Assignment - 2 Report

Naive Bayes Classifier

Group-6

Aaditya Agrawal, 19CS10003
Debanjan Saha, 19CS30014

Dataset

Spooky Author Identification:
https://www.kaggle.com/c/spooky-author-identification/overview

About the Dataset

The dataset was obtained from Kaggle and is prepared from the excerpts from horror stories by Edgar Allan Poe**(EAP)**, Mary Shelley(MWS), and HP Lovecraft (HPL). The objective of the dataset is to predict the author from a given excerpt or specifically a set of words.

Analysing the Data

- The given dataset has 19579 rows each consisting of 3 columns which give us the information about a unique id of the row, text and the author who wrote that text.

	id	text	author
0	id26305	This process, however, afforded me no means of...	EAP
1	id17569	It never once occurred to me that the fumbling...	HPL
2	id11008	In his left hand was a gold snuff box, from wh...	EAP
3	id27763	How lovely is spring As we looked from Windsor...	MWS
4	id12958	Finding nothing else, not even gold, the Super...	HPL
...	...	...	...
19574	id17718	I could have fancied, while I looked at it, th...	EAP
19575	id08973	The lids clenched themselves together as if in...	EAP
19576	id05267	Mais il faut agir that is to say, a Frenchman ...	EAP
19577	id17513	For an item of news like this, it strikes us i...	EAP
19578	id00393	He laid a gnarled claw on my shoulder, and it ...	HPL

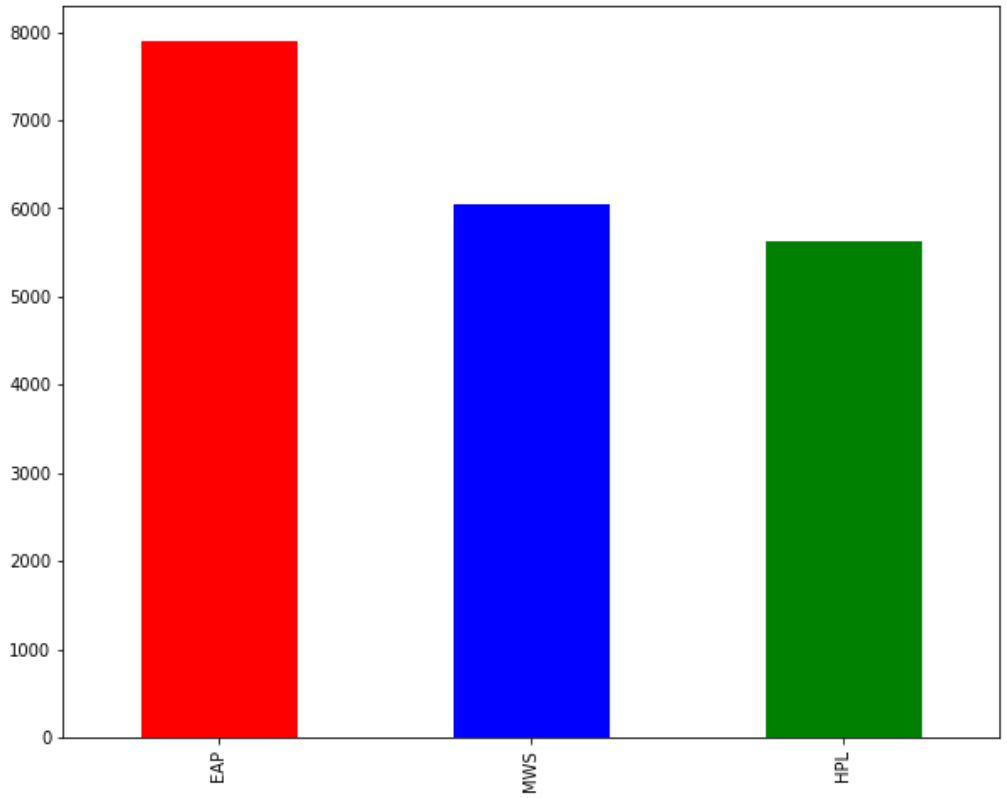
19579 rows × 3 columns

- From further analysis of the dataset, we observed and confirmed that there are 3 different authors and the dataset does not contain any missing values.

