Report (Classification Assignment 1)

GROUP_1

DTI5125[EG] DATA SCIENCE APPLICATIONS [LEC] 20225

Team Members:

1-Khaled Mohammed Elsaka

2-Hadeer Mamdouh Abdelfattah

3-Nada Abdellatef Shaker

4-Nada Montasser Hassan

1.Objective

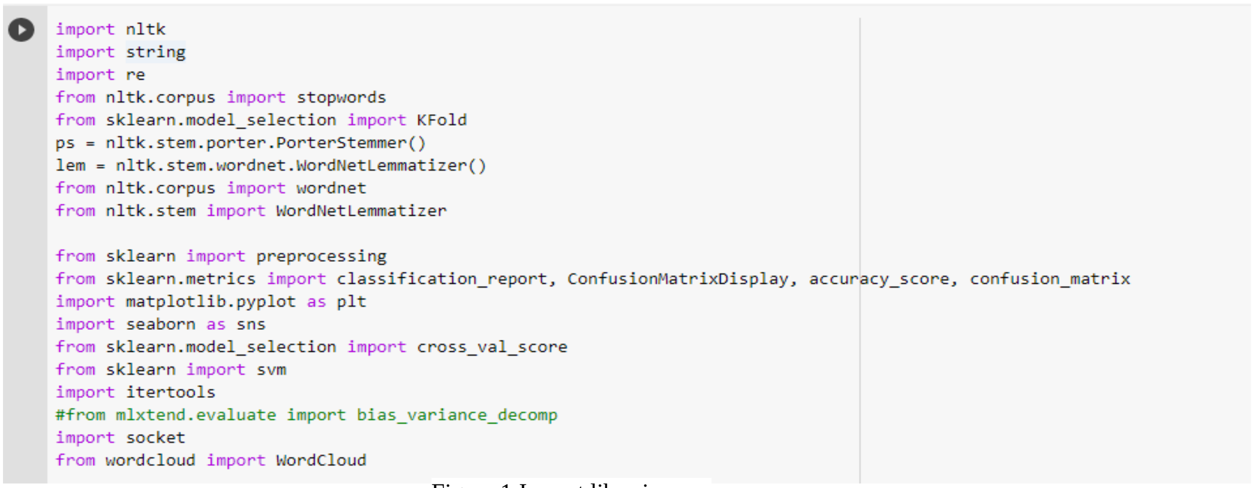
analyze similarities between partitions of different books from the same genre, use different transformations and models to classify these partitions, analyze the pros and cons of algorithms, and finally generate and communicate insights.

2.Requirements

Using a random sample of size 5 of similar books written by different authors from Gutenberg Digital Library, split the data into training and testing partitions. After that, try different transformation techniques and classification methods to accurately classify these partitions.

3. Methodology

3.1. Import libraries for use in the task

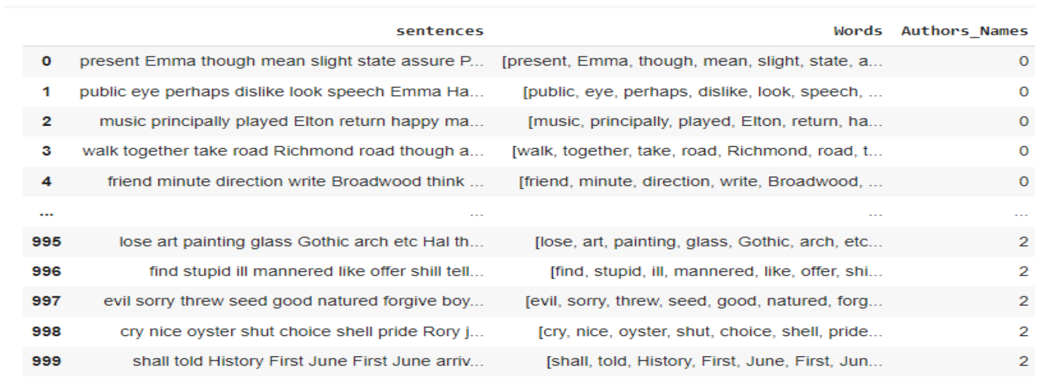


```
import nltk
import string
import re
from nltk.corpus import stopwords
from sklearn.model_selection import KFold
ps = nltk.stem.porter.PorterStemmer()
lem = nltk.stem.wordnet.WordNetLemmatizer()
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

from sklearn import preprocessing
from sklearn.metrics import classification_report, ConfusionMatrixDisplay, accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import cross_val_score
from sklearn import svm
import itertools
#from mlxtend.evaluate import bias_variance_decomp
import socket
from wordcloud import WordCloud
```

Figure 1:Import libraries

3.2. Data Preparation

We start by reading the books from NLTK library. After that, we create random samples of 200 partitions from each book. As we sample before cleaning to avoid losing important features, we prepare the records of 200 words records for each document, label them as 0, 1 and 3 etc. as per the book they belong to. Figure 2 shows a sample of the resulting data frame



	sentences	Words	Authors_Names
0	present Emma though mean slight state assure P...	[present, Emma, though, mean, slight, state, a...	0
1	public eye perhaps dislike look speech Emma Ha...	[public, eye, perhaps, dislike, look, speech, ...	0
2	music principally played Elton return happy ma...	[music, principally, played, Elton, return, ha...	0
3	walk together take road Richmond road though a...	[walk, together, take, road, Richmond, road, t...	0
4	friend minute direction write Broadwood think ...	[friend, minute, direction, write, Broadwood, ...	0
...	...	...	...
995	lose art painting glass Gothic arch etc Hal th...	[lose, art, painting, glass, Gothic, arch, etc...	2
996	find stupid ill mannered like offer shill tell...	[find, stupid, ill, mannered, like, offer, shi...	2
997	evil sorry threw seed good natured forgive boy...	[evil, sorry, threw, seed, good, natured, forg...	2
998	cry nice oyster shut choice shell pride Rory j...	[cry, nice, oyster, shut, choice, shell, pride...	2
999	shall told History First June First June arriv...	[shall, told, History, First, June, First, Jun...	2

