmatizing is the process of grouping together the inflected forms of a word so they can be analyzed as a single item. This is quite similar to stemming in its working but differs since it depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document.
3. The wordnet library in nltk will be used for this purpose. Stemmer and Lemmatizer are also imported from nltk.

Applying Count Vectorizer:

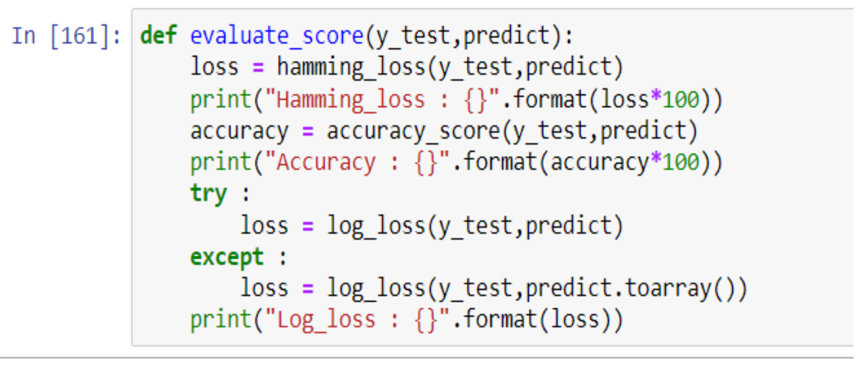
1. To convert a string of words into a matrix of words with column headers represented by words and their values signifying the frequency of occurrence of the word Count Vectorizer is used.
2. Stop words were accepted, convert to lowercase, and regular expression as its parameters. Here, we will be supplying our custom list of stop words created earlier and using lowercase option. Regular expression will have its default value.

Splitting dataset into Training and Testing:

Since we have our train and test data separated we can perform the above mentioned processes separately on train and test data and then form the train and test sets.

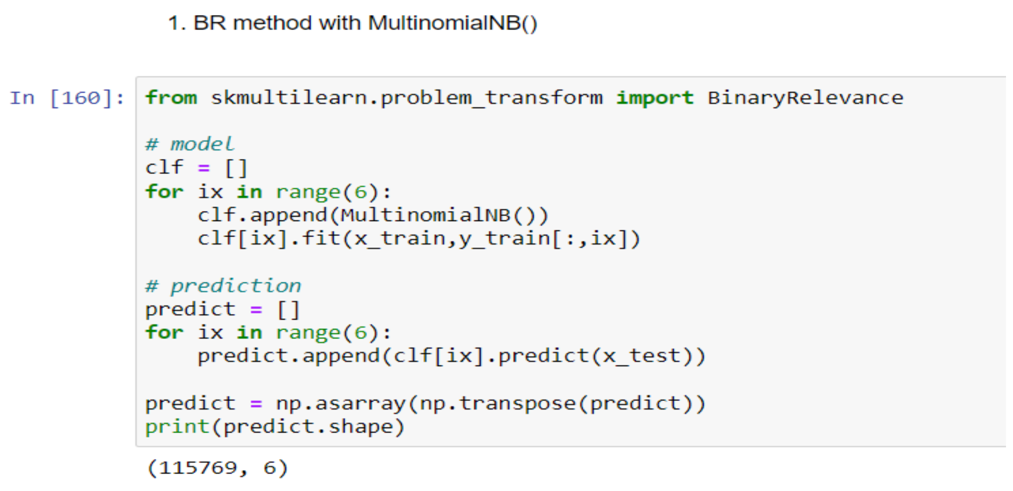
Implementation

We will be defining function evaluate_score for printing all evaluation metrics: Some implementations return a sparse matrix for the predictions and others return a dense matrix. So, a try except block were used to handle it.



```
In [161]: def evaluate_score(y_test,predict):
              loss = hamming_loss(y_test,predict)
              print("Hamming_loss : {}".format(loss*100))
              accuracy = accuracy_score(y_test,predict)
              print("Accuracy : {}".format(accuracy*100))
              try :
                  loss = log_loss(y_test,predict)
              except :
                  loss = log_loss(y_test,predict.toarray())
              print("Log_loss : {}".format(loss))
```

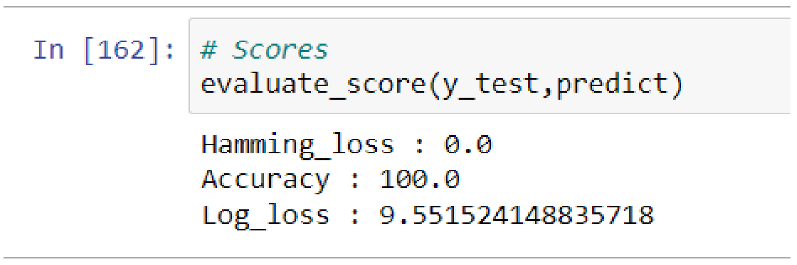
Binary Relevance method were implemented from scratch. It does not consider the interdependence of labels and basically creates a separate classifier for each of the labels. The code is as follows:



1. BR method with MultinomialNB()

```
In [160]: from skmultilearn.problem_transform import BinaryRelevance

          # model
          clf = []
          for ix in range(6):
              clf.append(MultinomialNB())
              clf[ix].fit(x_train,y_train[:,ix])

          # prediction
          predict = []
          for ix in range(6):
              predict.append(clf[ix].predict(x_test))

          predict = np.asarray(np.transpose(predict))
          print(predict.shape)

          (115769, 6)
```

Our model was the Binary Relevance method using MultinomialNB classifier. It led to the following results:



```
In [162]: # Scores
          evaluate_score(y_test,predict)

          Hamming_loss : 0.0
          Accuracy : 100.0
          Log_loss : 9.551524148835718
```

CONCLUSION

To summarize this project, the first step involved studying the train and test data.

The second major step was performing cleaning of data including punctuation removal, stop word removal, stemming and lemmatizing: This step was also crucial since the occurrence of similar origin words but having different spellings will intend to give similar classification, but computer cannot recognize this on its own. Hence, this step helped to a large extent in both removing and modifying existing words.

The third step was to build model Binary Relevance method with MultinomialNB()

Finally checking for the evaluation metrics: The two major evaluation metrics I planned to check on were hamming-loss and log-loss. Hence the performance of the model was verified on the basis of the combination of both these losses.

Although we have we have built a model which is giving us accuracy of 100%, it is always good to build 4 to 5 models for the same test and train data.