Master of Engineering Electrical and Computer Engineering 2021

University of Ottawa

Classification Assignment

Authors: Seifeldin Abdelghany, Sara Abdelghafar, Omar Sorour and Zeyad Elsayed

Supervisor: Dr. Arya Rahgozar

Group Number: 8

Submission date: May 30$^{\text{th}}$, 2021

Abstract

Recently, the rise of big data and natural language processing algorithm has gained a huge amount interest from the competing technology companies. Text classification is one of the problems solved by NLP algorithm. Text classification refers to the process of supervised learning of specified text based on fixed rules. This report describes the various techniques of text classification, including text representation, feature selection and classification algorithms, and draws the basic ideas, advantages and disadvantages of several current mainstream classification techniques.

Contents

List of Figures

Chapter 1

Introduction

Text classification can be simply explained as the process of passing the unknown category of text through some rules to obtain in which category the text belongs to. Assuming that there is an objective function, the training of the classification method is carried out through a large amount of corpora, and then the classifier is trained to obtain the model, and then according to the classifier that has been obtained, the various feature item sets are sorted into the initially defined category number. Text classification is a process in which the final classification label of the text in the previous category is specified, and then associates the specified text with the category number according to the content of the text[1].

Chapter 2

Code Implementation

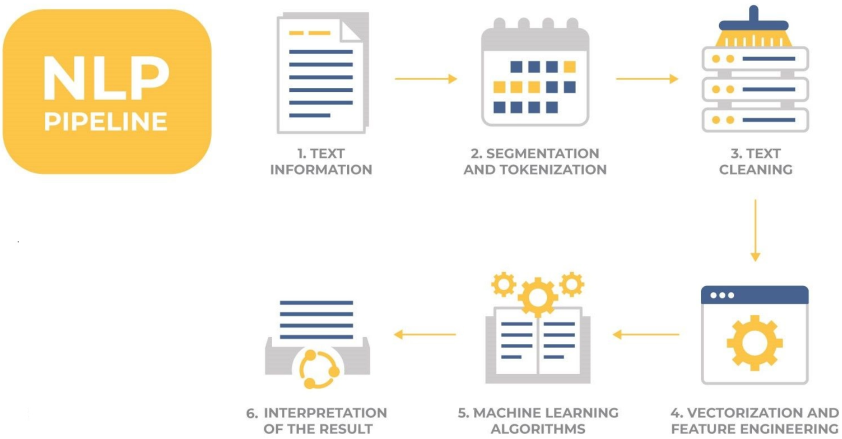


Figure 2.1: NLP Pipeline

To classify the text, it should pass through modeling process as in Fig 2.1.

2.1 Setting up the Environment

2.1.1 Importing Libraries

```
import matplotlib.pyplot as plt
import nltk
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from collections import Counter
import random
from random import randrange
from random import sample
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sn
nltk.download('gutenberg')
nltk.download('punkt')
nltk.download('stopwords')
stop_words=set(stopwords.words("english"))
```

Figure 2.2: Environment

Starting by setting up our environment, so we imported some libraries to help us in coding, as in Fig 2.2.

2.1.2 Importing Data

```
files = nltk.corpus.gutenberg.fileids() #array that contains all files retreived from gutenberg
texts = []
print(files)

def get_texts(files, texts): # function to get the books from gutenburg
  prev_vals = [0,5,6,10,16] #5 books from different authors that we believe have the same or similar genres
  for i in range(len(prev_vals)):
    texts.append(files[prev_vals[i]])

get_texts(files, texts)
print(texts)
print(len(files))
```

Figure 2.3: Importing Dataset from Gutenberg

We imported five books from Gutenberg. We chose five books belonging to five different authors as in Fig 2.3. The five books we chose were austen-emma.txt, bryant-stories.txt, burgess-busterbrown.txt, chesterton-thursday.txt and shakespeare-macbeth.txt.

2.2 Cleaning, Partitioning and Labeling Data

```
def get_df(texts): #function that takes the list of books and returns a df with 200 cleaned samples for each book
  filtered_sentences = []
  labels = []
  for i in range(len(texts)):
    text = nltk.corpus.gutenberg.raw(texts[i]) #get the book
    tokenized_word=nltk.word_tokenize(text) #tokenize the words
    cleaned_words = [word for word in tokenized_word if word.isalnum()] #clean the words from symbols and keep alphanumeric characters instead of using regex
    filtered_words=[]
    for w in cleaned_words:# Clean the words yet again from stop words.
        if w not in stop_words:
            filtered_words.append(w)
    #filtered_sentences.append(' '.join(filtered_words[0:4]))
    random_numbers = []
    for c in range(200):
      while True: # To stay in loop and change random number if it was used before
        x = randrange(0,len(filtered_words)-100) #generate random number from 0 to length of string - 100 (to be able to take last 100 words)
        if (x not in random_numbers): # make sure the number was not used before
          random_numbers.append(x) #add random number to list
          filtered_sentences.append(' '.join(filtered_words[x:x+100])) #add 100 words from random position to array.
          labels.append(texts[i][0:-4]) #append the label of the book while removing last 4 characters ".txt"
          break

  print(len(filtered_sentences))
  print(len(filtered_words))
  df = pd.DataFrame({'label': labels, 'sample': filtered_sentences})

  return df, filtered_sentences
df, s = get_df(texts)
```

Figure 2.4: Preparing Data

We cleaned the words from symbols and removing any garbage characters by keeping alphanumeric characters only, lastly we removed any stop words from text.

Then we performed partitioning, by taking random 200 partition from each book, each partition consists of 100 words. So we tokenized text where we transform sentences into words using nltk word tokenize and appended these words forming a sentence.

Then we performed labeling by appending author name to the list of labels simultaneously with the appending of sentence as in Fig 2.4.

This function returns a dataframe of one thousand records(200 for each of the 5 books) and two columns, the first is for sentences and the second is the label we trying to predict as in Fig 2.5.

```
[ ] df
```

	label	sample
0	austen-emma	Woodhouse good My father tried formerly withou...
1	austen-emma	good You surprize Emma must Harriet good suppl...
2	austen-emma	dear little boys I must say Aunt Emma time I t...
3	austen-emma	could never bear think strange hands mere comm...
4	austen-emma	feelings said Emma guess I listen pleasure wou...
...	...	...
995	shakespeare-macbeth	occasion call vs And shew vs Watchers lost So ...
996	shakespeare-macbeth	Gent Good night good Doctor Exeunt Scena Secun...
997	shakespeare-macbeth	comes Fit againe I else beene perfect Whole Ma...
998	shakespeare-macbeth	bides With twenty trenched gashes head The lea...
999	shakespeare-macbeth	Well I thither Macd Well may see things wel do...

1000 rows × 2 columns

Figure 2.5: Dataframe

Then we split the data to 70% training and 30% testing as in Fig 2.6.

```
X_train, X_test, y_train, y_test = train_test_split(df["sample"], df["label"], test_size=0.3, random_state=42)
```

Figure 2.6: Splitting Data

2.3 Text Transformation

We used three transformation techniques,

- BOW
- TF-IDF
- N-gram

2.3.1 BOW

```
[ ]  count_vect = CountVectorizer()
     X_train_counts = count_vect.fit_transform(X_train)
     X_train_counts.shape
     X_test_counts= count_vect.transform(X_test)
     X_test_counts.shape
     # #print(X_train_counts)
```

Figure 2.7: BOW

A bag of words is a representation of text that describes the occurrence of words within a document. We used sklearn count vectorizer which converts a collection of text documents to a matrix of token counts as in Fig 2.7.