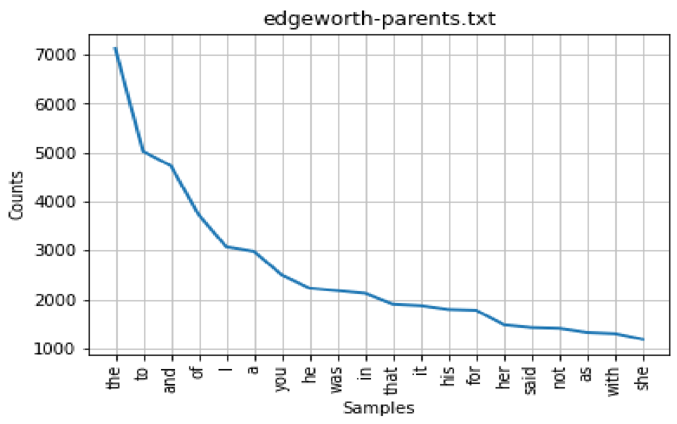
Figure 2: Reading books and generate partitions of 100 words

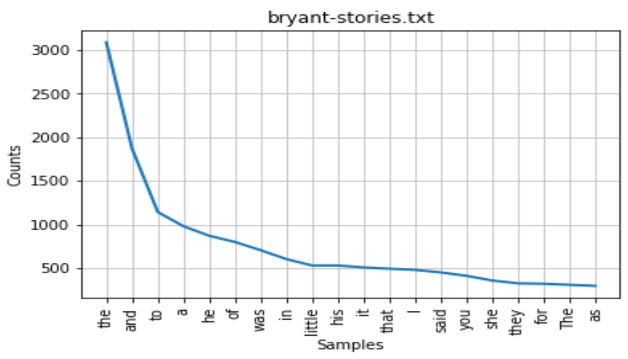
4. Preprocess the data

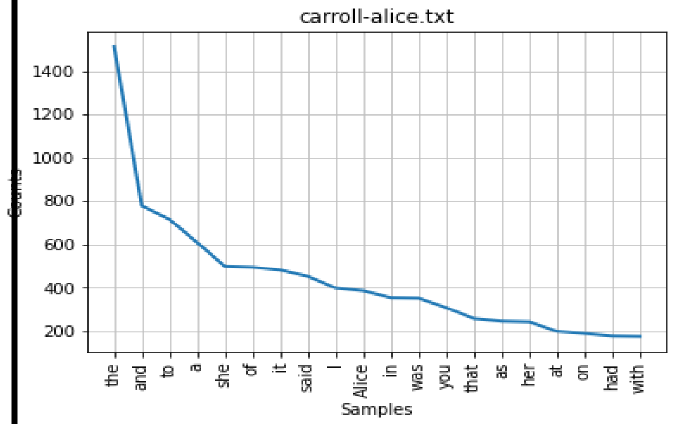
```
for i in range(len(clean_book)):
  freq = nltk.FreqDist(clean_book[i])
  freq.plot(20, cumulative=False,title=list_of_books[i])
```

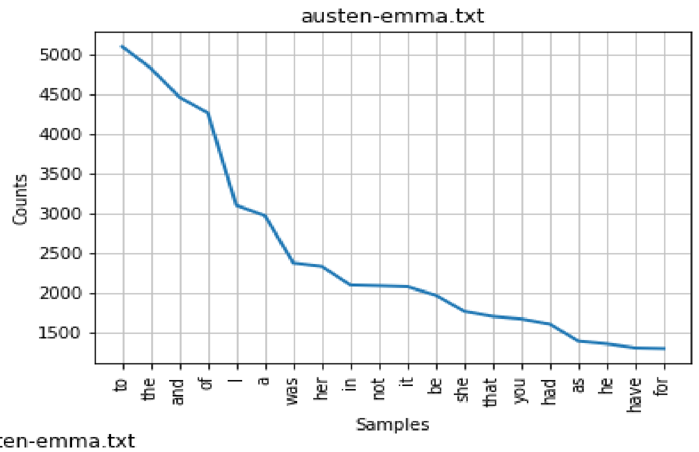
Figure 3: Plot Data after Cleansing

After reading the 5 books. we made some plots to explore various features of the raw data to get a better understanding of the structure of each book.









