REPORT

GUTENBERG TEXT CLASSIFICATION

GROUP NUMBER: 10

GROUP MEMBERS: DEEPANRAJ ARUMUGAM MURUGESAN
KEERTHIVASAN SURYANARAYANAN
RANJITKUMAR SETHURAMAN
SRUTI SRIRAM

README FILE:
https://github.com/Ranjitsethu/gutenberg/blob/main/README.md

PREPARING THE DATA:

We have taken five different samples from the Gutenberg digital books corpus, each sample representing a different author and belonging to the same genre and semantic category. We have **randomly selected 200 partitions from each book** to serve as representative samples of the source material. Also, we have prepared **100 words records for each document** and labeled them as "a," "b," "c," etc. according to the book to which they belong.

These are the books that we selected:

BOOK ID	BOOK	AUTHOR
1533	Macbeth	William Shakespeare
1754	The Sea-Gull	Anton Pavlovich Chekhov
1342	Pride and Prejudice	Jane Austen
345	Dracula	Bram Stoker
2446	An Enemy of the People	Henrik Ibsen

PREPROCESSING AND DATA CLEANING:

The books are pre-selected by their book IDs and stored in the list "bi_list". The 200 random partitions of 100 words each are then selected from the processed text

and stored in a data frame along with the label for each book. Finally, all the data frames are combined into one and saved as a CSV file.

The following python snippet uses the NTLK library to scrape text data from books on the Gutenberg project website.

```
def partition(bi, la):
    url = f"http://www.gutenberg.org/files/{bi}/{bi}-0.txt" # Downloading the book from gutenberg website
    response = urllib.request.urlopen(url)
    raw_text = response.read().decode('utf-8')
    #cleaned_text = re.sub(r'\b\w*\d+\w*\b|\w*_\w*|(?<=\w)\'(?=\w)', '', raw_text)
    cleaned_text = re.sub(r'\r\n?', '\n', raw_text).strip()# Cleaning the text
    wo = re.findall(r'\w+', cleaned_text)# Splitting to list of words
    random.seed(42)
    starting_indices = random.sample(range(0, len(wo) - 100), 200)# Selecting 200 random starting indices for partitions
    partitions = []#creating empty list for partitions
    for i in starting_indices: #iterating through the starting indices
        partition = wo[i:i+100]
        partitions.append(partition)#appending each partition to the partitions list
    df = pd.DataFrame({'label': la, 'partition': partitions})# Creating a DF to store all the partitions
    return df #the dataframe is returned
bi_list = [1533,1754,1342,345,2446]# List of book ids on the website
la_list = ['a', 'b', 'c','d','e']
df_list = [] #empty lsit
for bi, la in zip(bi_list, la_list):
    df_list.append(partition(bi, la)) #function is being called inside and the values are appended
df = pd.concat(df_list)# Concatenating the DataFrames and exporting to CSV format
print(df)
df.to_csv('finaloutput.csv', index=False)
```

1. Cleaning the raw text: The newline characters "\r\n" in the text are replaced with "\n" and the text is stripped of any whitespaces at the beginning and end.
2. Splitting the cleaned text into words: The cleaned text is then split into a list of words using the re.findall() method and a regular expression pattern "\w+".

Further pre-processing of the partitions is done using the following statements.

```
df['clean'] = df['partition'].apply(lambda x: ' '.join(map(str, x)))
df["clean"]=df["clean"].apply(lambda x: " ".join([w for w in x.split() if len(w)>2]))
df["clean"] = df["clean"].apply(lambda x: " ".join(re.sub('[^0-9a-zA-Z]+', '', w) for w in x.split()))
```

The code is preprocessing the text in the "partition" column of the dataframe df. It's doing the following three steps:

1. **df['clean'] = df['partition'].apply(lambda x: ' '.join(map(str, x)))**: Here, a new column "clean" is created in the dataframe **df** and the values in the column "partition" are concatenated into a single string separated by spaces.

The **map** function is used to convert each element in the list to a string and the **join** function is used to concatenate the strings.

