uOttawa

ELG 5125: Data Science Applications Assignment 2

Prof. Arya Rahgozar

GROUP: DSA_202101_13

Table of Contents

Table of Figures

Abstract

Text classification is becoming a significantly effective part of real-life applications. In this report, five books have been selected from Gutenberg dataset. Preprocessing Data cleaning techniques have been used on the chosen set of books. Feature engineering techniques have also been applied for ease of use. Labeling the books' names will be done. A count vectorizer is later used to create a matrix of text and token counts into a bag of words. Five different classifier models will be used with all data features then only 2000 features will be used with building 5 models with previously used classifiers then TF-IDF will be applied with same five different classifier models. Random forest was chosen as the champion model and LDA was applied to it. Finally, data augmentation was done to increase training data and it gave the best outcome (Random Forest with LDA after data augmentation).

Introduction

Text processing challenges are one of the main provocations which can be compromised and dealt with by NLP (natural language processing) techniques. Gutenberg dataset is an online library of free electronic books which is very useful for learning projects. Generation of different models and picking a champion model for handling five sample books of different authors, nevertheless, analyzing the pros and cons of used machine learning algorithms and generate and link the insights using NLP techniques including removal of stop words, Bag-of-Words, TF-IDF, N-gram, cross-validation and more. Finally, Verification and validation.

Methodology

First of all, Gutenberg dataset is being used as the raw labelled data, five books have been stored in a data frame with the columns [Text] that's the content text of books and labels containing books' names. Clean the book text by replacing any character with space and also using regular expression and converting the text to lower case and split the text then removing stop words. Samples are then taken from text books and stored in the same data frame with column name text_sample and frequency of words is then plotted. Encode the names of books with numbers that will represent the labels. A count vectorizer is later used to create a matrix of text and token counts into a bag of words. The data is then being split into train and test and some algorithms are applied like SVM Classifier, Decision Tree Classifier, Naïve Bayes Classifier, K-Nearest Neighbors Classifier and print classification report for every classifier. The previously mentioned steps are then being applied with only 2000 features. Moreover, TF-IDF is then used with the same steps following it. Likely, a Bi-gram model is being used with term TF-IDF with the previously mentioned steps for models running. Ten-Fold Cross-Validation is then used for the Bi-gram model with TF-IDF. After that, LDA is then used for dimensionality reduction to apply for topic modeling. Error analysis is then applied on the picked champion model with LDA which is random forest. Data augmentation is then applied to increase the training data by generating different versions of the real dataset. Finally, another error analysis of champion model with LDA after data augmentation.

Preprocessing and Data Cleaning

Installed Packages

```
! pip install interpret-community
```
```
! pip install mlxtend --upgrade
```
```
! pip install nlpaug==1.1.0 transformers==3.0.2
! pip install snorkel==0.9.8
```
```
! pip install pycaret[full]
```
```
! pip uninstall numpy
```
```
! pip install numpy==1.20.0
```
```
! pip uninstall Jinja2
```
```
! pip install Jinja2
```

Used Libraries

```
import nltk
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```
```
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
```
```
from nltk.probability import FreqDist
```
```
from wordcloud import WordCloud, STOPWORDS
```
```
from sklearn.feature_extraction.text import CountVectorizer
```
```
from sklearn.model_selection import train_test_split
```
```
from sklearn.naive_bayes import MultinomialNB
```
```
from sklearn.metrics import confusion_matrix
```
```
from sklearn.metrics import classification_report
```
```
from mlxtend.evaluate import bias_variance_decomp
from sklearn import preprocessing
```
```
from sklearn.linear_model import PassiveAggressiveClassifier
```
```
from sklearn.tree import DecisionTreeClassifier
```
```
from sklearn.svm import SVC
```
```
from sklearn.neighbors import KNeighborsClassifier
```
```
from sklearn.model_selection import cross_val_score
```
```
from sklearn.feature_extraction.text import TfidfVectorizer
```
```
from pycaret.utils import enable_colab
```
```
from pycaret.nlp import *
```
```
from pycaret.classification import *
```
```
import nlpaug.augmenter.word as naw
```
```
from pycaret.nlp import *
```
```
from pycaret.classification import *
```

Data Selection

Choosing "gutenburg" list of books from NLTK library as the raw data to work on.

```
nltk.download('gutenberg')
nltk.corpus.gutenberg.fileids()
```

Picking five different books from the dataset.

```
book_1 = ' '.join(nltk.corpus.gutenberg.words('austen-emma.txt'))
book_2 = ' '.join(nltk.corpus.gutenberg.words('burgess-busterbrown.txt'))
book_3 = ' '.join(nltk.corpus.gutenberg.words('carroll-alice.txt'))
book_4 = ' '.join(nltk.corpus.gutenberg.words('edgeworth-parents.txt'))
book_5 = ' '.join(nltk.corpus.gutenberg.words('bryant-stories.txt'))
df = pd.DataFrame({'Text':[book_1,book_2,book_3,book_4,book_5],'label':['austen-emma','burgess-usterbrown','carroll-alice','edgeworth-parents','bryant-stories']})
```

Data Cleaning

Cleaning the data using Regex , removing stop words and applying stemming.

```
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
nltk.download('stopwords')
def clean_books(df):
  stemer=PorterStemmer()
  corpus = []
  for i in range(0,len(df)):
    # replace any character with space and leave the from (a - z )
    text = re.sub('[^A-Za-z]',' ',df['Text'][i])
    text = text.lower()
    text = text.split()
    text = [stemer.stem(word) for word in text if word not in set(stopwords.words('english'))]
    text = ' '.join(text)
    corpus.append(text)
  return corpus
```

Feature Engineering

Creating Partitions

Preparing the data by creating a function named “samples” for making 200 partitions out of each book with 100 words each.

```
corpus = clean_books(df)
```

```
def samples(book):
  l=[]
  count = 0

  while count <200:
    sample = np.random.choice(book,100)
    l.append(sample)
    count+= 1
  return l
```

Frequency Distribution and Tokenization

Using frequency distribution for showing outcomes of an experiment and using sentence tokenizer “punct” on two labels.

```
from nltk.probability import FreqDist
nltk.download('punkt')

fdist = FreqDist(nltk.word_tokenize(df_final['Text_sample'][df_final['la
bel']=='austen-emma'][10]))
print(fdist)
print(fdist.most_common(2))

fdist.plot(30,cumulative=False)
plt.show()
```

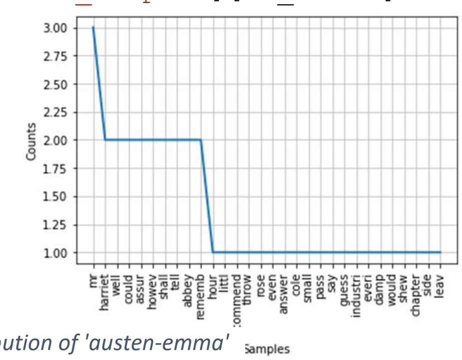


Figure 1 Frequency Distribution of 'austen-emma'

```
fdist = FreqDist(nltk.word_tokenize(df_final['Text_sample'][df_final['la
bel']=='burgess-busterbrown'][250]))
print(fdist)
print(fdist.most_common(2))

fdist.plot(30,cumulative=False)
plt.show()
```

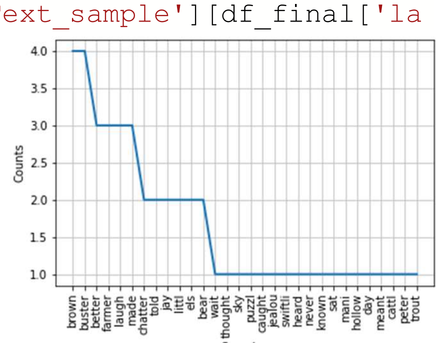


Figure 2 Frequency Distribution of 'burgess-busterbrown'

Word Cloud

Creating a function to create word cloud of any book.

```
from wordcloud import WordCloud, STOPWORDS
def worldcloud(df):
  comment_words = ''
  stopwords = set(STOPWORDS)
    # iterate through the csv file
  for val in df:
      # typecaste each val to string
      val = str(val)
      # split the value
      tokens = val.split()
      # Converts each token into lowercase
      for i in range(len(tokens)):
          tokens[i] = tokens[i].lower()
      comment_words += " ".join(tokens)+" "
  wordcloud = WordCloud(width = 800, height = 800,
                  background_color ='white',
                  stopwords = stopwords,
                  min_font_size = 10).generate(comment_words)
  # plot the WordCloud image
  plt.figure(figsize = (8, 8), facecolor = None)
  plt.imshow(wordcloud)
  plt.axis("off")
  plt.tight_layout(pad = 0)
  plt.show()
```

Then applying the function on all of the five books to show most frequent words.

```
worldcloud(emma)
```

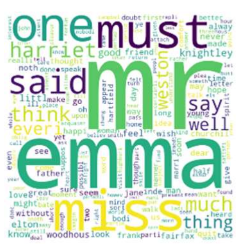


Figure 4 Word cloud of book 1

```
worldcloud(busterbrown)
```

Figure 3Word cloud of book

```
worldcloud(carroll_alice)
```

Figure 5 Word cloud of book 3

```
worldcloud(edgeworth_parents)
```

Figure 6 Word cloud of book 4

```
worldcloud(bryant_stories)
```

Figure 7 Word cloud of book 5

Modeling

Prediction Error Function

This is the main function for predicting mean squared error, bias and variance for models.