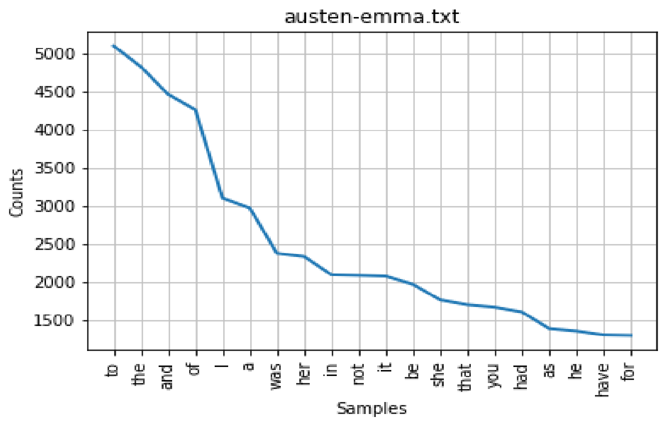
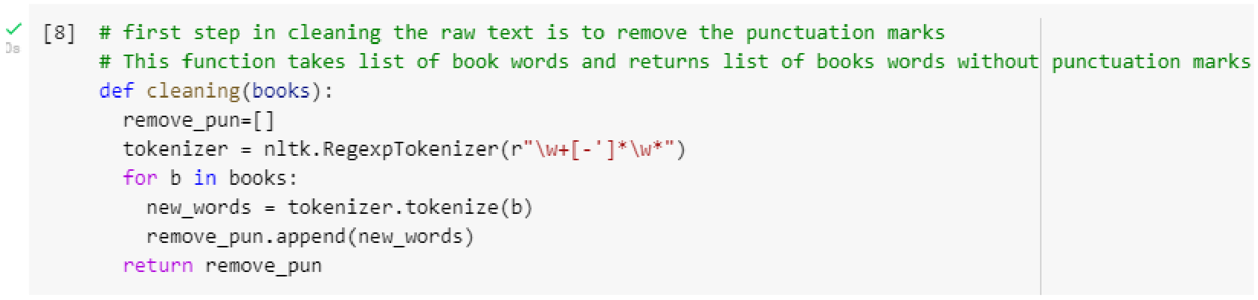


Figure 4: Data plots after Cleansing

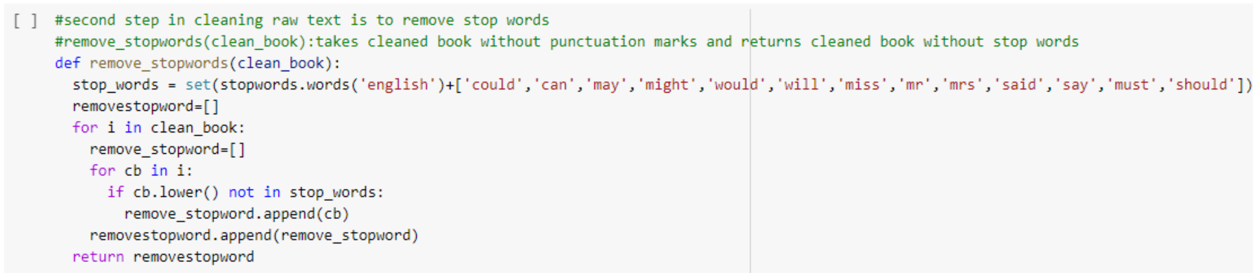
4.1. Data Cleansing

We start by removing punctuations by using **`RegexpTokenizer`** `from nltk library` and removing stop words, which are the noise in the text. Finally, we normalize text to words by using **Lemmatisation** which reduces words to their word root.



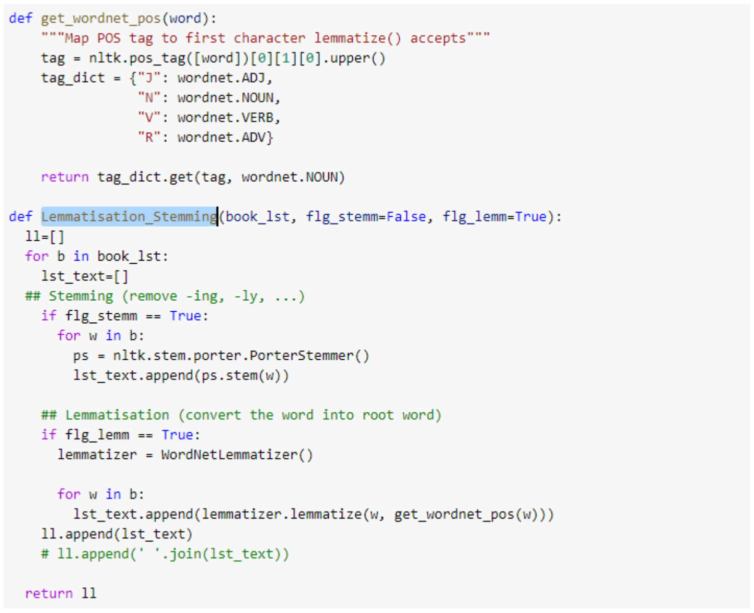
```
[8] # first step in cleaning the raw text is to remove the punctuation marks
    # This function takes list of book words and returns list of books words without punctuation marks
    def cleaning(books):
      remove_pun=[]
      tokenizer = nltk.RegexpTokenizer(r"\w+[-']*\w*")
      for b in books:
        new_words = tokenizer.tokenize(b)
        remove_pun.append(new_words)
      return remove_pun
```

Figure 5: Remove Punctuation



```
[ ] #second step in cleaning raw text is to remove stop words
    #remove_stopwords(clean_book):takes cleaned book without punctuation marks and returns cleaned book without stop words
    def remove_stopwords(clean_book):
      stop_words = set(stopwords.words('english')+['could','can','may','might','would','will','miss','mr','mrs','said','say','must','should'])
      removestopword=[]
      for i in clean_book:
        remove_stopword=[]
        for cb in i:
          if cb.lower() not in stop_words:
            remove_stopword.append(cb)
        removestopword.append(remove_stopword)
      return removestopword
```