2. **df["clean"]=df["clean"].apply(lambda x: " ".join([w for w in x.split() if len(w)>2]))**: In this step, only the words with length greater than 2 are selected from the "clean" column and concatenated with a space in between to form a single string.

3. **df["clean"] = df["clean"].apply(lambda x: " ".join(re.sub('[^0-9a-zA-Z]+', '', w) for w in x.split()))**: This step is using regular expressions to remove all characters that are not alphanumeric (letters or numbers). The **re.sub** function is used to replace all characters that are not in the set **[0-9a-zA-Z]** (i.e., not a number or a letter) with an empty string. The final result is a string that contains only alphanumeric characters.

FEATURE ENGINEERING:

Feature engineering refers to tokenization, stemming/lemmatization, and converting the text into numerical representation (e.g. through a bag of words or a term frequency-inverse document frequency approach). These steps convert the raw text into a numerical representation that can be used as input to machine learning models.

STEMMING:

```
from nltk.stem.porter import PorterStemmer
stemmer= PorterStemmer()
tokenizedtweet1 = tokenizedtweet.apply(lambda x : [stemmer.stem(word) for word in x])
tokenizedtweet1.head()
```

We have used the PorterStemmer algorithm from the nltk library to stem the words. The tokenized words are first converted into a list of words and then the stemmer function is applied to each word in the list to produce the stemmed version of each word.

Finally, the list of stemmed words are converted into strings and stored in “clean” column of the dataframe.

LEMMATIZATION:

```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
```

```
lemmatizer = WordNetLemmatizer()
tokenizedtweet2=tokenizedtweet.apply(lambda x : [lemmatizer.lemmatize(word) for word in x])
tokenizedtweet2.head()
```

```
df["clean_lem"]=tokenizedtweet2
df.head()
```

```
df['clean_lem'] = df['clean_lem'].apply(lambda x: x.lower())
```

Firstly, a WordNetLemmatizer() object is created. Using, the lemmatize() function is applied to each tokenized word to get its root form. Then, the lemmatized text is stored in the dataframe as a new column.

LABEL ENCODING:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df["label"] = le.fit_transform(df["label"])
```

The categorical labels(a,b,c,d,e) are converted into numerical representations(0,1,2,3,4) for training the Machine Learning models.

BAG OF WORDS:

```
from sklearn.feature_extraction.text import CountVectorizer
bow_vectorizer=CountVectorizer(max_df=0.90,min_df=2,max_features=1000,stop_words="english")
```

```
bow=bow_vectorizer.fit_transform(df["clean"])
```

```
bow_l=bow_vectorizer.fit_transform(df["clean_lem"]) #run this or above
```

Initialize a CountVectorizer object called bow_vectorizer. The parameters passed to it are defined:

max_df: The maximum number of documents that a word can appear in. If a word appears in more than max_df documents, it is ignored. The default value is 1.0.

min_df: The minimum number of documents that a word must appear in to be considered. If a word appears in fewer than min_df documents, it is ignored.

max_features: The maximum number of features to consider.

stop_words: The words that should be ignored. In this case, "english" is passed to ignore all the English stop words.

The term frequency matrix is computed using the **bow_vectorizer.fit_transform()** function. The same procedure is done for the lemmatized sentences.

TERM FREQUENCY – INVERSE DOCUMENT FREQUENCY:

```
from sklearn.feature_extraction.text import TfidfVectorizer
t=TfidfVectorizer(max_df=0.90,min_df=2,max_features=1000,stop_words="english",use_idf=True,ngram_range=(1,2))

tf=t.fit_transform(df["clean"])