```
from mlxtend.evaluate import bias_variance_decomp
from sklearn import preprocessing
def PredictionError(Model_X_train, Model_X_test, Model_Y_train, Model_Y_
test, Model_Y_Prediction,model):
  ErrorsList = []
  TrueList = []
  PredictionList = []
  TestLabel = np.array(Model_Y_test)
  for i, z in enumerate(x_test):
    if Model_Y_Prediction[i] != TestLabel[i]:
      err = z
      ErrorsList.append(err)

      correctLabel = TestLabel[i]
      TrueList.append(correctLabel)

      prediction = Model_Y_Prediction[i]
      PredictionList.append(prediction)
  data_frame = pd.DataFrame()
  data_frame['doc_error'] = ErrorsList
  data_frame['correct']   = TrueList
  data_frame['Predicted'] = PredictionList
  #label_encoder object knows how to understand word labels.
  Label_Encoding = preprocessing.LabelEncoder()
  CopyOfXTrain = np.copy(Model_X_train)
  CopyOfXTest = np.copy(Model_X_test)
  CopyOfYTrain = np.copy(Model_Y_train)
  CopyOfYTest = np.copy(Model_Y_test)
  MSE, Bias, Variance = bias_variance_decomp(model,
np.array(CopyOfXTrain)  , Label_Encoding.fit_transform(CopyOfYTrain)  ,
np.array(CopyOfXTest)  ,  Label_Encoding.fit_transform(CopyOfYTest),
num_rounds=2, random_seed=123)
  print('THE MSE ERROR IS : &.3f' % MSE)
  print('THE BIAS IS : %.3f' % Bias)
  print('THE VARIANCE IS : %.3f' % Variance)
```

Working on all the features

Creating Bag of words

```
from sklearn.feature_extraction.text import CountVectorizer
bow = CountVectorizer()
X_bow = bow.fit_transform(X).toarray()
```

```
pd.DataFrame(X_bow,columns=bow.get_feature_names())
```

	ab	abash	abbey	abid	abil	abject	abl	ablaz	abomin	abroad	...	yonder	yo
0	0	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
995	0	0	0	0	0	0	0	0	0	0	...	0	
996	0	0	0	0	0	0	0	0	0	0	...	0	
997	0	0	0	0	0	0	0	0	0	0	...	0	
998	0	0	0	0	0	0	0	0	0	0	...	0	
999	0	0	0	0	0	0	0	0	0	0	...	0	

1000 rows × 5482 columns

Figure 8 Bag of words output

Naïve Bayes classifier

```
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
nb.fit(x_train,y_train)
```

```
y_pred=nb.predict(x_test)
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
cm
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
print(classification_report(y_train,nb.predict(x_train)))
```

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 0,  0,  0, 40,  0],
       [ 0,  0,  0,  0, 40]])
```

Figure 10 Naive Bayes confusion matrix

```
THE BIAS IS : 0.000
THE VARIANCE IS : 0.000
```

Figure 9 Naive Bayes Prediction Error

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 11 Naive Bayes train classification report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	40
1	1.00	1.00	1.00	40
2	1.00	1.00	1.00	40
3	1.00	1.00	1.00	40
4	1.00	1.00	1.00	40
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

Figure 12 Naive Bayes test classification report

Passive aggressive classifier

```
from sklearn.linear_model import PassiveAggressiveClassifier
pc = PassiveAggressiveClassifier()
pc.fit(x_train,y_train)
```

```
y_pred_pc=pc.predict(x_test)
cm = confusion_matrix(y_test,y_pred_pc)
cm
```

```
print(classification_report(y_train,pc.predict(x_train)))
print(classification_report(y_test,y_pred_pc))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,pc.fit(x_train,y_train))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 14 Passive agressive classifier train classification report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	40
1	1.00	1.00	1.00	40
2	1.00	1.00	1.00	40
3	1.00	1.00	1.00	40
4	1.00	1.00	1.00	40
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

Figure 13 Passive agressive classifier test classification report

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 0,  0,  0, 40,  0],
       [ 0,  0,  0,  0, 40]])
```

Figure 15 13 Passive agressive classifier Confusion matrix

```
THE BIAS IS : 0.000
THE VARIANCE IS : 0.000
```

Figure 16 13 Passive agressive classifier Prediction Error

Decision tree classifier

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train,y_train)
```

```
y_pred_dt=dt.predict(x_test)
cm = confusion_matrix(y_test,y_pred_dt)
cm
```

```
print(classification_report(y_train,dt.predict(x_train)))
print(classification_report(y_test,y_pred_dt))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,dt.fit(x_train,y_train))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

	precision	recall	f1-score	support
0	1.00	1.00	1.00	40
1	0.95	1.00	0.98	40
2	0.97	0.97	0.97	40
3	0.93	0.97	0.95	40
4	0.97	0.88	0.92	40
accuracy			0.96	200
macro avg	0.97	0.97	0.96	200
weighted avg	0.97	0.96	0.96	200

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 39,  0,  1],
       [ 0,  0,  1, 39,  0],
       [ 0,  2,  0,  3, 35]])
```

Figure 18 Decision tree Confusion Matrix

```
THE BIAS IS : 0.065
THE VARIANCE IS : 0.022
```

Figure 17 Decision Tree Classifier Prediction error

Figure 20 Decision Tree Classifier train classification report

Figure 19 Decision Tree Classifier test classification report

SVM classifier

```python
from sklearn.svm import SVC

svm = SVC()
svm.fit(x_train,y_train)
```

```python
y_pred_svm=svm.predict(x_test)
cm = confusion_matrix(y_test,y_pred_svm)
cm
```

```python
print(classification_report(y_train,svm.predict(x_train)))
print(classification_report(y_test,y_pred_svm))
```

```python
PredictionError(x_train,x_test,y_train,y_test,y_pred,svm.fit(x_train,y_train))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 21 SVM Classifier train classification report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	40
1	1.00	1.00	1.00	40
2	1.00	1.00	1.00	40
3	1.00	1.00	1.00	40
4	1.00	1.00	1.00	40
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

Figure 22 SVM Classifier test classification report

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 0,  0,  0, 40,  0],
       [ 0,  0,  0,  0, 40]])
```

Figure 24 SVM classifier Confusion Matrix

```
THE BIAS IS : 0.000
THE VARIANCE IS : 0.000
```

Figure 23 SVM classifier Prediction Error

K-nearest neighbors' classifier

```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
```

```python
y_pred_knn=knn.predict(x_test)
cm = confusion_matrix(y_test,y_pred_knn)
cm
```

```python
print(classification_report(y_train,knn.predict(x_train)))
print(classification_report(y_test,y_pred_knn))
```

```python
PredictionError(x_train,x_test,y_train,y_test,y_pred,knn.fit(x_train,y_train))
```

	precision	recall	f1-score	support
0	0.98	0.99	0.98	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	0.99	0.97	0.98	160
4	0.99	1.00	1.00	160
accuracy			0.99	800
macro avg	0.99	0.99	0.99	800
weighted avg	0.99	0.99	0.99	800

Figure 28 KNN classifier train Classification report

	precision	recall	f1-score	support
0	0.95	1.00	0.98	40
1	1.00	1.00	1.00	40
2	0.98	1.00	0.99	40
3	0.97	0.93	0.95	40
4	1.00	0.97	0.99	40
accuracy			0.98	200
macro avg	0.98	0.98	0.98	200
weighted avg	0.98	0.98	0.98	200

Figure 27 KNN classifier test Classification report

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 2,  0,  1, 37,  0],
       [ 0,  0,  0,  1, 39]])
```

Figure 26 KNN classifier confusion matrix

```
THE BIAS IS : 0.025
THE VARIANCE IS : 0.02
```

Figure 25 KNN classifier Prediction Error

Making ten-fold cross-validation for each classifier

```
from sklearn.model_selection import cross_val_score
nb_cv = MultinomialNB()
scores_nb = cross_val_score(nb_cv, x_train, y_train, cv=10)
scores_nb
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
pc_cv = PassiveAggressiveClassifier()
scores_pc = cross_val_score(pc_cv, x_train, y_train, cv=10)
scores_pc
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
dt_cv = DecisionTreeClassifier()
scores_dt = cross_val_score(dt_cv, x_train, y_train, cv=10)
scores_dt
```

```
array([0.9375, 0.9375, 0.975 , 0.95  , 0.8875, 0.8875, 0.9625, 0.925 ,
       0.9375, 0.975 ])
```

```
svm_cv = SVC()
scores_svm = cross_val_score(svm_cv, x_train, y_train, cv=10)
scores_svm
```

```
array([1.    , 1.    , 1.    , 1.    , 1.    , 0.9875, 1.    , 1.    ,
       1.    , 1.    ])
```

```
knn_cv = KNeighborsClassifier()
scores_knn = cross_val_score(knn_cv, x_train, y_train, cv=10)
scores_knn
```

```
array([0.9875, 1.    , 1.    , 1.    , 1.    , 0.9625, 1.    , 1.    ,
       0.9625, 0.9625])
```