Figure 6: Remove Stop words



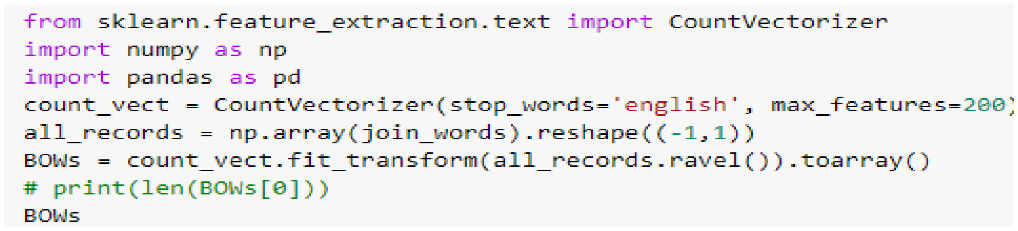
```
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}

    return tag_dict.get(tag, wordnet.NOUN)

def Lemmatisation_Stemming(book_lst, flg_stemm=False, flg_lemm=True):
  ll=[]
  for b in book_lst:
    lst_text=[]
  ## Stemming (remove -ing, -ly, ...)
    if flg_stemm == True:
      for w in b:
        ps = nltk.stem.porter.PorterStemmer()
        lst_text.append(ps.stem(w))

    ## Lemmatisation (convert the word into root word)
    if flg_lemm == True:
      lemmatizer = WordNetLemmatizer()

      for w in b:
        lst_text.append(lemmatizer.lemmatize(w, get_wordnet_pos(w)))
    ll.append(lst_text)
    # ll.append(' '.join(lst_text))

  return ll
```

Figure 7: Lemmatization

4.2: Data Transformation:

Data Transformation is very important because of the problem of text classification, we have a series of texts and their corresponding labels. But we directly can't use text for our model. We have to convert that text into some numbers or number vectors. And we applied algorithms on every transformation model (BOW, and TF-IDF, n-gram).

a) BOW Unigram



```
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
import pandas as pd
count_vect = CountVectorizer(stop_words='english', max_features=200)
all_records = np.array(join_words).reshape((-1,1))
BOWs = count_vect.fit_transform(all_records.ravel()).toarray()
# print(len(BOWs[0]))
BOWs
```

Figure 8: BOW

BOW-Cross Validations

The first transformation we apply is bag of words. It basically runs over the whole dataset, constructs its vocabulary of predefined size, and then transforms each sentence to number of occurrences of each word in it. To account for varying size sentence, we normalize the vector by the number of words in the sentence. After that, we pass the transformed vector to 3 different classifiers, namely Multinomial Naïve Bayes, K-Nearest Neighbors (K-NN), Support Vector Machine and Decision tree. Figure 8 show the cross validation of all applied models