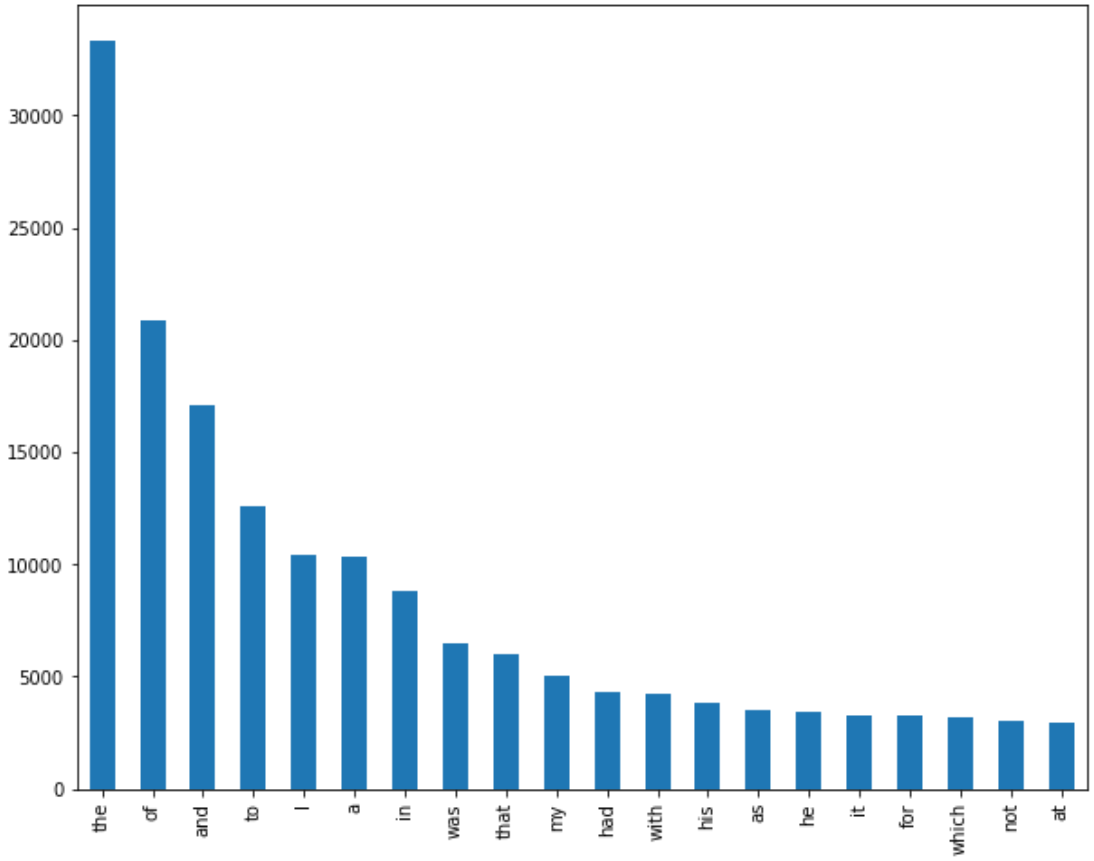
	id	text	author
count	19579	19579	19579
unique	19579	19579	3
top	id26305	This process, however, afforded me no means of...	EAP
freq	1	1	7900

- To check the distribution of the different authors, we plotted a bar graph to analyse how many texts present in our dataset have been written by each author. The plot is as follows:

```
EAP    7900
MWS    6044
HPL    5635
Name: author, dtype: int64
```



- We also analyzed the most frequent words present in all the texts and tried to check the top 20 out of the most frequent words:



- As we can see from the above histogram, the most frequent do not provide much information about the authors and are mostly **articles, prepositions and pronouns**.

Implementation:

Tokenization:

- From each example in the dataset, we extracted the text and split it into words using **regex** library function after converting it to lowercase characters and stored them as a list of words

- We maintained a **set data structure** and stored all the unique words obtained from each of the raw text present in the dataset.

- The size of this set was found to be **25078**

Text Cleaning:

- As instructed in our problem statement, we went through the stopwords listed on the page: https://gist.github.com/sebleier/554280 and prepared a list of strings from those stopwords

- We removed any occurrence of these stopwords from our set of unique words and the final size of the set after removing the stopwords was found to be **24951**.

FeatureMatrix Generation:

- **wordIndex (Helper Mapping) :**
    - We sorted the set of relevant words after removing the stopwords in alphabetical order and created a **mapping** called **wordIndex** for each of the words to an index which denotes in which column it will be stored in the featureMatrix. This map was stored in a JSON file called *"wordIndex.json"* for future reference.

- We went through each row of the dataset and for each relevant word from the raw text in our dataset, we set the corresponding column of that row in the featureMatrix as 0/1 using the mapping **word index**.