Figure 29 Ten-Fold Cross-Validation with all features

Working with some of the features

By using only 2000 features

Creating Bag of words of 2000 features only

```
bow= CountVectorizer(max_features= 2000)
X_bow_2000 = bow.fit_transform(X).toarray()
pd.DataFrame(X_bow_2000,columns=bow.get_feature_names())
```

	abbey	abl	absenc	absent	absolut	abund	accept	accid	accomplish	account	...	year
0	0	0	0	0	0	0	0	0	0	0	...	1
1	0	0	0	0	0	0	0	0	0	0	...	0
2	0	0	0	0	0	0	0	0	0	0	...	0
3	0	0	1	0	0	0	0	0	1	0	...	0
4	0	0	0	0	0	1	0	0	0	0	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
995	0	0	0	0	0	0	0	0	0	0	...	0
996	0	0	0	0	0	0	0	0	0	0	...	0
997	0	0	0	0	0	0	0	0	0	0	...	0
998	0	0	0	0	0	0	0	0	0	0	...	0
999	0	0	0	0	0	0	0	0	0	0	...	0

1000 rows × 2000 columns

Figure 30 Bag of words for 2000 features

Naïve Bayes classifier

```
nb_2000 = MultinomialNB()
nb_2000.fit(x_train,y_train)
y_pred=nb_2000.predict(x_test)
cm = confusion_matrix(y_test,y_pred)
cm
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       160
           1       1.00      1.00      1.00       160
           2       1.00      1.00      1.00       160
           3       1.00      1.00      1.00       160
           4       1.00      1.00      1.00       160

    accuracy                           1.00       800
   macro avg       1.00      1.00      1.00       800
weighted avg       1.00      1.00      1.00       800
```

Figure 32 Naive Bayes classifier train Classification report

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       1.00      1.00      1.00        40
           2       1.00      1.00      1.00        40
           3       1.00      1.00      1.00        40
           4       1.00      1.00      1.00        40

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```

Figure 31 Naive Bayes classifier test Classification report

```
print(classification_report(y_test,y_pred))
print(classification_report(y_train,nb_2000.predict(x_train)))
PredictionError(x_train,x_test,y_train,y_test,y_pred,nb_2000.fit(x_train,y_train))
```

Figure 34 Naive Bayes classifier Confusion Matrix

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 0,  0,  0, 40,  0],
       [ 0,  0,  0,  0, 40]])
```

Figure 33 Naive Bayes Classifier Prediction Error

```
THE BIAS IS : 0.000
THE VARIANCE IS : 0.000
```

Passive aggressive classifier

```
pc_2000 = PassiveAggressiveClassifier()
pc_2000.fit(x_train,y_train)
y_pred_pc=pc_2000.predict(x_test)
cm = confusion_matrix(y_test,y_pred_pc)
cm
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       160
           1       1.00      1.00      1.00       160
           2       1.00      1.00      1.00       160
           3       1.00      1.00      1.00       160
           4       1.00      1.00      1.00       160

    accuracy                           1.00       800
   macro avg       1.00      1.00      1.00       800
weighted avg       1.00      1.00      1.00       800
```

Figure 36 PA train classification

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       1.00      1.00      1.00        40
           2       1.00      1.00      1.00        40
           3       1.00      1.00      1.00        40
           4       1.00      1.00      1.00        40

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```

Figure 35 PA Test Classification

```
print(classification_report(y_train,pc_2000.predict(x_train)))
print(classification_report(y_test,y_pred_pc))
print(classification_report(y_train,pc_2000.predict(x_train)))
print(classification_report(y_test,y_pred_pc))
PredictionError(x_train,x_test,y_train,y_test,y_pred,pc_2000.fit(x_train,y_train))
```

Figure 38 PA Confusion Matrix

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 0,  0,  0, 40,  0],
       [ 0,  0,  0,  0, 40]])
```

Figure 37 PA Prediction Error

```
THE BIAS IS : 0.000
THE VARIANCE IS : 0.000
```

Decision tree classifier

```
dt_2000 = DecisionTreeClassifier()
dt_2000.fit(x_train,y_train)
y_pred_dt=dt_2000.predict(x_test)
cm = confusion_matrix(y_test,y_pred_dt)
c
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       160
           1       1.00      1.00      1.00       160
           2       1.00      1.00      1.00       160
           3       1.00      1.00      1.00       160
           4       1.00      1.00      1.00       160

    accuracy                           1.00       800
   macro avg       1.00      1.00      1.00       800
weighted avg       1.00      1.00      1.00       800
```

Fiaure 40 DT Train Classification

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       1.00      1.00      1.00        40
           2       1.00      1.00      1.00        40
           3       1.00      1.00      1.00        40
           4       1.00      1.00      1.00        40

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```

Figure 39 DT Test Classification

```
print(classification_report(y_train,dt_2000.predict(x_train)))
print(classification_report(y_test,y_pred_dt))
PredictionError(x_train,x_test,y_train,y_test,y_pred,dt_2000.fit(x_train,y_train))
```

Figure 41 DT Confusion Matrix

```
array([[38,  0,  0,  2,  0],
       [ 0, 37,  0,  3,  0],
       [ 0,  0, 37,  1,  2],
       [ 3,  0,  1, 34,  2],
       [ 0,  0,  0,  1, 39]])
```

Figure 42 DT Prediction Error

```
THE BIAS IS : 0.090
THE VARIANCE IS : 0.030
```

SVM classifier

```
svm_2000 = SVC()
svm_2000.fit(x_train,y_train)
```

```
y_pred_svm=svm_2000.predict(x_test)
cm = confusion_matrix(y_test,y_pred_svm)
cm
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 44 SVM Train Classification

	precision	recall	f1-score	support
0	1.00	1.00	1.00	40
1	1.00	1.00	1.00	40
2	1.00	1.00	1.00	40
3	1.00	1.00	1.00	40
4	1.00	1.00	1.00	40
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

Figure 43 SVM Test Classification

```
print(classification_report(y_train,svm_2000.predict(x_train)))
print(classification_report(y_test,y_pred_svm))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,svm_2000.fit(x_train,y_train))
```

Figure 46 SVM Confusion Matrix

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 0,  0,  0, 40,  0],
       [ 0,  0,  0,  0, 40]])
```

Figure 45 SVM Prediction Error

```
THE BIAS IS : 0.000
THE VARIANCE IS : 0.000
```

K-nearest neighbors' classifier

```
knn_2000 = KNeighborsClassifier()

knn_2000.fit(x_train,y_train)
```

```
y_pred_knn=knn_2000.predict(x_test)
cm = confusion_matrix(y_test,y_pred_knn)
cm
```

	precision	recall	f1-score	support
0	0.99	0.99	0.99	160
1	1.00	1.00	1.00	160
2	1.00	0.99	1.00	160
3	0.99	0.99	0.99	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 47 KNN Train Classification Report

	precision	recall	f1-score	support
0	0.97	0.95	0.96	40
1	1.00	1.00	1.00	40
2	1.00	1.00	1.00	40
3	0.93	0.97	0.95	40
4	1.00	0.97	0.99	40
accuracy			0.98	200
macro avg	0.98	0.98	0.98	200
weighted avg	0.98	0.98	0.98	200

Figure 48 KNN Test Classification Report

```
print(classification_report(y_train,knn_2000.predict(x_train)))
print(classification_report(y_test,y_pred_knn))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,knn_2000.fit(x_train,y_train))
```

Figure 50 KNN Confusion Matrix

```
array([[38,  0,  0,  2,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 1,  0,  0, 39,  0],
       [ 0,  0,  0,  1, 39]])
```

Figure 49 KNN Prediction Error

```
HE BIAS IS : 0.030
HE VARIANCE IS : 0.013
```

Making ten-fold cross-validation for each classifier

Try different ten fold cross validation folds to test our models' accuracies using only 2000 words.

```
nb_cv_2000 = MultinomialNB()
scores_nb = cross_val_score(nb_cv_2000, x_train, y_train, cv=10)
scores_nb
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
pc_cv_2000 = PassiveAggressiveClassifier()
scores_pc = cross_val_score(pc_cv_2000, x_train, y_train, cv=10)
scores_pc
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
dt_cv_2000 = DecisionTreeClassifier()
scores_dt = cross_val_score(dt_cv_2000, x_train, y_train, cv=10)
scores_dt
```

```
array([0.8875, 0.95  , 0.95  , 0.925 , 0.9375, 0.9875, 0.9375, 0.9   ,
       0.95  , 0.925 ])
```

```
svm_cv_2000 = SVC()
scores_svm = cross_val_score(svm_cv_2000, x_train, y_train, cv=10)
scores_svm
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
knn_cv_2000 = KNeighborsClassifier()
scores_knn = cross_val_score(knn_cv_2000, x_train, y_train, cv=10)
scores_knn
```

```
array([1.    , 0.9875, 1.    , 0.975 , 0.975 , 1.    , 0.9875, 0.9875,
       0.9875, 0.9875])
```

Figure 51 Ten-Fold Cross-Validation with 2000 features

Applying TF-IDF

Creating TF-IDF vectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
tf = TfidfVectorizer()
X_tf=tf.fit_transform(X).toarray()
```

```
pd.DataFrame(X_tf,columns=tf.get_feature_names())
```

	abbey	abbot	abhor	abid	abil	abject	abl	ablaz	abomin
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 52 TF-IDF vectorizer

Naïve Bayes classifier