2.3.2 TF-IDF

```
from sklearn.feature_extraction.text import TfidfTransformer
tf_transformer = TfidfTransformer(use_idf=False).fit(X_train_counts)
X_train_tf = tf_transformer.transform(X_train_counts)
X_train_tf.shape
```

```
(500, 4152)
```

```
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
X_train_tfidf.shape

#Transforming the test set as well
X_test_tfidf = tfidf_transformer.transform(X_test_counts)
X_test_tfidf.shape
```

Figure 2.8: TF-IDF

Bag of word doesn't capture the importance of the word it gives you the frequency of the word. TF-IDF resolves this matter through computation of two values. Tf is count of occurrences of the word in a document. IDF of the word across a set of documents. It tells us how common or rare a word is in the entire document set. The closer it is to 0, the more common is the word. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm. We then multiply these two values TF and IDF. We used sklearn tfidf-transformer which transforms a count matrix to a normalized tf or tf-idf representation as in Fig 2.8.

2.3.3 N-gram

```
[ ]  count_vect_ngram = CountVectorizer(ngram_range=(3,3))
     X_train_ngram = count_vect_ngram.fit_transform(X_train)
     X_train_ngram.shape
     X_test_ngram = count_vect_ngram.transform(X_test)
     X_test_ngram.shape
```

Figure 2.9: N-gram

Given a sequence of N-1 words, an N-gram model predicts the most probable word that might follow this sequence. It's a probabilistic model that's trained on a corpus of text. An N-gram model is built by counting how often word sequences occur in corpus text and then estimating the probabilities. Since a simple N-gram model has limitations, improvements are often made via smoothing, interpolation and backoff. We used sklearn countvectorizer giving it parameter for n-gram range which is min and max number of the word sequence as in Fig 2.9.

2.4 Classification

We used four classification techniques to classify the sentence to it's corresponding author using different transformation techniques,

- Naive Bayes
- KNN
- SVM
- Decision Tree

2.4.1 Naive Bayes

Naive bayes with TF-IDF

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(X_train_tfidf, y_train)
```

```
docs_new = ['And five couple enough make worth stand Five couple nothing one', 'Let Light see black deepe desires The']
X_new_counts = count_vect.transform(docs_new)
X_new_tfidf = tfidf_transformer.transform(X_new_counts)

predicted = clf.predict(X_new_tfidf)
# print(predicted)
for doc, category in zip(docs_new, predicted):
    print('%r => %s' % (doc, [category]))
```

```
'And five couple enough make worth stand Five couple nothing one' => ['chesterton-thursday']
'Let Light see black deepe desires The' => ['chesterton-thursday']
```

Figure 2.10: Naive bayes with TF-IDF

Using TF-IDF transformation technique on the training data and then feeding it to the model by using sklearn naive-bayes then we fit the model as in Fig 2.10.

Naive bayes with BOW

```
from sklearn.naive_bayes import MultinomialNB
clf2 = MultinomialNB().fit(X_train_counts, y_train)
docs_new = ['And five couple enough make worth stand Five couple nothing one', 'Let Light see black deepe desires The']
X_new_counts = count_vect.transform(docs_new)

predicted = clf2.predict(X_new_counts)
# print(predicted)
for doc, category in zip(docs_new, predicted):
    print('%r => %s' % (doc, [category]))
```

```
'And five couple enough make worth stand Five couple nothing one' => ['austen-emma']
'Let Light see black deepe desires The' => ['chesterton-thursday']
```

Figure 2.11: Naive bayes with BOW

Using combination between BOW transformation technique on the training data and naive bayes, then feeding it to the model by using sklearn naive-bayes then we fit the model as in Fig 2.11.

Naive bayes with N-gram

```
from sklearn.naive_bayes import MultinomialNB
clf_ngram_naivebayes = MultinomialNB().fit(X_train_ngram, y_train)
docs_new = ['And five couple enough make worth stand Five couple nothing one', 'Let Light see black deepe desires The']
X_new_counts =count_vect_ngram.transform(docs_new)
X_new_counts.shape

predicted = clf_ngram_naivebayes.predict(X_new_counts)
for doc, category in zip(docs_new, predicted):
        print('%r => %s' % (doc, [category]))
```

```
'And five couple enough make worth stand Five couple nothing one' => ['chesterton-thursday']
'Let Light see black deepe desires The' => ['shakespeare-macbeth']
```

Figure 2.12: Naive bayes with N-gram

Using combination between N-gram transformation technique on the training data and naive bayes, then feeding it to the model by using sklearn naive-bayes then we fit the model as in Fig 2.12.

2.4.2 KNN

KNN with BOW

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_counts, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
[23]
X_test_transformed = count_vect.transform(X_test)
knn_pred = knn.predict(X_test_transformed)
```

Figure 2.13: KNN with BOW

Using combination of BOW transformation technique on the training data KNN, and then giving data to the sklearn KNeighborsClassifier model and fitting the model as in Fig 2.13.

KNN with TF-IDF

```
knn.fit(X_train_tfidf, y_train)
knn_pred = knn.predict(X_test_transformed)
for doc, category in zip(docs_new, knn_pred):
    print('%r => %s' % (doc, [category]))
```

```
'And five couple enough make worth stand Five couple nothing one' => ['burgess-busterbrown']
'Let Light see black deepe desires The' => ['chesterton-thursday']
```

Figure 2.14: KNN with TF-IDF

Using combination of TF-IDF transformation technique on the training data and KNN ,then giving data to sklearn KNeighborsClassifier model and fitting the model as in Fig 2.14.

KNN with N-gram

```
X_test_ngram = count_vect_ngram.transform(X_test)
knn.fit(X_train_ngram, y_train)
knn_pred = knn.predict(X_test_ngram)
for doc, category in zip(docs_new, knn_pred):
    print('%r => %s' % (doc, [category]))
```

```
'And five couple enough make worth stand Five couple nothing one' => ['chesterton-thursday']
'Let Light see black deepe desires The' => ['chesterton-thursday']
```

Figure 2.15: KNN with N-gram

Using combination of N-gram transformation technique on the training data and KNN ,then giving data to sklearn KNeighborsClassifier model and fitting the model as in Fig 2.15.

2.4.3 SVM

SVM with TF-IDF

```
SVM_TFIDF=svm.fit(X_train_tfidf, y_train)
svm_pred = svm.predict(X_test_transformed)
for doc, category in zip(docs_new, svm_pred):
        print('%r => %s' % (doc, [category]))
```

```
'And five couple enough make worth stand Five couple nothing one' => ['burgess-busterbrown']
'Let Light see black deepe desires The' => ['chesterton-thursday']
```

Figure 2.16: SVM with TF-IDF

Using combination of TF-IDF transformation technique on the training data and SVM as in Fig 2.16.