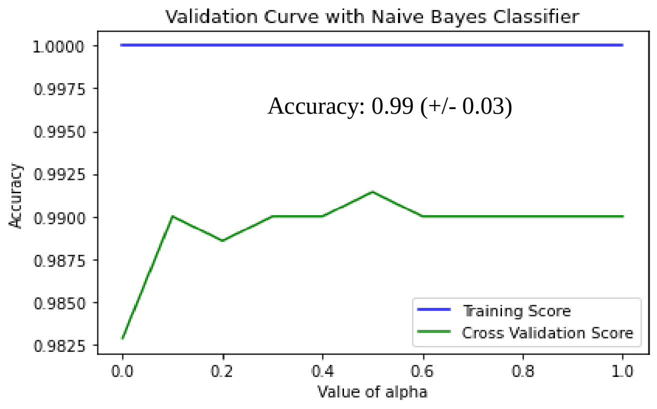


Figure 9: BOW-Naive Bayes

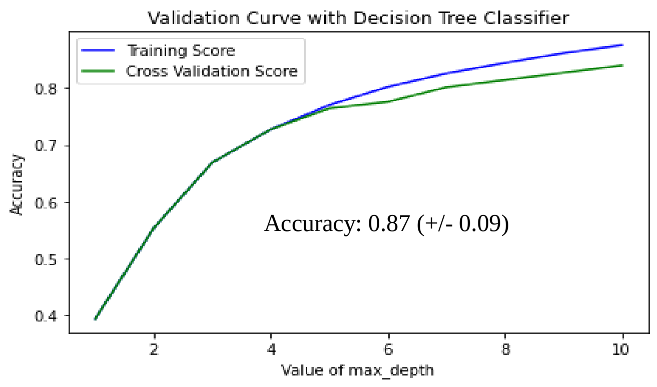


Figure 10: BOW-Decision Tree

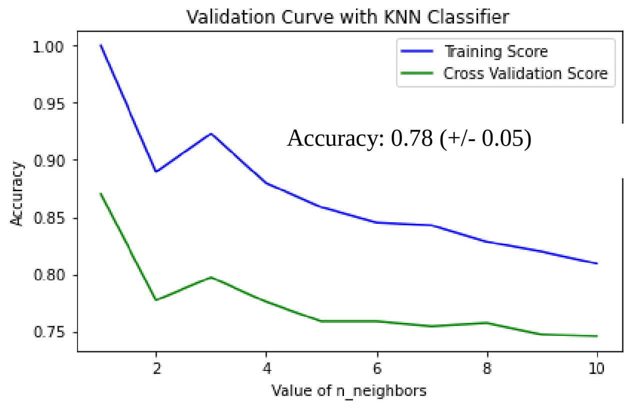


Figure 11: BOW-KNN

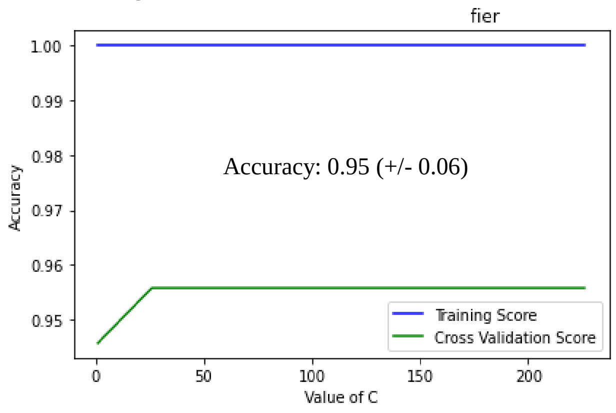
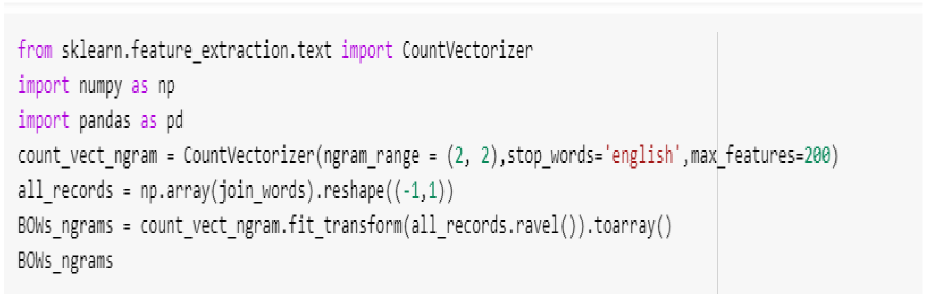


Figure 13: BOW Cross Validations

Figure 12: BOW-SVM

b) BOW bi-gram:

Aiming to capture some of the sentence structure in the books, we use the same Bag of Words approach mentioned in the previous section, but this time with each 2 consecutive words.



```
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
import pandas as pd
count_vect_ngram = CountVectorizer(ngram_range = (2, 2),stop_words='english',max_features=200)
all_records = np.array(join_words).reshape((-1,1))
BOWs_ngrams = count_vect_ngram.fit_transform(all_records.ravel()).toarray()
BOWs_ngrams
```