```
nb_tf = MultinomialNB()
nb_tf.fit(x_train,y_train)
```

```
y_pred=nb_tf.predict(x_test)
cm = confusion_matrix(y_test,y_pred)
cm
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 54 NB Train Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	40
1	1.00	1.00	1.00	40
2	1.00	1.00	1.00	40
3	1.00	1.00	1.00	40
4	1.00	1.00	1.00	40
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

Figure 53 NB Test Classification Report

```
print(classification_report(y_train,nb_tf.predict(x_train)))
print(classification_report(y_test,y_pred))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,nb_tf.fit(x_train,y_train))
```

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 0,  0,  0, 40,  0],
       [ 0,  0,  0,  0, 40]])
```

Figure 56 NB Confusion Matrix

```
THE BIAS IS : 0.005
THE VARIANCE IS : 0.003
```

Figure 55 NB Prediction Error

Passive aggressive classifier

```
pc_tf = PassiveAggressiveClassifier()
pc_tf.fit(x_train,y_train)
```

```
y_pred_pc=pc_tf.predict(x_test)
cm = confusion_matrix(y_test,y_pred_pc)
cm
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 58 PA Train Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	40
1	1.00	1.00	1.00	40
2	1.00	1.00	1.00	40
3	1.00	1.00	1.00	40
4	1.00	1.00	1.00	40
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

Figure 57 PA Test Classification Report

```
print(classification_report(y_train,pc_tf.predict(x_train)))
print(classification_report(y_test,y_pred_pc))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,pc_tf.fit(x_train,y_train))
```

Figure 60 PA Confusion Matrix

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 0,  0,  0, 40,  0],
       [ 0,  0,  0,  0, 40]])
```

Figure 59 PA Prediction Error

```
THE BIAS IS : 0.000
THE VARIANCE IS : 0.000
```

Decision tree classifier

```
dt_tf = DecisionTreeClassifier()
dt_tf.fit(x_train,y_train)
```

```
y_pred_dt=dt_tf.predict(x_test)
cm = confusion_matrix(y_test,y_pred_dt)
cm
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 62 DT Train Classification Report

	precision	recall	f1-score	support
0	0.95	0.90	0.92	40
1	0.97	0.95	0.96	40
2	1.00	0.93	0.96	40
3	0.80	0.90	0.85	40
4	0.88	0.90	0.89	40
accuracy			0.92	200
macro avg	0.92	0.92	0.92	200
weighted avg	0.92	0.92	0.92	200

Figure 61 DT test Classification Report

```
print(classification_report(y_train,dt_tf.predict(x_train)))
print(classification_report(y_test,y_pred_dt))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,dt_tf.fit(x_train,y_train))
```

Figure 63 DT Confusion Matrix

```
array([[36,  0,  0,  4,  0],
       [ 1, 38,  0,  1,  0],
       [ 0,  0, 37,  1,  2],
       [ 0,  1,  0, 36,  3],
       [ 1,  0,  0,  3, 36]])
```

Figure 64 DT Prediction Error

```
THE BIAS IS : 0.080
THE VARIANCE IS : 0.037
```

SVM classifier

```
svm_tf = SVC()
svm_tf.fit(x_train,y_train)
```

```
y_pred_svm=svm_tf.predict(x_test)
cm = confusion_matrix(y_test,y_pred_svm)
cm
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 66 SVM Train Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	40
1	1.00	1.00	1.00	40
2	1.00	1.00	1.00	40
3	1.00	1.00	1.00	40
4	1.00	1.00	1.00	40
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

Figure 65 SVM Test Classification Report

```
print(classification_report(y_train,svm_tf.predict(x_train)))
print(classification_report(y_test,y_pred_svm))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,svm_tf.fit(x_train,y_train))
```

Figure 68 SVM Confusion Matrix

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 0,  0,  0, 40,  0],
       [ 0,  0,  0,  0, 40]])
```

Figure 67 SVM Prediction Error

```
THE BIAS IS : 0.000
THE VARIANCE IS : 0.000
```

K-nearest neighbors’ classifier

```
knn_tf = KNeighborsClassifier()
knn_tf.fit(x_train,y_train)
```

```
y_pred_knn=knn_tf.predict(x_test)
cm = confusion_matrix(y_test,y_pred_knn)
cm
```

	precision	recall	f1-score	support
0	0.98	1.00	0.99	160
1	1.00	1.00	1.00	160
2	0.98	1.00	0.99	160
3	1.00	0.96	0.98	160
4	0.99	1.00	1.00	160
accuracy			0.99	800
macro avg	0.99	0.99	0.99	800
weighted avg	0.99	0.99	0.99	800

Figure 70 KNN Train Classification Report

	precision	recall	f1-score	support
0	0.89	1.00	0.94	40
1	1.00	1.00	1.00	40
2	0.95	1.00	0.98	40
3	1.00	0.82	0.90	40
4	1.00	1.00	1.00	40
accuracy			0.96	200
macro avg	0.97	0.97	0.96	200
weighted avg	0.97	0.96	0.96	200

Figure 69 KNN Test Classification Report

```
print(classification_report(y_train,knn_tf.predict(x_train)))
print(classification_report(y_test,y_pred_knn))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,knn_tf.fit(x_train,y_train))
```

Figure 72 KNN Confusion Matrix

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 5,  0,  2, 33,  0],
       [ 0,  0,  0,  0, 40]])
```

Figure 71 KNN Prediction Error

```
THE BIAS IS : 0.025
THE VARIANCE IS : 0.010
```

Making ten-fold cross-validation for each classifier

Try different ten fold cross validation folds to test our models accuracies that use tf-idf vectorization technique.

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,knn_tf.fit(x_train,y_train))
```

```
nb_cv_tf = MultinomialNB()
scores_nb = cross_val_score(nb_cv_tf, x_train, y_train, cv=10)
scores_nb
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
pc_cv_tf = PassiveAggressiveClassifier()
scores_pc = cross_val_score(pc_cv_tf, x_train, y_train, cv=10)
scores_pc
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
dt_cv_tf = DecisionTreeClassifier()
scores_dt = cross_val_score(dt_cv_tf, x_train, y_train, cv=10)
scores_dt
```

```
array([0.9625, 0.95  , 0.9125, 0.925 , 0.9125, 0.9   , 0.9375, 0.925 ,
       0.8875, 0.875 ])
```

```
svm_cv_tf = SVC()
scores_svm = cross_val_score(svm_cv_tf, x_train, y_train, cv=10)
scores_svm
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
knn_cv_tf = KNeighborsClassifier()
scores_knn = cross_val_score(knn_cv_tf, x_train, y_train, cv=10)
scores_knn
```

```
array([0.9875, 0.975 , 0.9875, 0.9875, 0.9875, 0.9875, 0.975 , 0.9875,
       1.    , 0.975 ])
```

Figure 73 Ten-Fold Cross Validation of TF-IDF

Applying TF-IDF with bigram

Creating Bag of words

```
tf = TfidfVectorizer(ngram_range=(2,2))
X_tf_bi=tf.fit_transform(X).toarray()
```

```
pd.DataFrame(X_tf_bi,columns=tf.get_feature_names())
```

	abbey cord	abbey cri	abbey draw	abbey hal	abbey happi	abbey henc
0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...
995	0.0	0.0	0.0	0.0	0.0	0.0
996	0.0	0.0	0.0	0.0	0.0	0.0
997	0.0	0.0	0.0	0.0	0.0	0.0
998	0.0	0.0	0.0	0.0	0.0	0.0
999	0.0	0.0	0.0	0.0	0.0	0.0

1000 rows × 86593 columns

Figure 74 TF-IDF Vectorizer with bigram

Naïve Bayes classifier

```
nb_tf_bi = MultinomialNB()
nb_tf_bi.fit(x_train,y_train)
```

```
y_pred=nb_tf_bi.predict(x_test)
cm = confusion_matrix(y_test,y_pred)
cm
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 76 NB Train Classification Report

	precision	recall	f1-score	support
0	0.82	0.90	0.86	40
1	0.80	1.00	0.89	40
2	0.61	1.00	0.75	40
3	1.00	0.38	0.55	40
4	0.80	0.50	0.62	40
accuracy			0.76	200
macro avg	0.80	0.76	0.73	200
weighted avg	0.80	0.76	0.73	200

Figure 75 NB Test Classification Report

```
print(classification_report(y_train,nb_tf_bi.predict(x_train)))
print(classification_report(y_test,y_pred))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,nb_tf_bi.fit(x_train,y_train))
```

Figure 78 NB Confusion Matrix

```
array([[36,  1,  3,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [ 7,  2, 11, 15,  5],
       [ 1,  7, 12,  0, 20]])
```

Figure 77 NB Prediction Error

```
THE BIAS IS : 0.475
THE VARIANCE IS : 0.398
```

Passive aggressive classifier

```
pc_tf_bi = PassiveAggressiveClassifier()
pc_tf_bi.fit(x_train,y_train)
```

```
y_pred_pc=pc_tf_bi.predict(x_test)
cm = confusion_matrix(y_test,y_pred_pc)
cm
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 79 PA Train Classification Report

	precision	recall	f1-score	support
0	0.75	1.00	0.86	40
1	0.87	1.00	0.93	40
2	0.80	1.00	0.89	40
3	0.81	0.42	0.56	40
4	0.87	0.65	0.74	40
accuracy			0.81	200
macro avg	0.82	0.82	0.80	200
weighted avg	0.82	0.81	0.80	200