SVM with BOW

```
from sklearn import svm

svm = svm.SVC(kernel='linear', C=1)
svm.fit(X_train_counts, y_train)
X_test_transformed = count_vect.transform(X_test)
svm_pred = svm.predict(X_test_transformed)
for doc, category in zip(docs_new, svm_pred):
        print('%r => %s' % (doc, [category]))
```

```
'And five couple enough make worth stand Five couple nothing one' => ['burgess-busterbrown']
'Let Light see black deepe desires The' => ['chesterton-thursday']
```

Figure 2.17: SVM with TF-IDF

Using combination of BOW transformation technique on the training data and SVM as in Fig 2.17.

SVM with N-gram

```
svm.fit(X_train_ngram, y_train)
svm_pred = svm.predict(X_test_ngram)
for doc, category in zip(docs_new, svm_pred):
        print('%r => %s' % (doc, [category]))
```

```
'And five couple enough make worth stand Five couple nothing one' => ['austen-emma']
'Let Light see black deepe desires The' => ['austen-emma']
```

Figure 2.18: SVM with N-gram

Using combination of N-gram transformation technique on the training data and SVM as in Fig 2.18.

2.4.4 Decision Tree

Decision Tree with BOW

```
from sklearn import tree
DT = tree.DecisionTreeClassifier()
DT.fit(X_train_counts, y_train)
X_test_transformed = count_vect.transform(X_test)
DT_pred = DT.predict(X_test_transformed)
for doc, category in zip(docs_new, knn_pred):
        print('%r => %s' % (doc, [category]))
```

```
'And five couple enough make worth stand Five couple nothing one' => ['chesterton-thursday']
'Let Light see black deepe desires The' => ['chesterton-thursday']
```

Figure 2.19: Decision Tree with BOW

Using combination of BOW transformation technique on the training data and Decision tree ,then giving data to sklearn DecisionTreeClassifier model and fitting the model as in Fig 2.19.

Decision Tree with TF-IDF

```
DT.fit(X_train_tfidf, y_train)
X_test_transformed = count_vect.transform(X_test)
DT_pred = DT.predict(X_test_transformed)
for doc, category in zip(docs_new, knn_pred):
        print('%r => %s' % (doc, [category]))
```

```
'And five couple enough make worth stand Five couple nothing one' => ['chesterton-thursday']
'Let Light see black deepe desires The' => ['chesterton-thursday']
```

Figure 2.20: Decision tree with TF-IDF

Using combination of TF-IDF transformation technique on the training data and Decision tree ,then giving data to sklearn DecisionTreeClassifier model and fitting the model as in Fig 2.20.

Decision Tree with N-gram

```
DT.fit(X_train_ngram, y_train)
DT_pred = DT.predict(X_test_ngram)
for doc, category in zip(docs_new, knn_pred):
        print('%r => %s' % (doc, [category]))
```

```
'And five couple enough make worth stand Five couple nothing one' => ['chesterton-thursday']
'Let Light see black deepe desires The' => ['chesterton-thursday']
```

Figure 2.21: Decision tree with N-gram

Using combination of N-gram transformation technique on the training data and Decision tree ,then giving data to sklearn DecisionTreeClassifier model and fitting the model as in Fig 2.21.

2.5 Evaluation

There are many ways to evaluate a model. We chose to evaluate it using a confusion matrix, a classification report, and the average accuracy of the cross validation. Some of the models produced very high accuracy up to 100%. Therefore we reduced the number words per sample from 100 to 30. This decreased the accuracy of the model to be able to compare between them. We also decreased the number of samples by half.

```
def performance(y,y_predict):
    print('Confusion Matrix\n')
    labels=['austen-emma', 'bryant-stories', 'burgess-busterbrown', 'chesterton-thursday', 'shakespeare-macbeth']
    hm=sn.heatmap(confusion_matrix(y,y_predict), annot=True)
    plt.show()
    print('Classification Report\n')
    clf_report = classification_report(y,y_predict,
                                       labels=labels,

                                       output_dict=True)
    sn.heatmap(pd.DataFrame(clf_report).iloc[:-1, :].T, annot=True)
```

Figure 2.22: Function that return CM and CR

First we implemented a function to return the confusion matrix and classification report as in Fig 2.22

2.5.1 Evaluation of Naive bayes

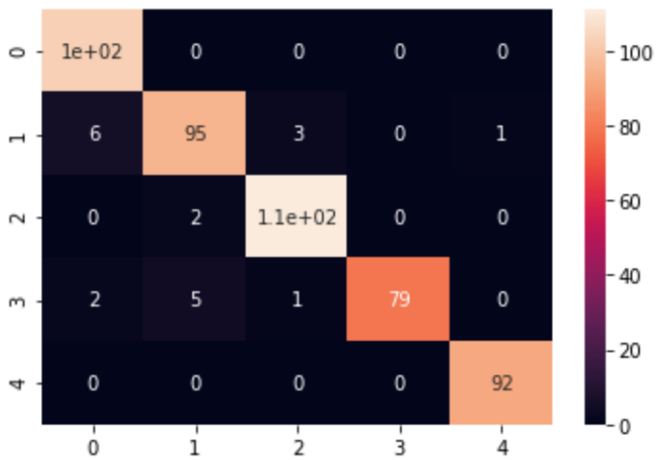
Given the Confusion matrix and classification report, it seems that TF-IDF and BOW obtained best results with naive bayes by accuracy 0.96 ,According to the figures 2.23, 2.24 and 2.25.

Naive bayes with TF-IDF

The following figure 2.23 represents the Confusion matrix and the classification report,

```
y_test_transformed_tfidf = clf2.predict(X_test_counts)
performance(y_test,y_test_transformed_tfidf)
```

```
Confusion Matrix
```




```
Classification Report
```