Figure 13: BOW Bi-gram

Bigram BOW-Cross Validations

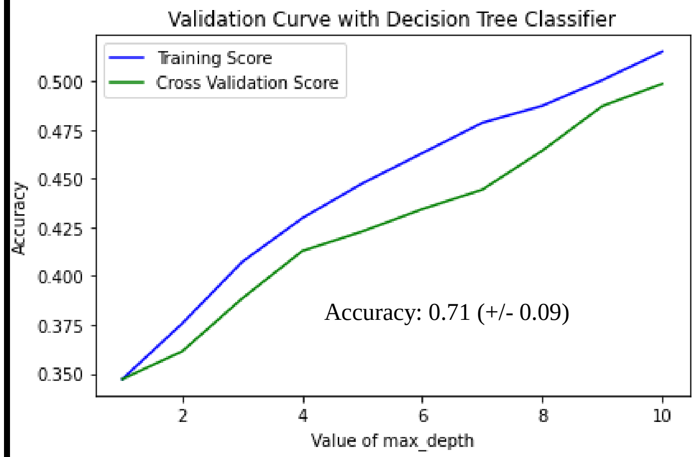


Figure 14: Bi-Gram Decision Tree

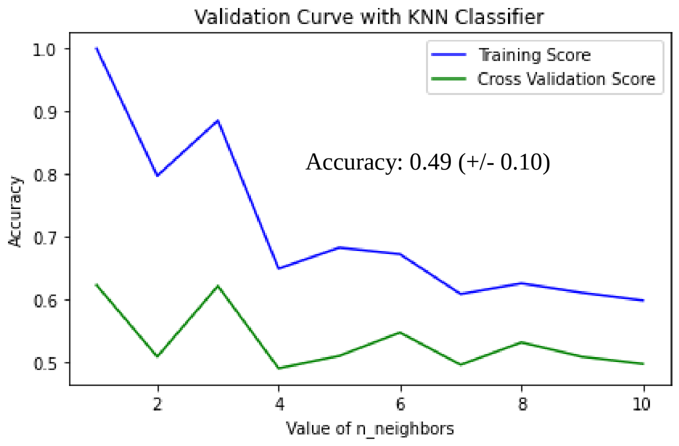


Figure 15: Bi-Gram KNN

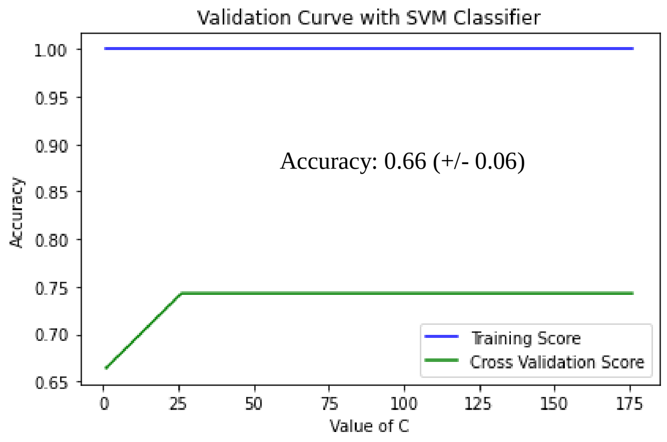


Figure 16: Bi-Gram SVM

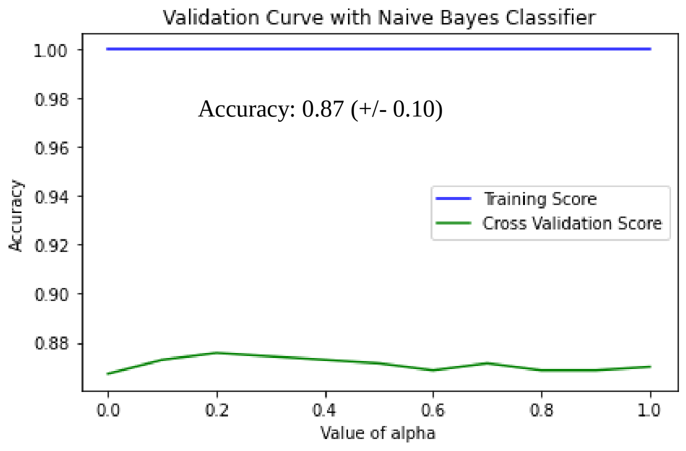
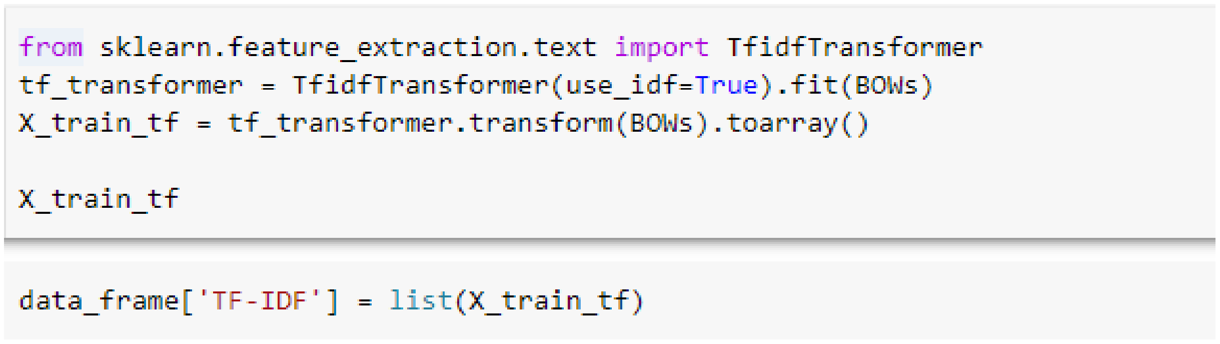


Figure 17: Bi-Gram Naïve Bayes

c)TF-IDF

We apply Tf_IDF on the data to identify the importance of words TF gives us information on how often a term appears in a document and IDF gives us information about the relative rarity of a term in the collection of documents. By multiplying these values together, we can get our final TF-IDF value



```
from sklearn.feature_extraction.text import TfidfTransformer
tf_transformer = TfidfTransformer(use_idf=True).fit(BOWs)
X_train_tf = tf_transformer.transform(BOWs).toarray()

X_train_tf
```

```
data_frame['TF-IDF'] = list(X_train_tf)
```