tf_l=t.fit_transform(df["clean_lem"])
```

Using the TfidfVectorizer is used to compute the TF-IDF values. First it computes the text data for the column 'clean' which contains the stemmed words. The same procedure is repeated for the column clean_lem which contains the lemmatized words.

The parameters passed to the TfidfVectorizer are as follows:

1. **max_df**: It represents the maximum document frequency, which is the maximum percentage of documents a term can appear in to be included in the matrix. A value of 0.90 means that a term can appear in 90% of the documents at most.
2. **min_df**: It represents the minimum document frequency, which is the minimum number of documents a term must appear in to be included in the matrix. A value of 2 means that a term must appear in at least two documents.
3. **max_features**: It represents the maximum number of features that should be extracted from the corpus of documents. In this case, the value is set to 1000, which means that only 1000 features will be used in the matrix.

4. **stop_words**: It represents the words that should be ignored. Here, "english" is passed as a value, which indicates that the standard English stop words will be ignored.
5. **use_idf:** It is a flag indicating whether the inverse document frequency (IDF) should be used. The value is set to True, which means that the TfidfVectorizer will use the IDF weighting.
6. **ngram_range:** It represents the range of n-grams to be extracted. The value (1,2) indicates that both unigrams (single words) and bigrams (pairs of words) will be extracted.

IMPLEMENTATION OF ML MODELS – LOGISTIC REGRESSION, DECISION TREE, RANDOM FOREST, SVM, XGBOOST, NAÏVE BAYES

To evaluate the performance of the various models using different combinations of pre-processing techniques and feature representation methods. The pre-processing techniques used are stemming and lemmatization and the feature representation methods used are Bag of Words and TF-IDF.

The function modeltype is used to fit the ML models on the training data, predict the target variable for the test data and compute the accuracy of the model. This function uses **K-Fold cross-validation to divide the data into train and test sets**, fit the model on the training set, and evaluate its performance on the test set. The mean accuracy of the model is computed over the 10 folds.

The mean accuracy of the models for **4 different feature representation methods**, each of them being a combination of either **stemming or lemmatization** and either **Bag of Words or TF-IDF**.

The python code for each of the models for the different techniques and feature representation methods are given below.

LOGISTIC REGRESSION

The mean accuracy, confusion matrices and classification reports are obtained by calling the modeltypefinal function. We are printing only the matrix and report for Stemming that uses bag of words.

Also, the accuracies obtained for the other feature representations are presented in the table.

```
cv_accuracy,c,reports = modeltypefinal(r,tf,labels)#STEMMING, TF-IDF
stemmed_accuracies_tf.append(cv_accuracy*100)
print("Mean Accuracy:", cv_accuracy)
print("CONFUSION MATRIX")
print(c[0]) #printing only the first fold by canging the number it can be shown for every fold
print("CLASSIFICATION REPORT")
print(reports[0]) #printing only the first fold by canging the number it can be shown for every fold
```

```
Mean Accuracy: 0.8470000000000001
CONFUSION MATRIX
[[10  1  1  0  1]
 [ 2 23  0  2  1]
 [ 0  1 19  1  0]
 [ 1  2  0 18  0]
 [ 1  2  0  0 14]]
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.71      0.77      0.74        13
           1       0.79      0.82      0.81        28
           2       0.95      0.90      0.93        21
           3       0.86      0.86      0.86        21
           4       0.88      0.82      0.85        17

    accuracy                           0.84       100
   macro avg       0.84      0.84      0.84       100
weighted avg       0.84      0.84      0.84       100
```

Pre-processing Technique	BOW	TF-IDF
STEMMING	0.882	0.914
LEMMATIZATION	0.882	0.909

DECISION TREE

```
cv_accuracy,c,reports = modeltypefinal(classifier,tf,labels)#STEMMING, TF-IDF
stemmed_accuracies_tf.append(cv_accuracy*100)
print("Mean Accuracy:", cv_accuracy)
print("CONFUSION MATRIX")
print(c[0]) #printing only the first fold by canging the number it can be shown for every fold
print("CLASSIFICATION REPORT")
print(reports[0]) #printing only the first fold by canging the number it can be shown for every fold
```

```
Mean Accuracy: 0.8230000000000001
CONFUSION MATRIX
[[ 9  3  1  0  0]
 [ 1 24  1  1  1]
 [ 0  1 16  4  0]
 [ 0  2  3 16  0]
 [ 0  1  0  0 16]]
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.90      0.69      0.78        13
           1       0.77      0.86      0.81        28
           2       0.76      0.76      0.76        21
           3       0.76      0.76      0.76        21
           4       0.94      0.94      0.94        17

    accuracy                           0.81       100
   macro avg       0.83      0.80      0.81       100
weighted avg       0.81      0.81      0.81       100
```

Pre-processing Technique	BOW	TF-IDF
STEMMING	0.768	0.823
LEMMATIZATION	0.762	0.787

RANDOM FOREST