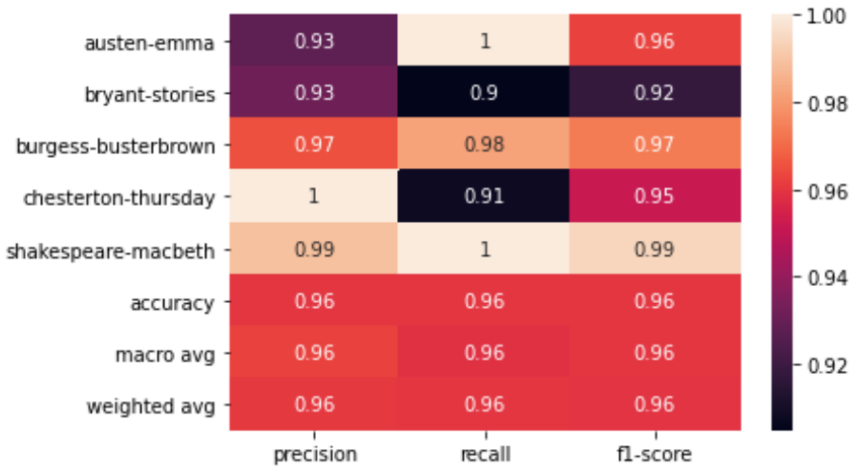


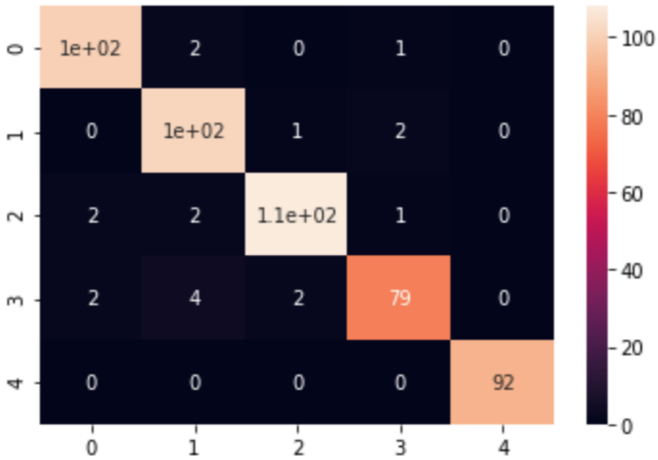
Figure 2.23: Naive bayes with TF-IDF

Naive bayes with BOW

The following figure 2.24 represents the Confusion matrix and the classification report,

```
y_test_transformed_bow = clf2.predict(X_test_counts)
performance(y_test,y_test_transformed_bow)
```

Confusion Matrix



Classification Report

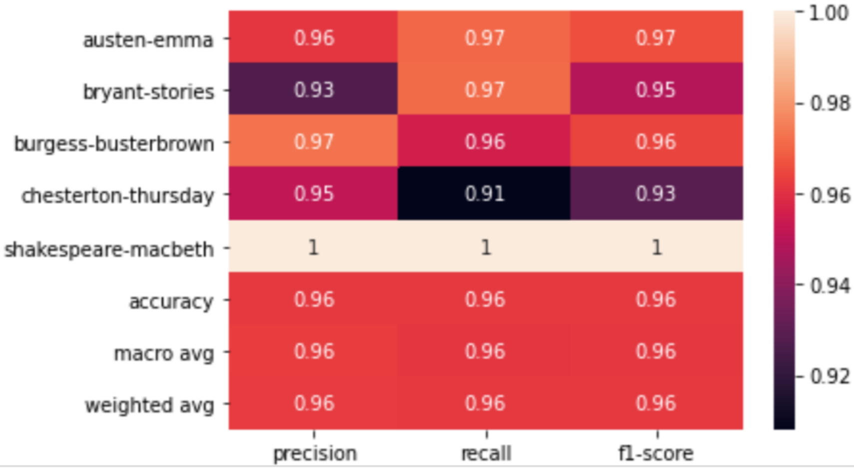


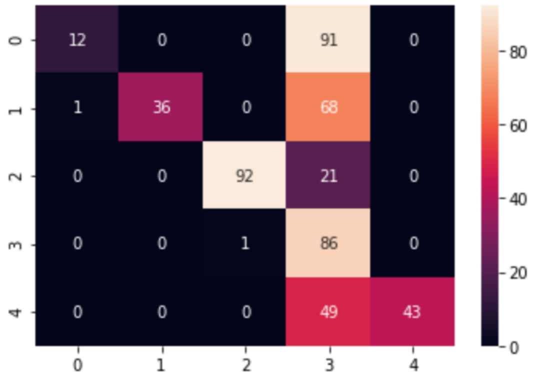
Figure 2.24: Naive bayes with BOW

Naive bayes with N-gram

The following figure 2.25 represents the Confusion matrix and the classification report,

```
y_test_transformed_ngram_bayes=clf_ngram_naivebayes.predict(X_test_ngram)
performance(y_test,y_test_transformed_ngram_bayes)
```

```
Confusion Matrix
```




```
Classification Report
```

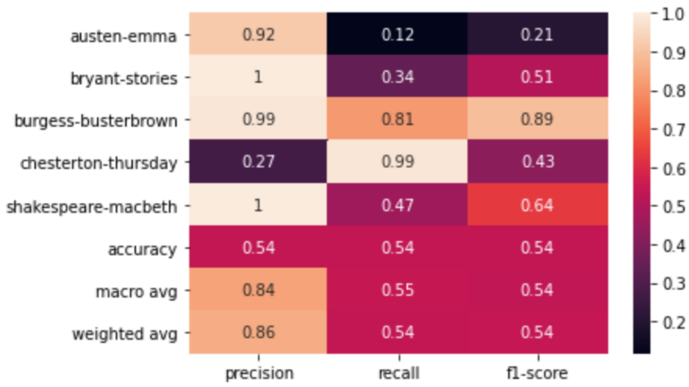


Figure 2.25: Naive bayes with N-gram

2.5.2 Evaluation of KNN

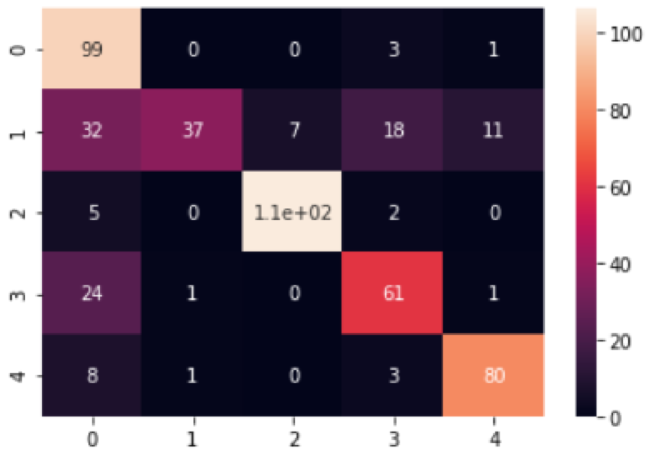
Given the Confusion matrices and classification report, it seems that BOW and TF-IDF obtained best results with KNN by accuracy 0.77 ,According to the figures 2.26, 2.28 and 2.27.

KNN with BOW

The following figure 2.26 represents the Confusion matrix and the classification report,

```
X_test_transformed = count_vect.transform(X_test)
knn_pred = knn.predict(X_test_transformed)
performance(y_test, knn_pred)
```

```
Confusion Matrix
```




```
Classification Report
```