Figure 18. TF_IDF Transformer

TF_IDF Cross Validations

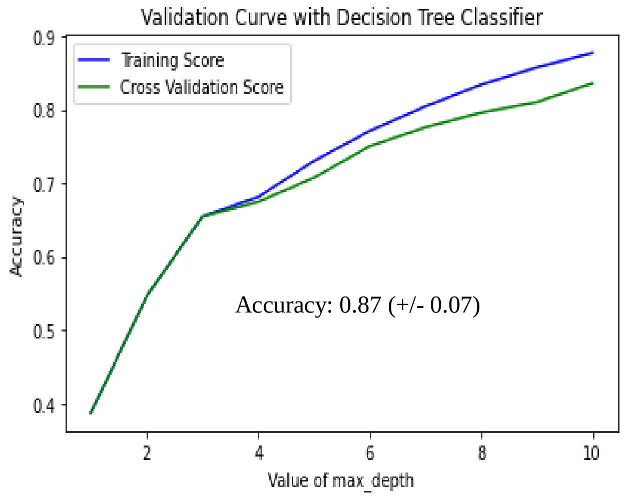


Figure 19. TF_IDF Decision Tree

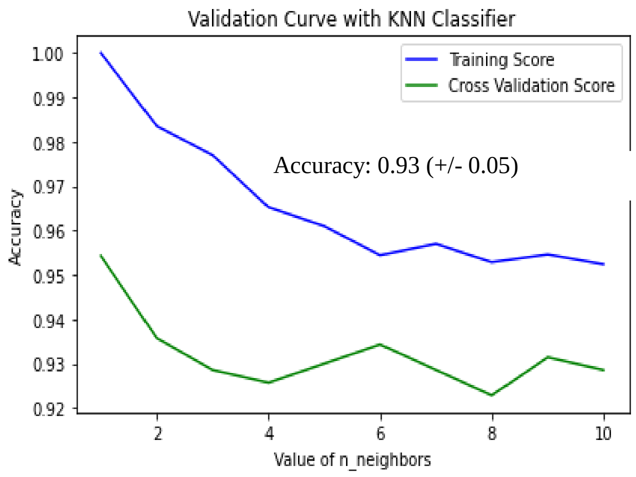


Figure 20. TF_IDF KNN

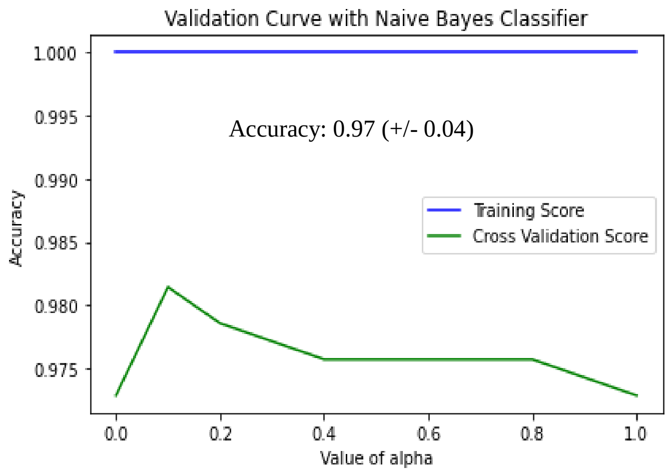


Figure 20. TF_IDF Naïve Bayes

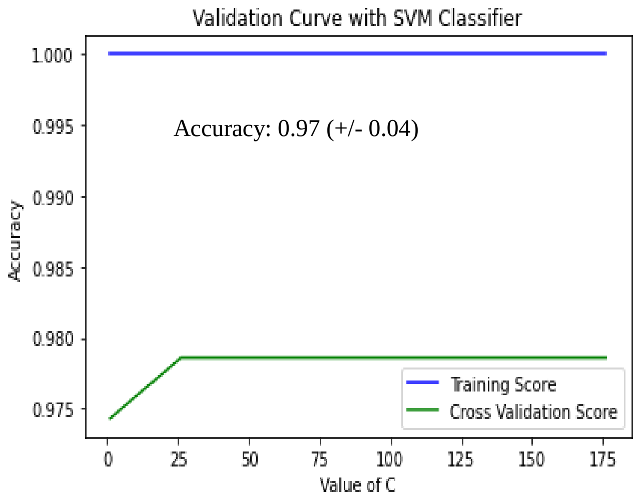


Figure 21. TF_IDF SVM

Figure 23. TF_IDF Cross Validations

5. Champion Model

After applying supervised algorithms such as Support Vector Machine, Decision Tree and KNN and Naïve Bayes to each transformer we have done. We comparing the results from evaluation for each transformer Choosing the champion model based on high accuracy, low variance to overcome the problem of overfitting and underfitting, high scores and high bias from our evaluation results.

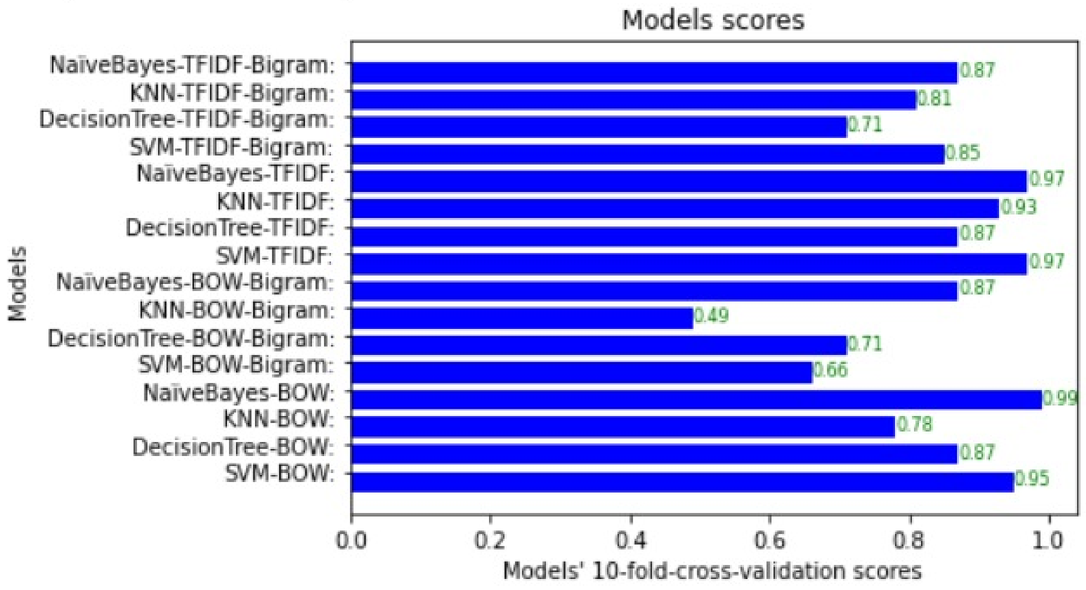
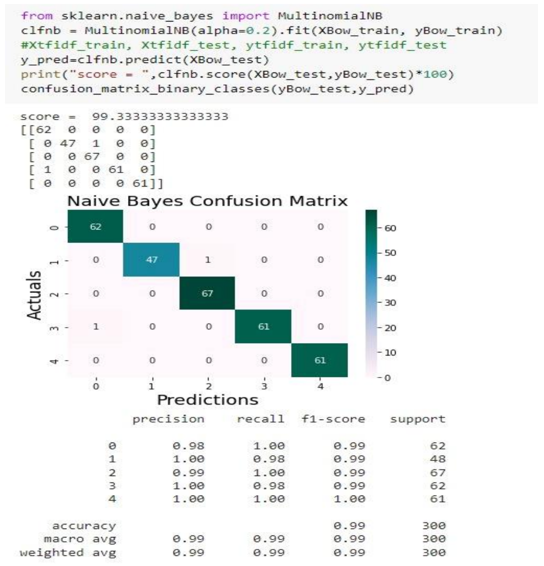