- The dimension of the featureMatrix was 19579x24951. Hence we found it convenient to store it in **scipy.sparse.csr_matrix** format

Naive Bayes Classifier Model:

We have implemented our Naive Bayes Classifier in a modular structure so that we can instantiate the classifier and do our prediction tasks using its methods.

➔ **Methods :**

- **init :** This is the constructor of our Naive Bayes Classifier. It takes a bunch of parameters such as a list of test indices and a list of train indices randomly generated from the list of row indices of the dataset to train and test our classifier. It also takes the featureMatrix and the data frame obtained from the dataset and saves it as its instance's attributes. A mapping of the possible outcomes/class-labels is also stored for computational purposes.

- **train:** This method trains the classifier on the training examples and finds out the class distributions, and the different probabilities(Likelihood

and Prior) which are used to calculate the posterior probability for a test example.

- **apply_laplace_correction:** This method works in the same way as the '*train*' method. However, during the calculation of likelihood probabilities, It applies Laplace correction to avoid zero probabilities. If probability is x/N, then after applying Laplace correction it becomes (x + alpha)/(N + k*alpha) for a test example where k is the dimension of multinomial distribution and alpha is a correction parameter to this method. We pass alpha = 1 for training our examples.

- **evaluate_example:** This method takes a feature_vector (X_1,X_2, ..., X_n) corresponding to a test example as its parameter [where n = number of unique vocabulary words / columns in our featureMatrix] and returns the author which the classifier is most confident about from the calculated probabilities.

- **test:** This method tests all the test examples from our dataset and stores the accuracy of the predictions on the test set and a list of the tuples (actual_author, predicted_author) as attributes in the instantiated object of our Classifier.

Utility functions

- **train_test_split_indices:** This function takes two parameters - number of examples in the dataset and a percentage of split for splitting the dataset into training set and test set. It generates a list of indices based on number of all examples and randomly splits the list into training and test set with the given percentage of split and returns the training indices and the test indices on which we will perform our training and testing

- **generate_statistics:** This method is a utility function to generate the precision, f-score, sensitivity, specificity obtained from the predictions of our Naive Bayes Classifier. It takes the list of the tuple of predictions -(actual_author,predicted_author), a list of possible outcomes/class-labels and a mapping of the class labels for computational purposes and generates the true positive, true negatives, false positives and false negatives for each class and generates the micro and macro precision, f-score, sensitivity, specificity.

- **get_confidence_interval:** This method takes two parameters - a score of the metric we are trying to find out the confidence interval of and the number of samples over which we have calculated our score and generates the 95% confidence interval of it.

Results

All the results below were obtained at the time when we ran the program. Due to the random splitting, the results you may obtain can be slightly different.

Without Laplace Correction:

- Accuracy of the classifier = **0.5881852230166837 ≃ 58.82%**

- 95% Confidence Interval = **[0.5755989508680747, 0.6007714951652928]**

- Precision, Sensitivity, Specificity, F-Score Stats:

```
MICRO STATS
Micro Precision = 0.5881852230166837
Micro Sensitivity(Recall) = 0.5881852230166837
Micro Specificity = 0.7407010397684639
Micro F-Score = 0.5881852230166837

****************

MACRO STATS
Macro Precision = 0.707326296762176
Macro Sensitivity(Recall) = 0.5446870666401318
Macro Specificity = 0.7581278620492882
Macro F-Score = 0.5472638202615497
```

After Laplace Correction:

- Accuracy of the classifier = **0.7892407218249915 ≃ 78.92%**

- 95% Confidence Interval = **[0.7788106525233968, 0.7996707911265862]**

- Precision, Sensitivity, Specificity, F-Score Stats:

```
MICRO STATS
Micro Precision = 0.7892407218249915
Micro Sensitivity(Recall) = 0.7892407218249915
Micro Specificity = 0.8822074215033302
Micro F-Score = 0.7892407218249915

****************

MACRO STATS
Macro Precision = 0.83412023119829
Macro Sensitivity(Recall) = 0.7705625414597101
Macro Specificity = 0.8788137016138269
Macro F-Score = 0.7864081731595802
```

- We observe that there is a substantial increase in accuracy on applying Laplacian correction. This is due to the fact that earlier, if one of the features had zero likelihood probability, it undermined the contribution of other features.

- The intuition for F-measure is that both precision and recall(sensitivity) measures are balanced in importance and that only a good precision and good recall together result in a good F-measure. After applying the Laplacian correction, we can see a massive improvement in F-Score proving us how robust it is in terms of confidently making predictions