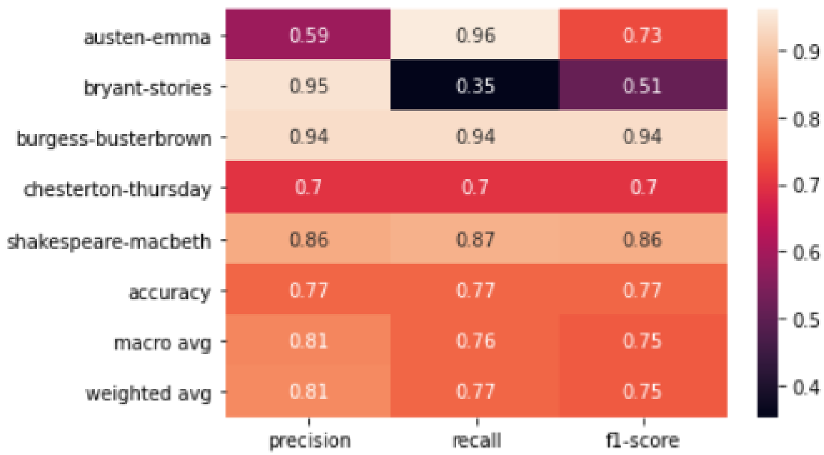


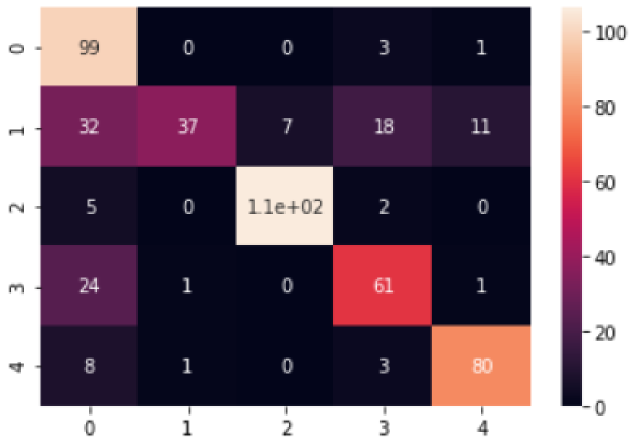
Figure 2.26: KNN with BOW Confusion Matrix

KNN with TF-IDF

The following figure 2.27 represents the Confusion matrix and the classification report,

```
knn_pred = knn.predict(X_test_transformed)
performance(y_test, knn_pred)
```

Confusion Matrix



Classification Report

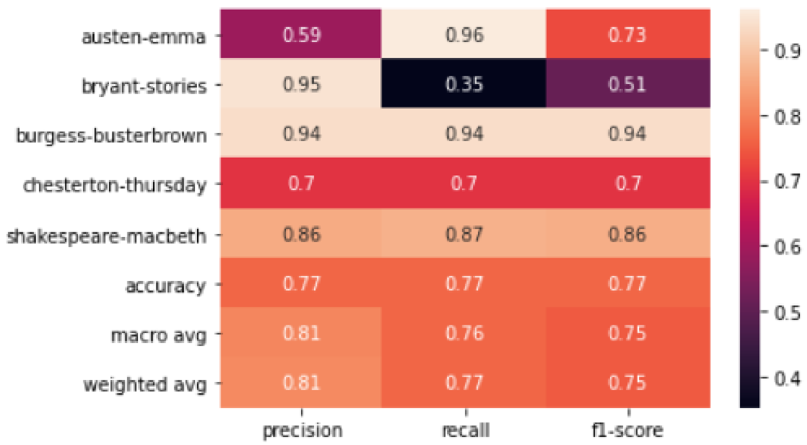


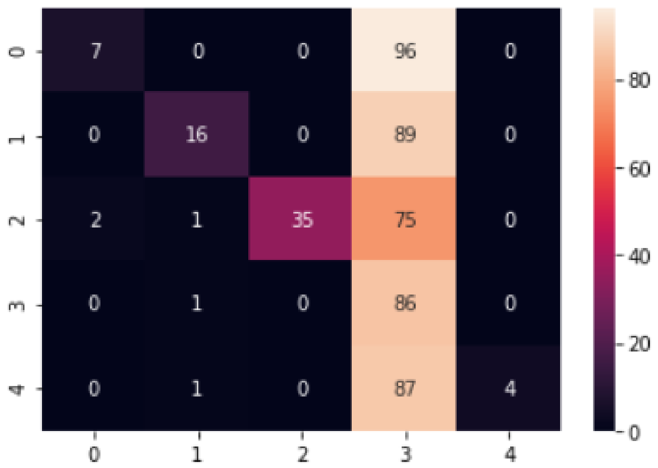
Figure 2.27: KNN with TF-IDF

KNN with N-gram

The following figure 2.28 represents the Confusion matrix and the classification report,

```
knn_pred = knn.predict(X_test_ngram)
performance(y_test, knn_pred)
```

```
Confusion Matrix
```




```
Classification Report
```

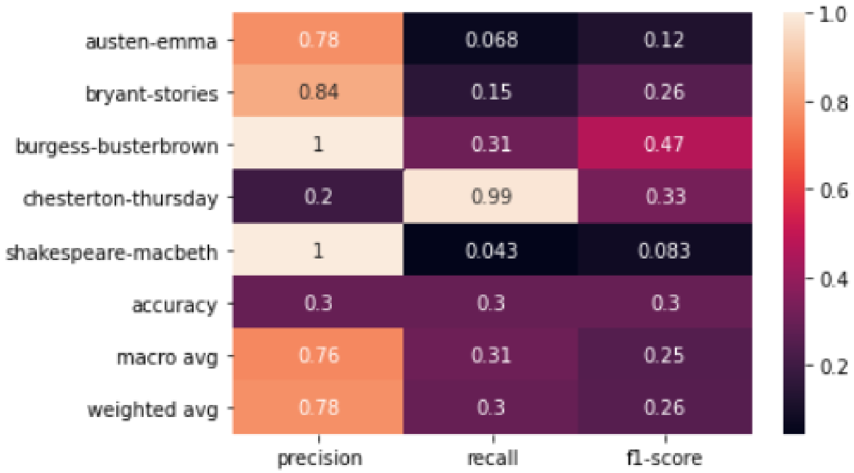


Figure 2.28: KNN with N-gram

.

2.5.3 Evaluation of SVM

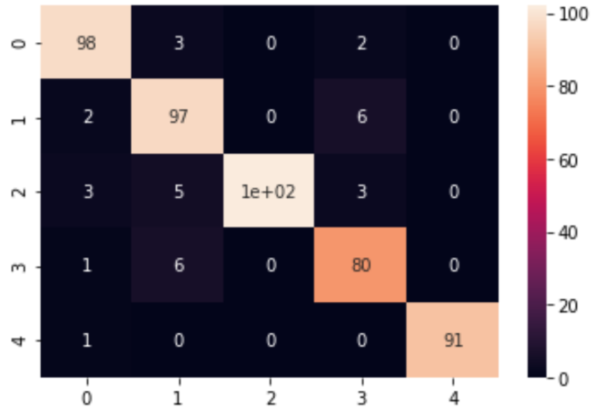
Given the Confusion matrices and classification report, it seems that TF-IDF and BOW obtained best results with SVM by accuracy 0.94 ,According to the figures 2.23, 2.24 and 2.25.