Figure 80 PA Test Classification Report

```
print(classification_report(y_train,pc_tf_bi.predict(x_train)))
print(classification_report(y_test,y_pred_pc))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,pc_tf_bi.fit(x_train,y_train))
```

Figure 82 PA Confusion Matrix

```
array([[40,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0],
       [ 0,  0, 40,  0,  0],
       [13,  2,  4, 17,  4],
       [ 0,  4,  6,  4, 26]])
```

Figure 81 PA Prediction Error

```
THE BIAS IS : 0.240
THE VARIANCE IS : 0.113
```

Decision tree classifier

```
dt_tf_bi = DecisionTreeClassifier()
dt_tf_bi.fit(x_train,y_train)
```

```
y_pred_dt=dt_tf_bi.predict(x_test)
cm = confusion_matrix(y_test,y_pred_dt)
cm
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 83 DT Train Classification Report

	precision	recall	f1-score	support
0	0.75	0.23	0.35	40
1	0.86	0.62	0.72	40
2	0.86	0.45	0.59	40
3	0.30	0.88	0.45	40
4	0.45	0.25	0.32	40
accuracy			0.48	200
macro avg	0.65	0.48	0.49	200
weighted avg	0.65	0.48	0.49	200

Figure 84 DT Test Classification Report

```
print(classification_report(y_train,dt_tf_bi.predict(x_train)))
print(classification_report(y_test,y_pred_dt))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,dt_tf_bi.fit(x_train,y_train))
```

Figure 86 DT Confusion Matrix

```
array([[ 9,  0,  0, 29,  2],
       [ 1, 25,  1, 10,  3],
       [ 2,  0, 18, 16,  4],
       [ 0,  1,  1, 35,  3],
       [ 0,  3,  1, 26, 10]])
```

Figure 85 DT Prediction Error

```
THE BIAS IS : 0.560
THE VARIANCE IS : 0.185
```

SVM classifier

```
svm_tf_bi = SVC()
svm_tf_bi.fit(x_train,y_train)
```

```
y_pred_svm=svm_tf_bi.predict(x_test)
cm = confusion_matrix(y_test,y_pred_svm)
cm
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	160
1	1.00	1.00	1.00	160
2	1.00	1.00	1.00	160
3	1.00	1.00	1.00	160
4	1.00	1.00	1.00	160
accuracy			1.00	800
macro avg	1.00	1.00	1.00	800
weighted avg	1.00	1.00	1.00	800

Figure 88 SVM Train Classification Report

	precision	recall	f1-score	support
0	1.00	0.57	0.73	40
1	1.00	0.45	0.62	40
2	1.00	0.38	0.55	40
3	0.35	0.93	0.51	40
4	0.67	0.65	0.66	40
accuracy			0.59	200
macro avg	0.80	0.59	0.61	200
weighted avg	0.80	0.59	0.61	200

Figure 87 SVM Test Classification Report

```
print(classification_report(y_train,svm_tf_bi.predict(x_train)))
print(classification_report(y_test,y_pred_svm))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,svm_tf_bi.fit(x_train,y_train))
```

Figure 90 SVM Confusion Matrix

```
array([[23,  0,  0, 17,  0],
       [ 0, 18,  0, 16,  6],
       [ 0,  0, 15, 21,  4],
       [ 0,  0,  0, 37,  3],
       [ 0,  0,  0, 14, 26]])
```

Figure 89 SVM Prediction Error

```
THE BIAS IS : 0.645
THE VARIANCE IS : 0.492
```

K-nearest neighbors' classifier

```
knn_tf_bi = KNeighborsClassifier()
knn_tf_bi.fit(x_train,y_train)
```

```
y_pred_knn=knn_tf_bi.predict(x_test)
cm = confusion_matrix(y_test,y_pred_knn)
cm
```

	precision	recall	f1-score	support
0	0.85	0.84	0.85	160
1	0.63	1.00	0.77	160
2	0.68	0.95	0.79	160
3	0.89	0.41	0.56	160
4	0.88	0.49	0.63	160
accuracy			0.74	800
macro avg	0.79	0.74	0.72	800
weighted avg	0.79	0.74	0.72	800

Figure 91 KNN Train Classification Report

	precision	recall	f1-score	support
0	0.68	0.57	0.62	40
1	0.58	1.00	0.73	40
2	0.65	0.97	0.78	40
3	0.83	0.25	0.38	40
4	0.72	0.45	0.55	40
accuracy			0.65	200
macro avg	0.69	0.65	0.61	200
weighted avg	0.69	0.65	0.61	200

Figure 92 KNN Test Classification Report

```
print(classification_report(y_train,knn_tf_bi.predict(x_train)))
print(classification_report(y_test,y_pred_knn))
```

```
PredictionError(x_train,x_test,y_train,y_test,y_pred,knn_tf_bi.fit(x_train,y_train))
```

Figure 94 KNN Confusion Matrix

```
array([[23,  4,  8,  2,  3],
       [ 0, 40,  0,  0,  0],
       [ 0,  1, 39,  0,  0],
       [11,  9,  6, 10,  4],
       [ 0, 15,  7,  0, 18]])
```

Figure 93 KNN Prediction Error

```
THE BIAS IS : 0.435
THE VARIANCE IS : 0.228
```

Making ten-fold cross-validation for each classifier

```
nb_cv_tf_bi = MultinomialNB()
scores_nb = cross_val_score(nb_cv_tf_bi, x_train, y_train, cv=10)
scores_nb
```

```
array([0.7125, 0.725 , 0.7875, 0.7375, 0.775 , 0.8   , 0.8   , 0.75  ,
       0.7625, 0.7375])
```

```
pc_cv_tf_bi = PassiveAggressiveClassifier()
scores_pc = cross_val_score(pc_cv_tf_bi, x_train, y_train, cv=10)
scores_pc
```

```
array([0.8   , 0.725 , 0.825 , 0.7875, 0.8125, 0.8375, 0.8   , 0.8375,
       0.8875, 0.775 ])
```

```
dt_cv_tf_bi = DecisionTreeClassifier()
scores_dt = cross_val_score(dt_cv_tf_bi, x_train, y_train, cv=10)
scores_dt
```

```
array([0.4625, 0.525 , 0.45  , 0.5   , 0.575 , 0.475 , 0.4625, 0.55  ,
       0.425 , 0.4   ])
```

```
svm_cv_tf_bi = SVC()
scores_svm = cross_val_score(svm_cv_tf_bi, x_train, y_train, cv=10)
scores_svm
```

```
array([0.625 , 0.575 , 0.5625, 0.475 , 0.625 , 0.625 , 0.55  , 0.65  ,
       0.5375, 0.525 ])
```

```
knn_cv_tf_bi = KNeighborsClassifier()
scores_knn = cross_val_score(knn_cv_tf_bi, x_train, y_train, cv=10)
scores_knn
```

```
array([0.6375, 0.675 , 0.7   , 0.65  , 0.5625, 0.65  , 0.625 , 0.5625,
       0.65  , 0.6375])
```

Figure 95 Ten-Fold Cross-Validation for TF-IDF with bigram

Topic Modeling

LDA

Applying LDA

```
m1 = create_model(model='lda',num_topics = 5,
 multi_core=True)
```
```
lda_data = assign_model(m1)
```
```
lda_data.head()
```
```
lda_data.Dominant_Topic .value_counts()
```

	Text_sample_	label	Text_sample	Topic_0	Topic_1	Topic_2
0	['letter' 'morn' 'weston' 'superior' 'elton' '...	austen-emma	occur aim rate finish see well attest soon tak...	0.513481	0.475802	0.003572
1	['mr' 'mr' 'sorrow' 'convinc' 'destin' 'think'...	austen-emma	think wish happi enjoy gener age go know diffe...	0.736969	0.254367	0.002879
2	['whisper' 'induc' 'extrem' 'inde' 'cannot' 'm...	austen-emma	induc mount kind offer may unabl astonish nigh...	0.678623	0.309150	0.004062
3	['due' 'chang' 'afraid' 'born' 'spoken' 'dear'...	austen-emma	due bear speak littl present spare could forlo...	0.651861	0.284273	0.004267
4	['wish' 'might' 'willingli' 'mr' 'surpriz' 'he...	austen-emma	may willingli tell make letter passion find fl...	0.988845	0.002764	0.002781

Figure 96 Data to be used after applying LDA using 5

Top 100 words after removing stop words

```
plot_model(m1, plot = 'frequency')
```

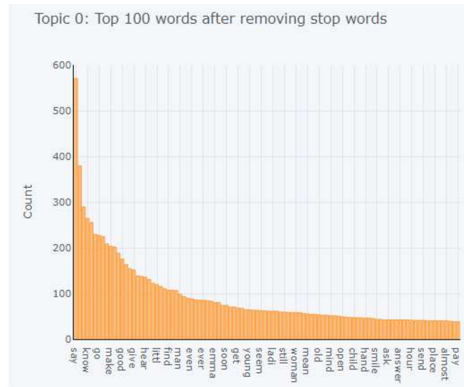


Figure 97 Top 100 words after removing stop words

Sentiment Polarity Distribution

```
plot_model(m1, plot = 'sentiment')
```

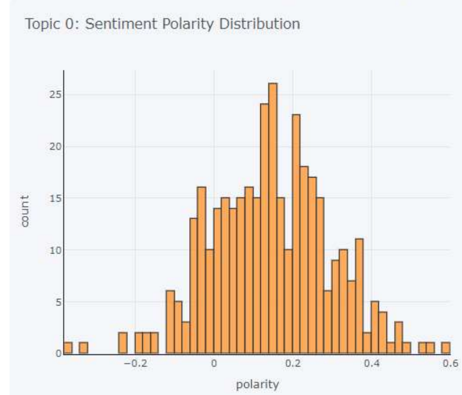


Figure 98 Sentiment Polarity Distribution

Model Description

```
model = setup(data = lda_data, target = 'label', session_id = 123)
```