```
cv_accuracy,c,reports = modeltypefinal(r,tf,labels)#STEMMING, TF-IDF
stemmed_accuracies_tf.append(cv_accuracy*100)
print("Mean Accuracy:", cv_accuracy)
print("CONFUSION MATRIX")
print(c[0]) #printing only the first fold by canging the number it can be shown for every fold
print("CLASSIFICATION REPORT")
print(reports[0]) #printing only the first fold by canging the number it can be shown for every fold
```

```
Mean Accuracy: 0.8470000000000001
CONFUSION MATRIX
[[10  1  1  0  1]
 [ 2 23  0  2  1]
 [ 0  1 19  1  0]
 [ 1  2  0 18  0]
 [ 1  2  0  0 14]]
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.71      0.77      0.74        13
           1       0.79      0.82      0.81        28
           2       0.95      0.90      0.93        21
           3       0.86      0.86      0.86        21
           4       0.88      0.82      0.85        17

    accuracy                           0.84       100
   macro avg       0.84      0.84      0.84       100
weighted avg       0.84      0.84      0.84       100
```

Pre-processsing Technique	BOW	TF-IDF
STEMMING	0.745	0.847
LEMMATIZATION	0.739	0.837

SVM

```
cv_accuracy,c,reports = modeltypefinal(s,tf,labels)#STEMMING, TF-IDF
stemmed_accuracies_tf.append(cv_accuracy*100)
print("Mean Accuracy:", cv_accuracy)
print("CONFUSION MATRIX")
print(c[0]) #printing only the first fold by changing the number it can be shown for every fold
print("CLASSIFICATION REPORT")
print(reports[0]) #printing only the first fold by changing the number it can be shown for every fold
```

```
Mean Accuracy: 0.9119999999999999
CONFUSION MATRIX
[[11  2  0  0  0]
 [ 3 25  0  0  0]
 [ 0  0 21  0  0]
 [ 0  0  1 20  0]
 [ 1  1  0  0 15]]
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.73      0.85      0.79        13
           1       0.89      0.89      0.89        28
           2       0.95      1.00      0.98        21
           3       1.00      0.95      0.98        21
           4       1.00      0.88      0.94        17

    accuracy                           0.92       100
   macro avg       0.92      0.91      0.91       100
weighted avg       0.93      0.92      0.92       100
```

Pre-processing Technique	BOW	TF-IDF
STEMMING	0.874	0.911
LEMMATIZATION	0.875	0.907

XGBOOST

```
cv_accuracy,c,reports = modeltypefinal(model,tf.todense(),labels)#STEMMING, TF-IDF
stemmed_accuracies_tf.append(cv_accuracy*100)
print("Mean Accuracy:", cv_accuracy)
print("CONFUSION MATRIX")
print(c[0]) #printing only the first fold by canging the number it can be shown for every fold
print("CLASSIFICATION REPORT")
print(reports[0]) #printing only the first fold by canging the number it can be shown for every fold
```

```
Mean Accuracy: 0.873
CONFUSION MATRIX
[[ 9  2  1  0  1]
 [ 1 24  1  0  2]
 [ 0  0 19  2  0]
 [ 0  0  0 21  0]
 [ 0  2  0  0 15]]
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.90      0.69      0.78        13
           1       0.86      0.86      0.86        28
           2       0.90      0.90      0.90        21
           3       0.91      1.00      0.95        21
           4       0.83      0.88      0.86        17

    accuracy                           0.88       100
   macro avg       0.88      0.87      0.87       100
weighted avg       0.88      0.88      0.88       100
```

Pre-processsing Technique	BOW	TF-IDF
STEMMING	0.843	0.873
LEMMATIZATION	0.844	0.869

NAÏVE BAYES