SVM with BOW

The following figure 2.29 represents the Confusion matrix and the classification report,

```
X_test_transformed = count_vect.transform(X_test)
svm_pred = svm.predict(X_test_transformed)
performance(y_test, svm_pred)
```

```
Confusion Matrix
```




```
Classification Report
```

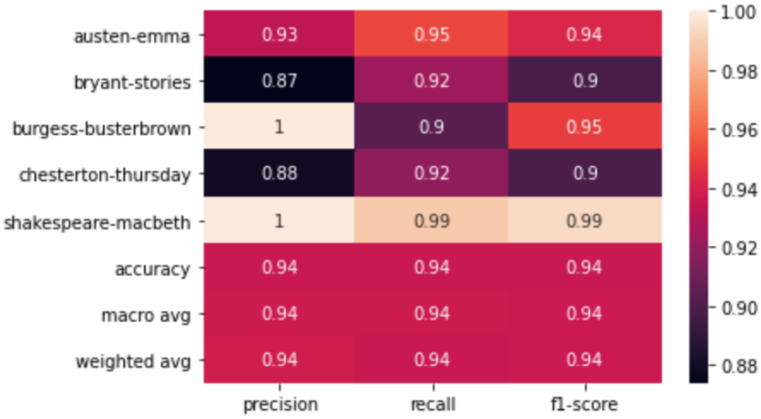


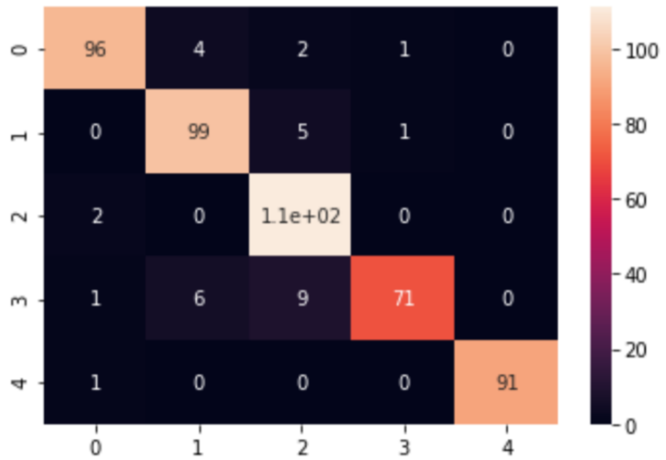
Figure 2.29: SVM with BOW Confusion Matrix

SVM with TF-IDF

The following figure 2.30 represents the Confusion matrix and the classification report,

```
SVM_TFIDF=svm.fit(X_train_tfidf, y_train)
svm_pred = svm.predict(X_test_transformed)
performance(y_test, svm_pred)
```

Confusion Matrix



Classification Report

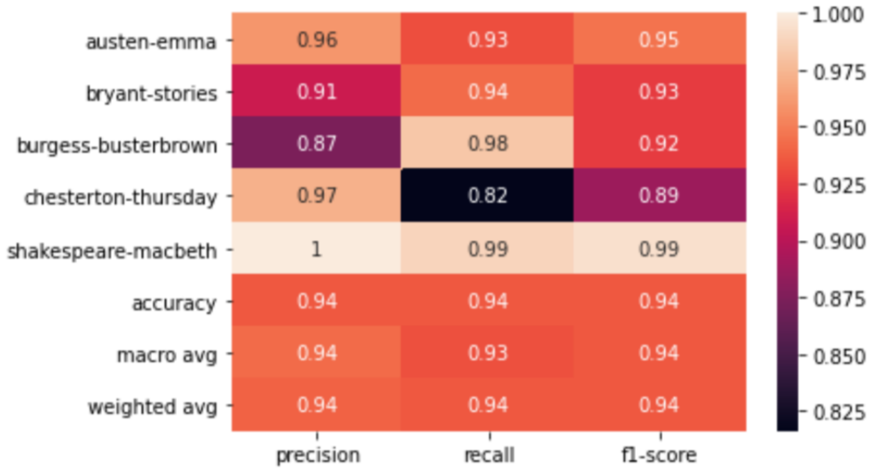


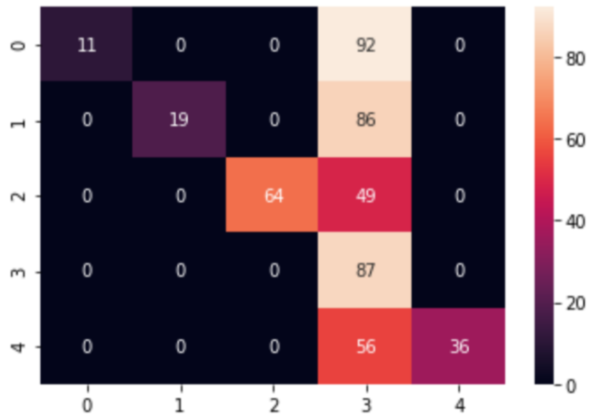
Figure 2.30: SVM with TF-IDF

SVM with N-gram

The following figure 2.31 represents the Confusion matrix and the classification report,

```
svm.fit(X_train_ngram, y_train)
svm_pred = svm.predict(X_test_ngram)
performance(y_test, svm_pred )
```

Confusion Matrix



Classification Report

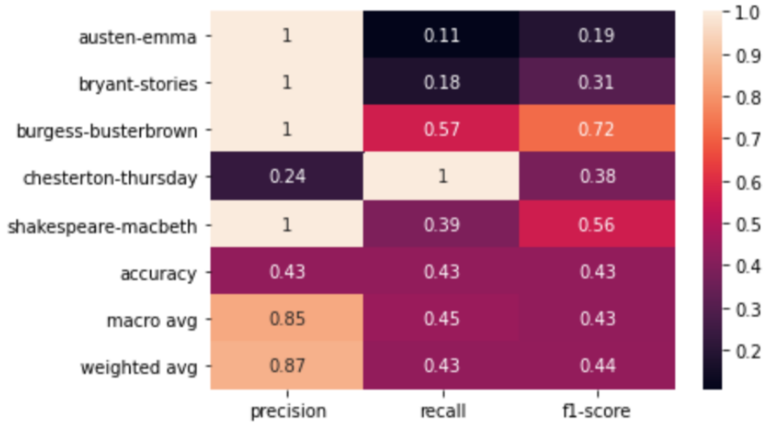


Figure 2.31: SVM with N-gram

.

2.5.4 Evaluation of Decision Tree

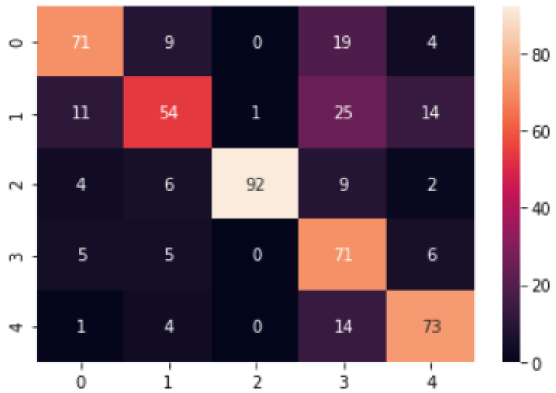
Given the Confusion matrices and classification report, it seems that TF-IDF obtained best results with Decision Tree by accuracy 0.73 ,According to the figures 2.32, 2.33 and 2.34.