Figure 22. Models Scores

we chose Naïve Bayes model on Bag of Words (BAG) transformer with accuracy 99%.
Naïve Bayes is the winner



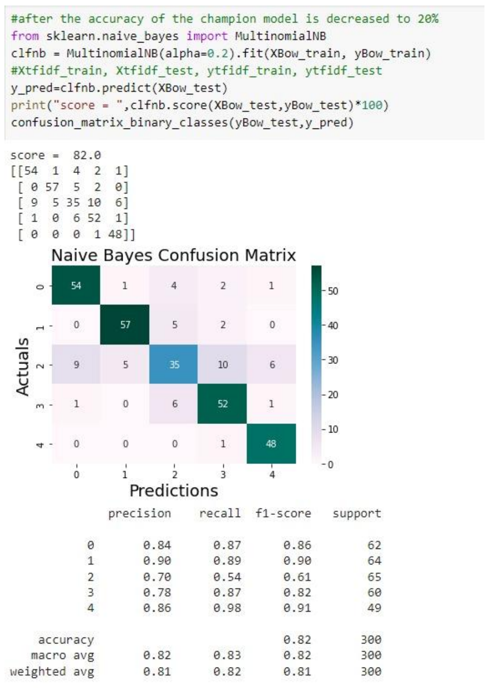
```
from sklearn.naive_bayes import MultinomialNB
clfnb = MultinomialNB(alpha=0.2).fit(XBow_train, yBow_train)
#Xtfidf_train, Xtfidf_test, ytfidf_train, ytfidf_test
y_pred=clfnb.predict(XBow_test)
print("score = ",clfnb.score(XBow_test,yBow_test)*100)
confusion_matrix_binary_classes(yBow_test,y_pred)
```

```
score =  99.33333333333333
[[62  0  0  0  0]
 [ 0 47  1  0  0]
 [ 0  0 67  0  0]
 [ 1  0  0 61  0]
 [ 0  0  0  0 61]]
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        62
           1       1.00      0.98      0.99        48
           2       0.99      1.00      0.99        67
           3       1.00      0.98      0.99        62
           4       1.00      1.00      1.00        61

    accuracy                           0.99       300
   macro avg       0.99      0.99      0.99       300
weighted avg       0.99      0.99      0.99       300
```

Figure 23. Champion model

In order to decrease the accuracy of the champion model by 20% the dimension of the data was increased by removing the stop_word () and Lemmatization () from the preprocessing pipeline and maximize the feature by adding (max_feature parameter=200)



```
#after the accuracy of the champion model is decreased to 20%
from sklearn.naive_bayes import MultinomialNB
clfnb = MultinomialNB(alpha=0.2).fit(XBow_train, yBow_train)
#Xtfidf_train, Xtfidf_test, ytfidf_train, ytfidf_test
y_pred=clfnb.predict(XBow_test)
print("score = ",clfnb.score(XBow_test,yBow_test)*100)
confusion_matrix_binary_classes(yBow_test,y_pred)
```

```
score =  82.0
[[54  1  4  2  1]
 [ 0 57  5  2  0]
 [ 9  5 35 10  6]
 [ 1  0  6 52  1]
 [ 0  0  0  1 48]]
```

	0	1	2	3	4
0	54	1	4	2	1
1	0	57	5	2	0
2	9	5	35	10	6
3	1	0	6	52	1
4	0	0	0	1	48

```
              precision    recall  f1-score   support

           0       0.84      0.87      0.86        62
           1       0.90      0.89      0.90        64
           2       0.70      0.54      0.61        65
           3       0.78      0.87      0.82        60
           4       0.86      0.98      0.91        49

    accuracy                           0.82       300
   macro avg       0.82      0.83      0.82       300
weighted avg       0.81      0.82      0.81       300
```

Figure 24. Champion model after decrease the accuracy

Error Analysis

For our champion model, we have used TF-IDF unigram for the input feature which is good as it differentiates the writing style for each author by measuring the importance of the words that are being used in their books and we had 99.3% accuracy but if the model also knows the meaning of the documents for each paragraph instead of a fixed 100 words as records it would perform better. Word-embedding would be doing the job to represent the meaning of each word and then getting the average meaning for each paragraph. Another factor is the size of training data are few as it is 60 words per record and if we have more words per record like 100 words it would perform better.

6.Conclusion

Summarizing all of the above we worked on text classification using many methods such as Bag of Words, Term Frequency Inverse Document Frequency TF IDF to split the raw text and encode it to numerical vector of number of words. applying these Transformation on Supervised algorithms to get the accuracy of each mode and choose the champion model.