```
cv_accuracy,c,reports = modeltypefinal(gnb,tf.todense(),labels)#STEMMING, TF-IDF
stemmed_accuracies_tf.append(cv_accuracy*100)
print("Mean Accuracy:", cv_accuracy)
print("CONFUSION MATRIX")
print(c[0]) #printing only the first fold by canging the number it can be shown for every fold
print("CLASSIFICATION REPORT")
print(reports[0]) #printing only the first fold by canging the number it can be shown for every fold
```

```
Mean Accuracy: 0.889
CONFUSION MATRIX
[[10  2  0  0  1]
 [ 1 25  0  0  2]
 [ 0  0 21  0  0]
 [ 0  0  1 20  0]
 [ 1  1  0  0 15]]
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.83      0.77      0.80        13
           1       0.89      0.89      0.89        28
           2       0.95      1.00      0.98        21
           3       1.00      0.95      0.98        21
           4       0.83      0.88      0.86        17

    accuracy                           0.91       100
   macro avg       0.90      0.90      0.90       100
weighted avg       0.91      0.91      0.91       100
```

Pre-processing Technique	BOW	TF-IDF

STEMMING	0.887	0.889
LEMMATIZATION	0.882	0.876

EVALUATION – CROSS VALIDATION

```
from sklearn.model_selection import KFold
def modeltype(model,b,labels, n_splits=10):
    kfold = KFold(n_splits=n_splits, shuffle=True, random_state=0)
    accuracy = []
    for train_index, test_index in kfold.split(b):
        X_train, X_test = b[train_index], b[test_index]
        y_train, y_test = labels[train_index], labels[test_index]
        model.fit(X_train, y_train)
        accuracy.append(model.score(X_test, y_test))
    return np.mean(accuracy)
labels = df["label"].to_numpy()
```

The modeltype function takes a model, a feature matrix b, and the corresponding target variable labels as inputs.

The function performs 10-fold cross-validation to evaluate the model's accuracy. In each iteration, the function splits the feature matrix b and the target variable labels into a training set and a testing set using the train_index and test_index obtained from kfold.split(b).

The model is then fit on the training set and the accuracy is computed on the testing set using the model.score method. Finally, the mean accuracy over all the 10 folds is returned as the result.

```
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
def modeltypefinal(model,b,labels, n_splits=10):
    kfold = KFold(n_splits=n_splits, shuffle=True, random_state=0)
    accuracy = []
    c=[]
    reports=[]
    for train_index, test_index in kfold.split(b):
        X_train, X_test = b[train_index], b[test_index]
        y_train, y_test = labels[train_index], labels[test_index]
        model.fit(X_train, y_train)
        y_pred=model.predict(X_test)
        cm=confusion_matrix(y_test,y_pred)
        reports.append(classification_report(y_test, y_pred))
        c.append(cm)
        accuracy.append(model.score(X_test, y_test))
    return np.mean(accuracy),c,reports
labels = df["label"].to_numpy()
```

The function modeltypefinal() trains and evaluates a machine learning model using k-fold cross-validation. It accepts a model object, the feature matrix b, and the target variable labels as input.

The function splits the data into n_splits (default 10) parts and uses each part as the test set once while training on the rest of the data.

It fits the model to the training data and makes predictions on the test data, computes the accuracy score, confusion matrix, and the classification report for each iteration.

The mean accuracy score, a list of confusion matrices, and a list of classification reports are returned at the end of the n_splits iterations.

DIFFERENCE BETWEEN modeltype() and modeltypefinal()

The difference between the modeltype and modeltypefinal functions is that the modeltype function only computes the accuracy of the model, while the modeltypefinal function computes not only the accuracy of the model, but also the confusion matrix and the classification report of the model.

THE CHAMPION MODEL

The Champion model is LOGISTIC REGRESSION which used stemmed data with TF-IDF.

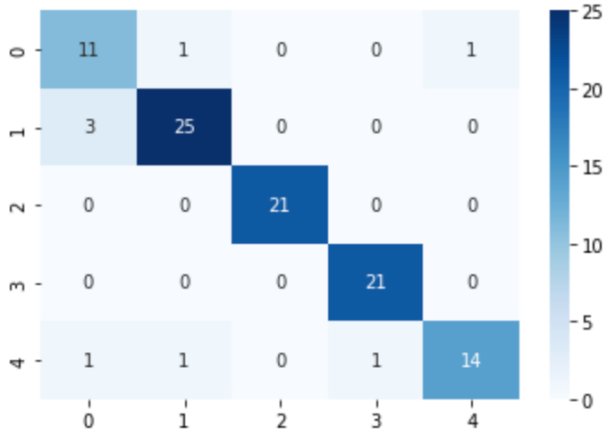
```
cv_accuracy = modeltype(le,tf,labels) #OUR BEST MODEL - TF-IDF WITH LOGISTIC REGRESSION AND STEMMING
print("Mean Accuracy:", cv_accuracy)
```