Decision Tree with BOW

The following figure 2.32 represents the Confusion matrix and the classification report,

```
X_test_transformed = count_vect.transform(X_test)
DT_pred = DT.predict(X_test_transformed)
performance(y_test, DT_pred)
```

```
Confusion Matrix
```




```
Classification Report
```

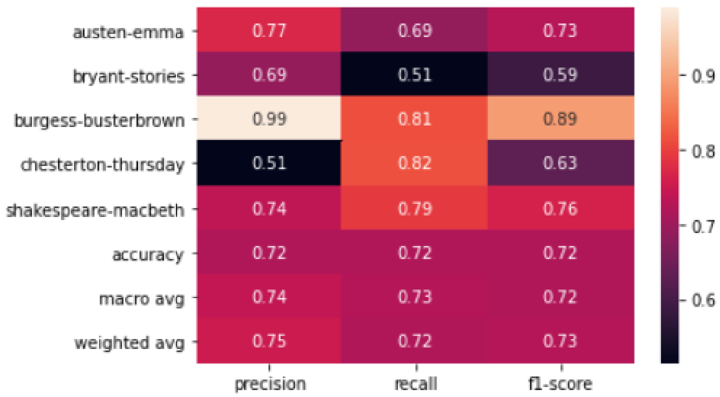


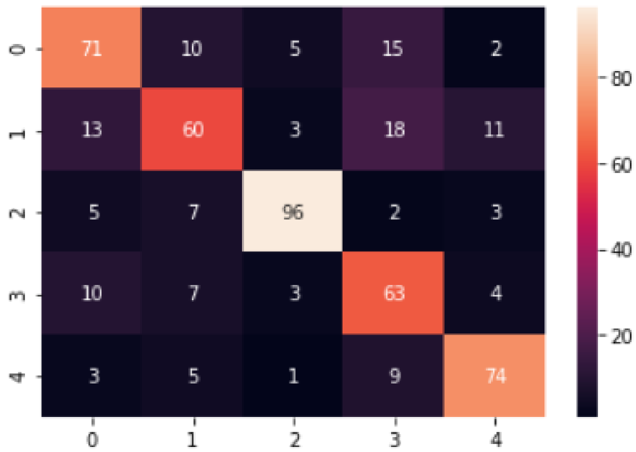
Figure 2.32: Decision Tree with BOW Confusion Matrix

Decision Tree with TF-IDF Confusion Matrix

The following figure 2.33 represents the Confusion matrix and the classification report,

```python
DT.fit(X_train_tfidf, y_train)
X_test_transformed = count_vect.transform(X_test)
DT_pred = DT.predict(X_test_transformed)
performance(y_test, DT_pred)
```

```
Confusion Matrix
```




```
Classification Report
```

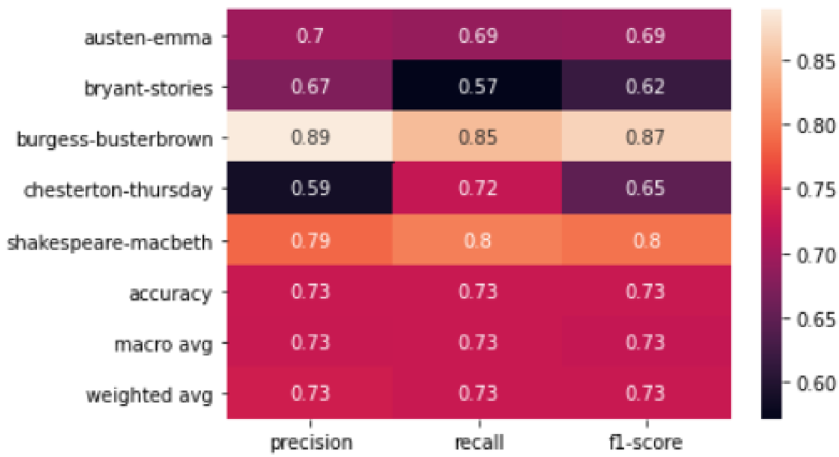


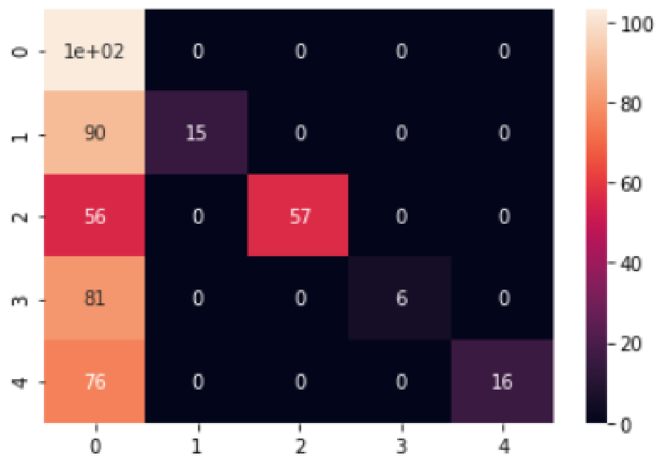
Figure 2.33: Decision Tree with TF-IDF

Decision Tree with N-gram Confusion Matrix

The following figure 2.34 represents the Confusion matrix and the classification report,

```
DT.fit(X_train_ngram, y_train)
DT_pred = DT.predict(X_test_ngram)
performance(y_test, DT_pred)
```

```
Confusion Matrix
```




```
Classification Report
```

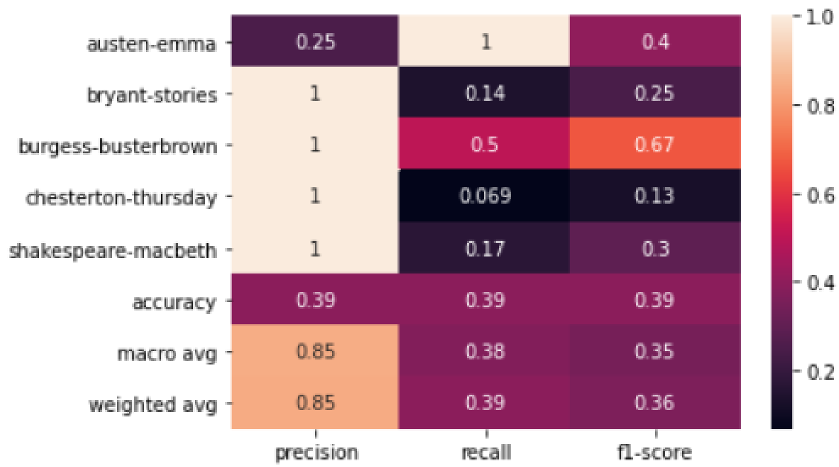


Figure 2.34: Decision Tree with N-gram

.

Evaluation by cross validation

Table 2.1: The average accuracy of the cross validation for all combinations.

	BOW	**TF-IDF**	**N-gram**
Naive Bayes	0.95	0.95	0.43
KNN	0.87	0.87	0.40
SVM	0.94	0.94	0.50
Decision Tree	0.77	0.77	0.42

Now we have for each combination between (model and transformation) the best accuracy with regards the confusion matrix and classification report. The TF-IDF transformation produced the best performance for each model. Comparing all the performances of the models with the TF-IDF transformation after cross validating, the naive bayes model outperformed them all with the highest accuracy of 0.95%.