0	session_id	123	29	Normalize	False
1	Target	label	30	Normalize Method	None
2	Target Type	Multiclass	31	Transformation	False
3	Label Encoded	austen-emma: 0, bryant-stories: 1, burgess-bus...	32	Transformation Method	None
4	Original Data	(1000, 6)	33	PCA	False
5	Missing Values	False	34	PCA Method	None
6	Numeric Features	5	35	PCA Components	None
7	Categorical Features	0	36	Ignore Low Variance	False
8	Ordinal Features	False	37	Combine Rare Levels	False
9	High Cardinality Features	False	38	Rare Level Threshold	None
10	High Cardinality Method	None	39	Numeric Binning	False
11	Transformed Train Set	(699, 5)	40	Remove Outliers	False
12	Transformed Test Set	(301, 5)	41	Outliers Threshold	None
13	Shuffle Train-Test	True	42	Remove Multicollinearity	False
14	Stratify Train-Test	False	43	Multicollinearity Threshold	None
15	Fold Generator	StratifiedKFold	44	Remove Perfect Collinearity	True
16	Fold Number	10	45	Clustering	False
17	CPU Jobs	-1	46	Clustering Iteration	None
18	Use GPU	False	47	Polynomial Features	False
19	Log Experiment	False	48	Polynomial Degree	None
20	Experiment Name	clf-default-name	49	Trignometry Features	False
21	USI	a69b	50	Polynomial Threshold	None
22	Imputation Type	simple	51	Group Features	False
23	Iterative Imputation Iteration	None	52	Feature Selection	False
24	Numeric Imputer	mean	53	Feature Selection Method	classic
25	Iterative Imputation Numeric Model	None	54	Features Selection Threshold	None
26	Categorical Imputer	constant	55	Feature Interaction	False
27	Iterative Imputation Categorical Model	None	56	Feature Ratio	False
28	Unknown Categoricals Handling	least_frequent	57	Interaction Threshold	None
29	Normalize	False	58	Fix Imbalance	False
30	Normalize Method	None	59	Fix Imbalance Method	SMOTE

Figure 99 Model Description with LDA data

Models’ Scores

```
compare_models()
```

Using compare_models() function to compare multiple models average performance metrics such as accuracy, AUC, Recall and etc.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
knn	K Neighbors Classifier	0.7424	0.9330	0.7427	0.7474	0.7382	0.6773	0.6806	0.118
et	Extra Trees Classifier	0.7410	0.9387	0.7430	0.7464	0.7398	0.6757	0.6774	0.552
catboost	CatBoost Classifier	0.7410	0.9404	0.7418	0.7499	0.7373	0.6755	0.6791	7.137
lightgbm	Light Gradient Boosting Machine	0.7381	0.9393	0.7397	0.7435	0.7371	0.6721	0.6738	0.360
rf	Random Forest Classifier	0.7309	0.9399	0.7326	0.7370	0.7283	0.6631	0.6657	0.524
xgboost	Extreme Gradient Boosting	0.7309	0.9393	0.7324	0.7367	0.7282	0.6631	0.6656	0.570
lr	Logistic Regression	0.7267	0.9328	0.7116	0.6342	0.6595	0.6547	0.6875	0.507
lda	Linear Discriminant Analysis	0.7210	0.9321	0.7058	0.6318	0.6531	0.6475	0.6812	0.025
dt	Decision Tree Classifier	0.7181	0.8231	0.7192	0.7216	0.7164	0.6470	0.6487	0.019
gbc	Gradient Boosting Classifier	0.7038	0.9332	0.7057	0.7058	0.7016	0.6291	0.6307	0.916
nb	Naive Bayes	0.7023	0.9335	0.7230	0.7399	0.6501	0.6307	0.6631	0.018
qda	Quadratic Discriminant Analysis	0.7009	0.9324	0.7217	0.7016	0.6478	0.6290	0.6620	0.018
svm	SVM - Linear Kernel	0.6710	0.0000	0.6756	0.6484	0.5926	0.5885	0.6327	0.063
ridge	Ridge Classifier	0.6381	0.0000	0.6239	0.5783	0.5238	0.5429	0.5898	0.017
ada	Ada Boost Classifier	0.5952	0.8979	0.5845	0.4034	0.4655	0.4892	0.5479	0.119
dummy	Dummy Classifier	0.2218	0.5000	0.2000	0.0492	0.0806	0.0000	0.0000	0.015

Figure 100 Models' Scores

Evaluation and Augmentation

Champion Model

AUC-ROC for champion model in each book

```
plot_model(rf, plot = 'auc')
```

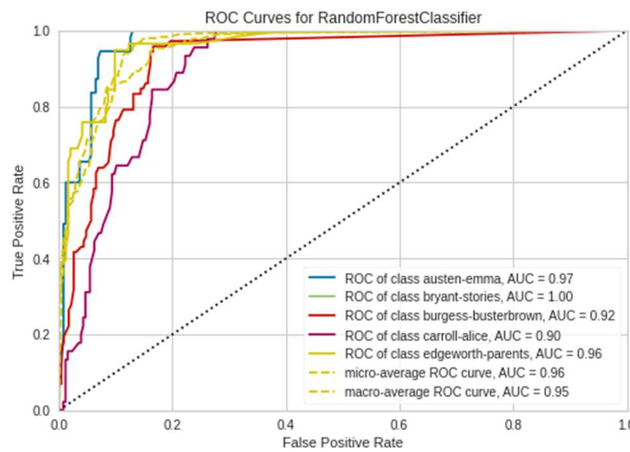


Figure 101 AUC-ROC of RF model

Precision-Recall curve of the champion model

```
plot_model(rf, plot = 'pr')
```

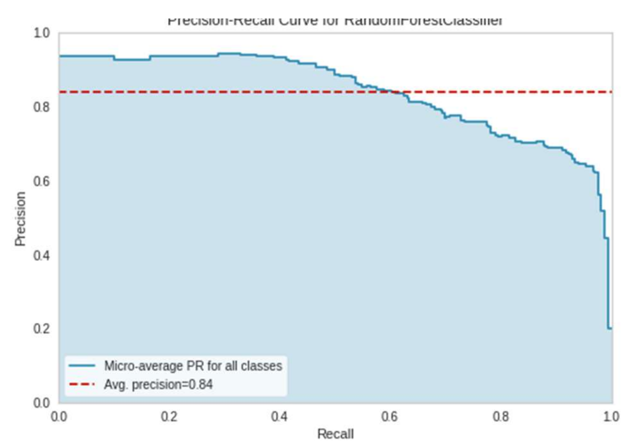


Figure 102 Precision-Recall Curve of RF model

Confusion Matrix of the champion model

```
plot_model(rf, plot = 'confusion_matrix')
```

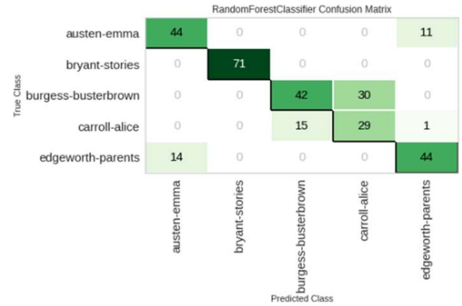


Figure 103 Confusion matrix of RF model

Class Prediction Error of the champion model

```
plot_model(rf, plot = 'error')
```

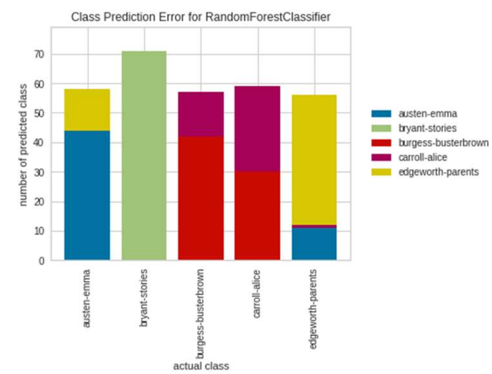


Figure 104 Class Prediction error of RF model

Decision Boundaries of the champion model

```
plot_model(rf, plot = 'boundary')
```