```
Mean Accuracy: 0.912
```

```
cv_accuracy,c,reports = modeltypefinal(le,tf,labels)#STEMMING, TF-IDF with LOGISTIC REGRESSION IS OUR BEST MODEL
print("Mean Accuracy:", cv_accuracy)
print("CONFUSION MATRIX")
print(c[0]) #printing only the first fold by canging the number it can be shown for every fold
print("CLASSIFICATION REPORT")
print(reports[0]) #printing only the first fold by canging the number it can be shown for every fold
```

CONFUSION MATRIX OF THE CHAMPION MODEL:

For illustration purposes, the confusion matrix of only the first fold is shown here:



HEAT MAP – A GRAPHICAL REPRESENTATION OF THE CONFUSION MATRIX

ERROR ANALYSIS

For the error analysis, we considered our best model - LOGISTIC REGRESSION with STEMMING and TF-IDF.

```
cv_accuracy,c,reports = modeltypefinal(le,tf,labels)#STEMMING, TF-IDF with LOGISTIC REGRESSION IS OUR BEST MODEL
print("Mean Accuracy:", cv_accuracy)
print("CONFUSION MATRIX")
print(c[0]) #printing only the first fold by canging the number it can be shown for every fold
print("CLASSIFICATION REPORT")
print(reports[0]) #printing only the first fold by canging the number it can be shown for every fold
```

```
Mean Accuracy: 0.9149999999999998
CONFUSION MATRIX
[[11  1  0  0  1]
 [ 3 25  0  0  0]
 [ 0  0 21  0  0]
 [ 0  0  0 21  0]
 [ 1  1  0  1 14]]
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.73      0.85      0.79        13
           1       0.93      0.89      0.91        28
           2       1.00      1.00      1.00        21
           3       0.95      1.00      0.98        21
           4       0.93      0.82      0.87        17