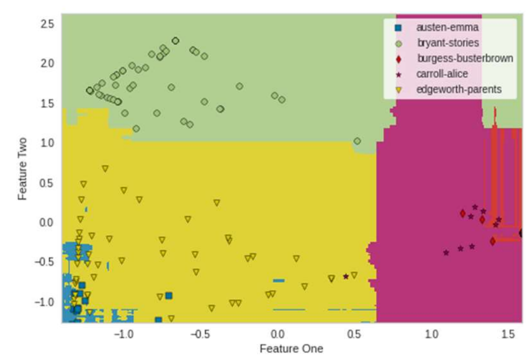
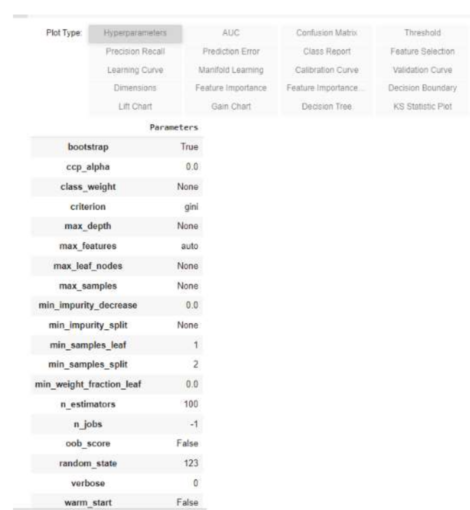


Figure 105 Decision boundaries of RF model

General Evaluation of the champion model

```
evaluate_model(rf)
```



	Parameters
bootstrap	True
ccp_alpha	0.0
class_weight	None
criterion	gini
max_depth	None
max_features	auto
max_leaf_nodes	None
max_samples	None
min_impurity_decrease	0.0
min_impurity_split	None
min_samples_leaf	1
min_samples_split	2
min_weight_fraction_leaf	0.0
n_estimators	100
n_jobs	-1
oob_score	False
random_state	123
verbose	0
warm_start	False

Figure 106 General evaluation of RF model

Error Analysis of champion model LDA + Random Forest

```
# Permutation Feature Importance
interpret_model(rf, plot = 'pfi')
```

using Feature permutation importance to show importance of features

The bar chart shows the model's view of the relative feature importance

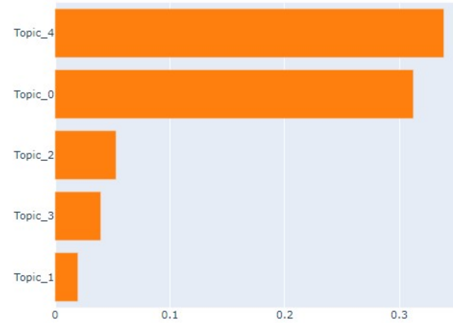


Figure 107 Permutation Feature Importance

```
interpret_model(rf)
```

The bar chart shows the model's view of the relative feature importance

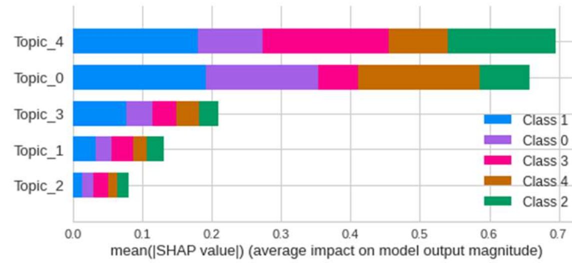


Figure 108 Mean SHAP value

```
#Morris Sensitivity Analysis
interpret_model(rf, plot = 'msa')
```

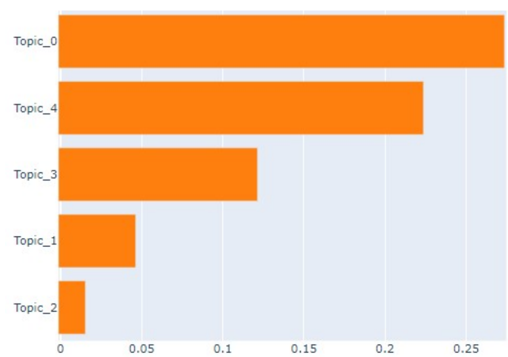


Figure 109 Morris Sensitivity Analysis

```
 interpret_model(rf, plot = 'pdp')
# Partial dependence plots.
Partial dependence plots (PDP)
 show the dependence between the
 target response and a set of input
 features of interest, marginalizing
over the values of all other input
features
```

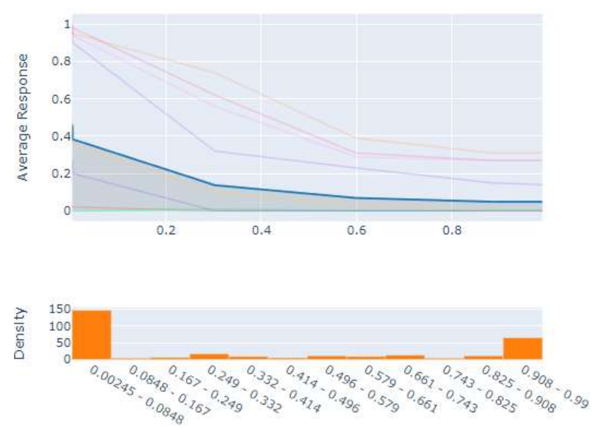


Figure 110 Partial Dependence Plots

Data Augmentation

```python
import nlpaug.augmenter.word as naw
substitution = naw.ContextualWordEmbsAug(model_path="distilbert-base-uncased", action="substitute")
insertion = naw.ContextualWordEmbsAug(model_path="distilbert-base-uncased", action="insert")
```

```python
corpus = []
for i in range(len(df_final['Text_sample'])):
  augmented_text = insertion.augment(str(df_final['Text_sample'][i]))
  corpus.append(augmented_text)
df = pd.DataFrame(corpus)
```

After going through same procedures before using data augmentation which are:

Models' Scores

```python
compare_models()
```

Using compare_models() function to compare multiple models average performance metrics such as accuracy, AUC, Recall and etc..

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
xgboost	Extreme Gradient Boosting	0.9871	0.9999	0.9642	0.9884	0.9868	0.9814	0.9817	0.400
rf	Random Forest Classifier	0.9843	0.9998	0.9596	0.9858	0.9839	0.9773	0.9776	0.522
lightgbm	Light Gradient Boosting Machine	0.9843	0.9998	0.9644	0.9858	0.9840	0.9773	0.9776	0.289
catboost	CatBoost Classifier	0.9843	0.9998	0.9702	0.9862	0.9843	0.9774	0.9777	6.973
ridge	Ridge Classifier	0.9828	0.0000	0.9293	0.9842	0.9816	0.9750	0.9755	0.015
svm	SVM - Linear Kernel	0.9814	0.0000	0.9546	0.9846	0.9811	0.9733	0.9739	0.062
gbc	Gradient Boosting Classifier	0.9814	0.9997	0.9473	0.9824	0.9807	0.9730	0.9734	0.718
lda	Linear Discriminant Analysis	0.9814	0.9998	0.9872	0.9876	0.9829	0.9735	0.9741	0.018
et	Extra Trees Classifier	0.9814	0.9998	0.9508	0.9834	0.9809	0.9731	0.9735	0.459
knn	K Neighbors Classifier	0.9800	0.9987	0.9481	0.9818	0.9794	0.9710	0.9714	0.118
dt	Decision Tree Classifier	0.9786	0.9859	0.9596	0.9814	0.9786	0.9692	0.9697	0.017
lr	Logistic Regression	0.9785	0.9997	0.9217	0.9798	0.9771	0.9687	0.9692	0.507
ada	Ada Boost Classifier	0.9714	0.9956	0.9395	0.9735	0.9713	0.9587	0.9591	0.116
nb	Naive Bayes	0.9614	0.9989	0.9708	0.9734	0.9642	0.9452	0.9467	0.017
qda	Quadratic Discriminant Analysis	0.9528	0.9979	0.9567	0.9663	0.9560	0.9331	0.9348	0.018
dummy									0.016

Figure 111 Models' Scores - Data augmentation

Champion Model

AUC-ROC for champion model in each book

```python
plot_model(rf, plot = 'auc')
```

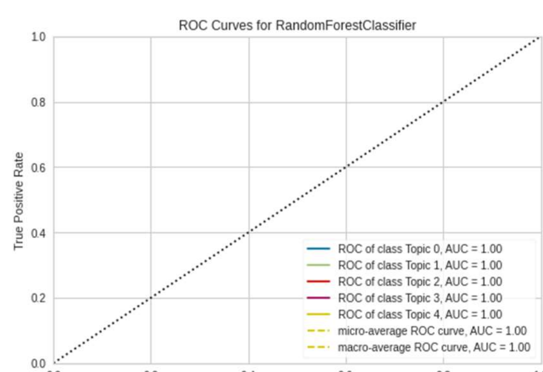


Figure 112 AUC-ROC - Data augmentation

Precision-Recall curve of the champion model

```
plot_model(rf, plot = 'pr')
```

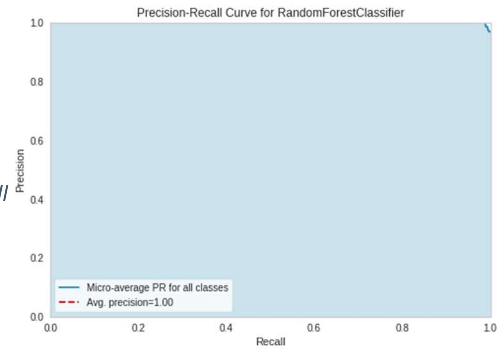


Figure 113 Precision-Recall curve of RF model - Data augmentation

Confusion Matrix of the champion model

```
plot_model(rf, plot = 'confusion_matrix')
```

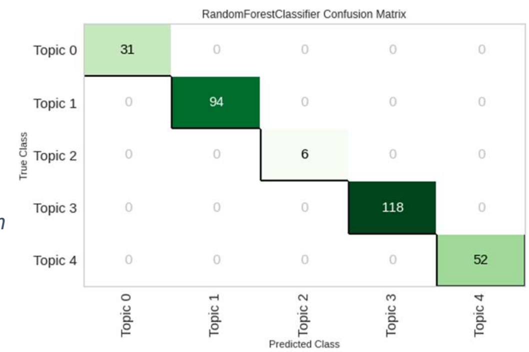


Figure 114 Confusion matrix of RC model - Data augmentation

Class Prediction Error of the champion model

```
plot_model(rf, plot = 'error')
```

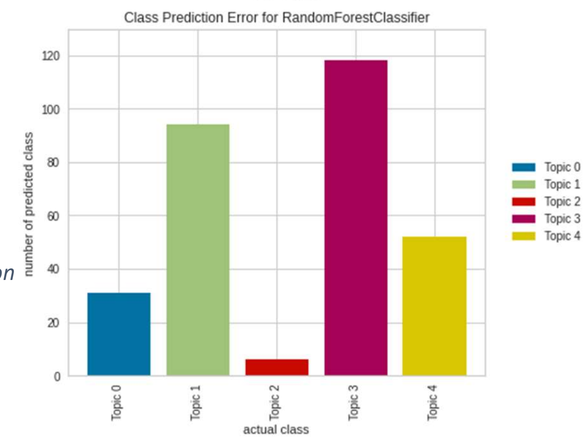
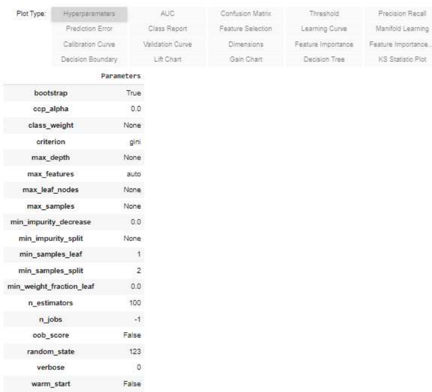


Figure 115 Class Prediction Error of RF model - Data augmentation

General Evaluation of the champion model

```
evaluate_model(rf)
```



	Parameters
bootstrap	True
ccp_alpha	0.0
class_weight	None
criterion	gini
max_depth	None
max_features	auto
max_leaf_nodes	None
max_samples	None
min_impurity_decrease	0.0
min_impurity_split	None
min_samples_leaf	1
min_samples_split	2
min_weight_fraction_leaf	0.0
n_estimators	100
n_jobs	-1
oob_score	False
random_state	123
verbose	0
warm_start	False

Figure 116 General evaluation of RF model - Data augmentation

Error Analysis of champion model LDA + Random Forest with augmentation

```
# Permutation Feature Importance
interpret_model(rf, plot = 'pfi')
```

using Feature permutation importance after data augmentation to show importance of features

The bar chart shows the model's view of the relative feature importance

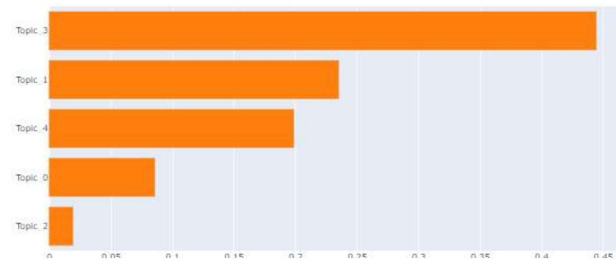


Figure 117 Permutation Feature Importance - Data augmentation

```
interpret_model(rf)
```

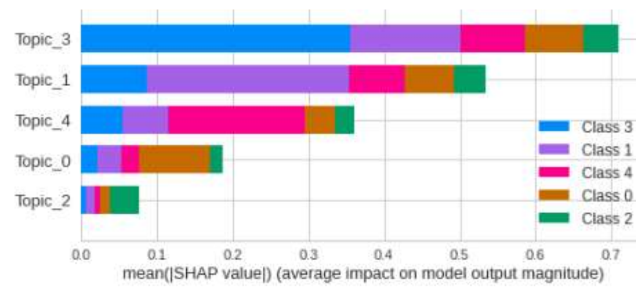


Figure 118 Mean SHAP values - Data augmentation

```
#Morris Sensitivity Analysis
interpret_model(rf, plot = 'msa')
```

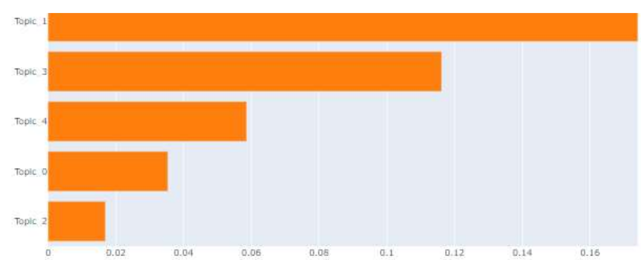


Figure 119 Morris Sensitivity Analysis - Data augmentation

```
interpret_model(rf, plot = 'pdp')
# Partial dependence plots.
Partial dependence plots (PDP) show
 the dependence between the target
response and a set of input features
 of interest, marginalizing over the
values of all other input features
```

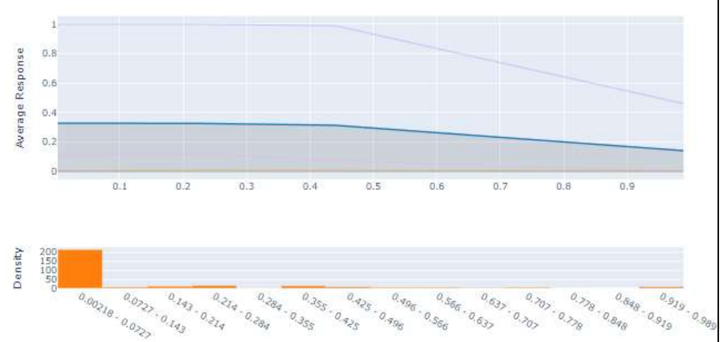


Figure 120 Partial Dependence Plots - Data augmentation

Conclusion

Finally, five models have been built for the classification task. These models are Naive Bayes Classifier, Passive Aggressive Classifier, SVM Classifier, Decision Tree Classifier, and K-nearest neighbor Classifier (KNN). The champion model is KNN which has more than 97% accuracy. After that, data augmentation was used to increase data size to enhance the models' accuracies and apply latent Dirichlet allocation (LDA) for topic modeling task and then Random Forest algorithm is applied to classify the topic of each document in corpus and it showed an outstanding result as it has AUC-ROC of approximately 0.999 and accuracy of approximately 0.984. At last, some error analyses were performed on the champion model which is Random Forest with LDA and data augmentation which was better than TF-IDF and TF-IDF with bigram model, also, Topic 3 was found to be the most important feature compared to others and also Random Forest model was found to be frequently confused when trying to classify "burgess-busterbrown" and "carroll-alice" books.

References

[1] Amato, F., Coppolino, L., Cozzolino, G., Mazzeo, G., Moscato, F., & Nardone, R. (2021). Enhancing random forest classification with NLP in DAMEH: A system for DAta Management in eHealth Domain. *Neurocomputing*, *444*, 79–91. https://doi.org/10.1016/j.neucom.2020.08.091

[2] *Build software better, together*. (n.d.). GitHub. Retrieved May 29, 2022, from https://github.com/topics/svm-classifier

[3] Feng, S. Y., Gangal, V., Wei, J., Chandar, S., Vosoughi, S., Mitamura, T., & Hovy, E. (2021). A Survey of Data Augmentation Approaches for NLP. *ArXiv:2105.03075 [Cs]*. https://arxiv.org/abs/2105.03075

[4] Maier, D., Waldherr, A., Miltner, P., Wiedemann, G., Niekler, A., Keinert, A., Pfetsch, B., Heyer, G., Reber, U., Häussler, T., Schmid-Petri, H., & Adam, S. (2018). Applying LDA Topic Modeling in Communication Research: Toward a Valid and Reliable Methodology. *Communication Methods and Measures*, *12*(2-3), 93–118. https://doi.org/10.1080/19312458.2018.1430754

[5] Pal, K., & Patel, Biraj. V. (2020, March 1). *Data Classification with k-fold Cross Validation and Holdout Accuracy Estimation Methods with 5 Different Machine Learning Techniques*. IEEE Xplore. https://doi.org/10.1109/ICCMC48092.2020.ICCMC-00016

[6] Riza, L. S., Pertiwi, A. D., Rahman, E. F., Munir, M., & Abdullah, C. U. (2019). Question Generator System of Sentence Completion in TOEFL Using NLP and K-Nearest Neighbor. *Indonesian Journal of Science and Technology*, *4*(2), 294–311. https://doi.org/10.17509/ijost.v4i2.18202

[7] Walkowiak, T., Datko, S., & Maciejewski, H. (2018). Bag-of-Words, Bag-of-Topics and Word-to-Vec Based Subject Classification of Text Documents in Polish - A Comparative Study. *Contemporary Complex Systems and Their Dependability*, 526–535. https://doi.org/10.1007/978-3-319-91446-6_49

[8] Yang, F.-J. (2018, December 1). *An Implementation of Naive Bayes Classifier*. IEEE Xplore. https://doi.org/10.1109/CSCI46756.2018.00065