    accuracy                           0.92       100
   macro avg       0.91      0.91      0.91       100
weighted avg       0.92      0.92      0.92       100
```

From the classification report of our best model, we can see that the F1 score of the 0th book was lesser when compared to the rest.
To investigate further on this, we decided to take the reports of all ML models which had stemming and TF-IDF

DECISION TREE CLASSIFIER	CLASSIFICATION REPORT precision recall f1-score support 0 0.90 0.69 0.78 13 1 0.77 0.86 0.81 28 2 0.76 0.76 0.76 21 3 0.76 0.76 0.76 21 4 0.94 0.94 0.94 17 accuracy 0.81 100 macro avg 0.83 0.80 0.81 100 weighted avg 0.81 0.81 0.81 100
RANDOM FORREST CLASSIFIER	CLASSIFICATION REPORT precision recall f1-score support 0 0.71 0.77 0.74 13 1 0.79 0.82 0.81 28 2 0.95 0.90 0.93 21 3 0.86 0.86 0.86 21 4 0.88 0.82 0.85 17 accuracy 0.84 100 macro avg 0.84 0.84 0.84 100 weighted avg 0.84 0.84 0.84 100

SVM

CLASSIFICATION REPORT

	precision	recall	f1-score	support
0	0.73	0.85	0.79	13
1	0.89	0.89	0.89	28
2	0.95	1.00	0.98	21
3	1.00	0.95	0.98	21
4	1.00	0.88	0.94	17
accuracy			0.92	100
macro avg	0.92	0.91	0.91	100
weighted avg	0.93	0.92	0.92	100

NAÏVE BAYES

CLASSIFICATION REPORT

	precision	recall	f1-score	support
0	0.83	0.77	0.80	13
1	0.89	0.89	0.89	28
2	0.95	1.00	0.98	21
3	1.00	0.95	0.98	21
4	0.83	0.88	0.86	17
accuracy			0.91	100
macro avg	0.90	0.90	0.90	100
weighted avg	0.91	0.91	0.91	100

LOGISTIC REGRESSION

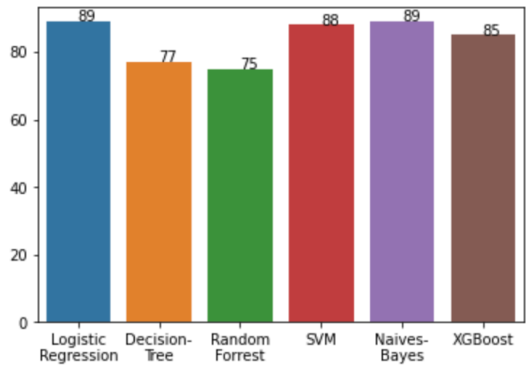
CLASSIFICATION REPORT

	precision	recall	f1-score	support
0	0.90	0.69	0.78	13
1	0.86	0.86	0.86	28
2	0.90	0.90	0.90	21
3	0.91	1.00	0.95	21
4	0.83	0.88	0.86	17
accuracy			0.88	100
macro avg	0.88	0.87	0.87	100
weighted avg	0.88	0.88	0.88	100

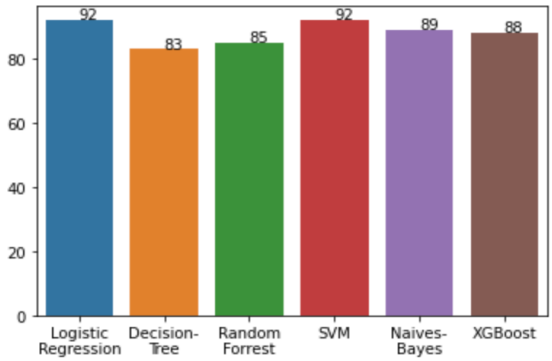
As seen above in all the models the 0^{th} Book has a low F1 score and our model was not able to make accurate predictions of this. As the precision was low in most of the cases we can say that the model was incorrectly predicting(classifying) sentences from other books as the 0^{th} book's sentences.

VISUALIZATIONS AND GRAPHING THE RESULTS

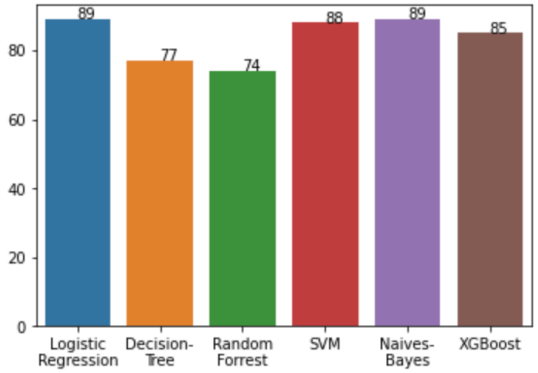
We have visualized the various classifier algorithms with the four feature representations. In all the four methods, Logistic Regression stood out and gave the highest accuracy.



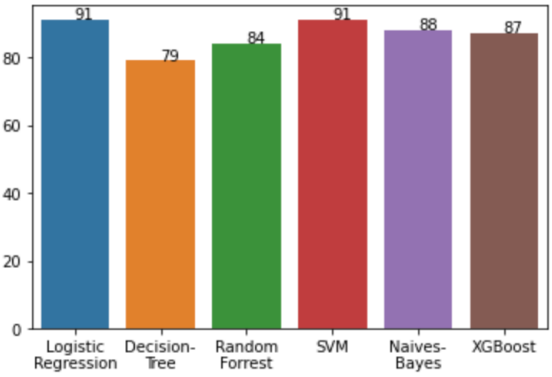
STEMMED DATA WITH BOW



STEMMED DATA WITH TF-IDF



LEMMATIZED DATA WITH BOW



LEMMATIZED DATA WITH TF-IDF

ANALYSIS OF BIAS AND VARIABILITY

```
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.73      0.85      0.79        13
           1       0.93      0.89      0.91        28
           2       1.00      1.00      1.00        21
           3       0.95      1.00      0.98        21
           4       0.93      0.82      0.87        17

    accuracy                           0.92       100
   macro avg       0.91      0.91      0.91       100
weighted avg       0.92      0.92      0.92       100
```

The Classification Report above is the report of our Champion Mode. Looking at the F1 score, Precision and Recall values of the model we can say that our model is performing well without overfitting. The values show that our model has a low bias and a low variance as it performed well in both our training and validation. Furthermore, to make sure our model doesn't overfit we have also done K-fold cross-validation with n_splits being 10.

```
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
def modeltypefinal(model,b,labels, n_splits=10):
    kfold = KFold(n_splits=n_splits, shuffle=True, random_state=0)
    accuracy = []
    c=[]
    reports=[]
    for train_index, test_index in kfold.split(b):
        X_train, X_test = b[train_index], b[test_index]
        y_train, y_test = labels[train_index], labels[test_index]
        model.fit(X_train, y_train)
        y_pred=model.predict(X_test)
        cm=confusion_matrix(y_test,y_pred)
        reports.append(classification_report(y_test, y_pred))
        c.append(cm)
        accuracy.append(model.score(X_test, y_test))
    return np.mean(accuracy),c,reports
labels = df["label"].to_numpy()
```

IDENTIFYING, MEASURING AND CONTROL THE MACHINE'S THRESHOLDS OF FACTORS OF PREDICTION HARDSHIP

Playing with the features and other factors provided us with leverages to make it harder for the model to predict and we were able to bring the accuracy down by about 20%.
The changes that we implemented were reducing the number of samples and records by 50%.

The number of words in each sentence was reduced from 100 to 50 and the number of sentences from each book was reduced to 100 from 200. So totally 500 records were used.

```
    starting_indices = random.sample(range(0, len(wo) - 100), 100)# Selecting 100 random starting indices for partitions
    partitions = []#creating empty list for partitions
    for i in starting_indices: #iterating through the starting indices
        partition = wo[i:i+50]
        partitions.append(partition)#appending each partition to the partitions list
```

Previously only words and numbers were considered, and the special characters were removed, now that line of code was commented out.

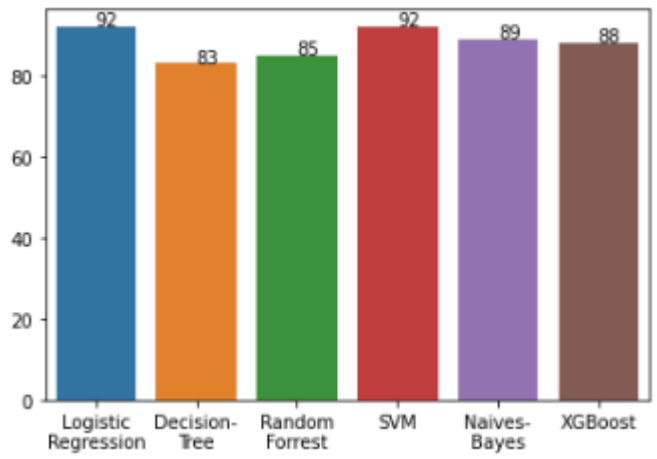
```
#df["clean"] = df["clean"].apply(lambda x: " ".join(re.sub('[^0-9a-zA-Z]+', '', w) for w in x.split()))
```

The nsplits value was reduced from 10 to 2

```
from sklearn.model_selection import KFold
def modeltype(model,b,labels, n_splits=2):
    kfold = KFold(n_splits=n_splits, shuffle=True, random_state=0)
```

All these changes impacted our accuracy in a huge way and the difference in accuracies are shown below:

BEFORE TWEAKING PARAMETERS



AFTER TWEAKING PARAMETERS