2.5.5 Error Analysis

The wrong predicated labels

We gathered all the records from the testing set that where wrongly predicted and we but them all into a data frame along with the right label and the predicted label as in Fig 2.35

```
df_misclassified = get_misclassified(y_test_transformed_tfidf_nv)
df_misclassified
#performance(y_test,y_test_transformed_tfidf_nv)
```

	misclassified_records	misclassified_label	right_label
0	dewdrop slid blade grass tumbled parched beak ...	austen-emma	bryant-stories
1	saw colonel right entered plans vague servilit...	burgess-busterbrown	chesterton-thursday
2	dear could catch little jackal ran far fast li...	burgess-busterbrown	bryant-stories
3	truth time begun feel third kind fear piercing...	burgess-busterbrown	chesterton-thursday
4	green jacket red cap white owl feather along r...	burgess-busterbrown	bryant-stories
5	said second graceful rear right foot said thir...	chesterton-thursday	bryant-stories
6	imitates however small consolation remember si...	austen-emma	bryant-stories
7	ball rolled along swifter wind passed travelle...	chesterton-thursday	bryant-stories
8	speak speak little house always speak everythi...	burgess-busterbrown	bryant-stories
9	come last time goloshes thing bad especially u...	austen-emma	chesterton-thursday
10	forest trees taller trees world farther away s...	burgess-busterbrown	bryant-stories
11	prince courtiers statue unveiled elector excla...	chesterton-thursday	bryant-stories
12	wee folk good folk trooping together green jac...	burgess-busterbrown	bryant-stories
13	king royal robes wore royal crown hand king ri...	shakespeare-macbeth	bryant-stories
14	two daughters affectionate indulgent father co...	bryant-stories	austen-emma
15	spoke thin clear voice live high air said man ...	chesterton-thursday	bryant-stories
16	came rose resilient rapidity professor spoken ...	bryant-stories	chesterton-thursday
17	even see fled along straight open doors big ba...	burgess-busterbrown	bryant-stories
18	time quaint red houses could think oddly shape...	austen-emma	chesterton-thursday
19	answered thou comest sword spear shield come t...	shakespeare-macbeth	bryant-stories
20	disdain impudent fellow merely apes imitates h...	austen-emma	bryant-stories

Figure 2.35: Wrong prediction data frame.

```
def get_misclassified(pred):
# Correcting the index of x,y_test to get the misclassified records
  y_test_right_index = []
  X_test_right_index =[]
  for item in y_test:
    y_test_right_index.append(item)


  for item1 in X_test:
    X_test_right_index.append(item1)
  # print(X_test_right_index)
#creating an array of the mis-classified records
  misclassified_records=[]
  misclassified_label = []
  right_label= []
  for i in range(len(pred)):
    if (pred[i]!= y_test_right_index[i]):
      misclassified_records.append(X_test_right_index[i])
      misclassified_label.append(pred[i])
      right_label.append(y_test_right_index[i])
  df=pd.DataFrame({"misclassified_records":misclassified_records,"misclassified_label":misclassified_label,"right_label":right_label })
  return df
```

Figure 2.36: Function that generate the wrong prediction.

Gathering all Records of the Same Label

After we generated wrong predication data frame we gathered all records of the same right label to be able to identify the words that confuse the model and make it choose another label as in Fig 2.37

	misclassified_records	misclassified_label	right_label
0	dewdrop slid blade grass tumbled parched beak ...	austen-emma	bryant-stories
2	dear could catch little jackal ran far fast li...	burgess-busterbrown	bryant-stories
4	green jacket red cap white owl feather along r...	burgess-busterbrown	bryant-stories
5	said second graceful rear right foot said thir...	chesterton-thursday	bryant-stories
6	imitates however small consolation remember si...	austen-emma	bryant-stories
7	ball rolled along swifter wind passed travelle...	chesterton-thursday	bryant-stories
8	speak speak little house always speak everythi...	burgess-busterbrown	bryant-stories
10	forest trees taller trees world farther away s...	burgess-busterbrown	bryant-stories
11	prince courtiers statue unveiled elector excla...	chesterton-thursday	bryant-stories
12	wee folk good folk trooping together green jac...	burgess-busterbrown	bryant-stories
13	king royal robes wore royal crown hand king ri...	shakespeare-macbeth	bryant-stories
15	spoke thin clear voice live high air said man ...	chesterton-thursday	bryant-stories
17	even see fled along straight open doors big ba...	burgess-busterbrown	bryant-stories
19	answered thou comest sword spear shield come t...	shakespeare-macbeth	bryant-stories
20	disdain impudent fellow merely apes imitates h...	austen-emma	bryant-stories

Figure 2.37: Record of the same label that where miss-predicted.

visualization of Error Analysis

We plotted the most frequent words in the miss-predicted records for each label so that we can get an insight of the words that confuse the model as in Fig 2.38 and 2.39.

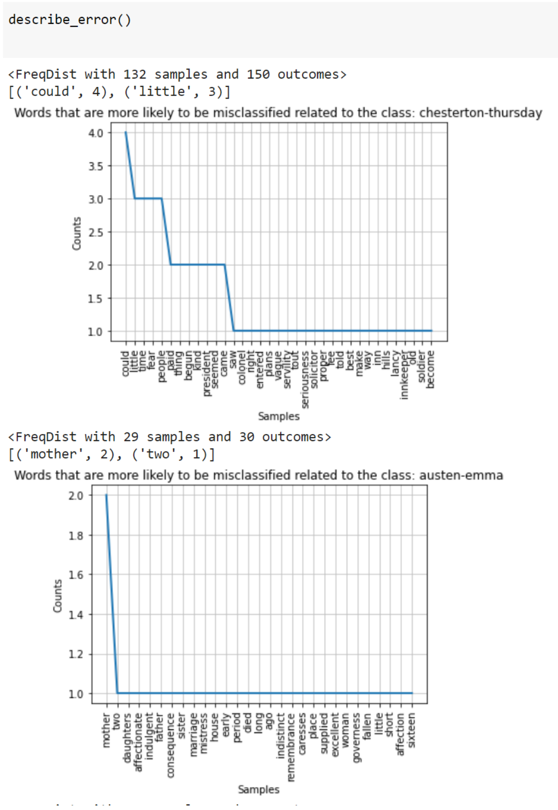


Figure 2.38: Most frequent words in the miss-predicted records.

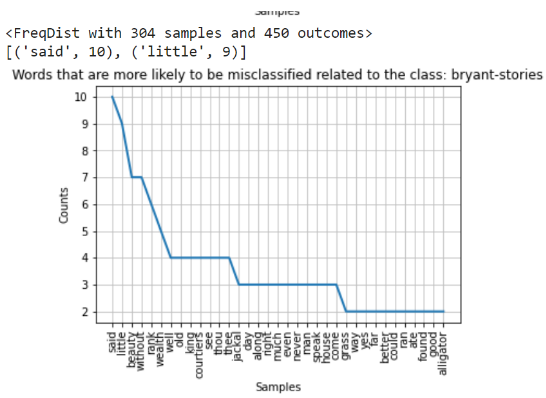


Figure 2.39: most frequent words in the mis-predicted records

Our target in this analysis is to capture the words for each label (author and book) that the model misclassified. First the records were gathered that were not predicted correctly and after this we plotted the frequency of the words in these records. These words are similarly used by two or more authors and have close frequency; hence the model wrongly predicts them.

2.5.6 Decreasing the Accuracy by 20% for the Champion Model

Decreasing the number of chunks to 150 and the number of words per chunk to 20, it's observed that the accuracy of the model decreased. The accuracy also decreased by changing the train test data split. When changing the split to 30% training and 70% testing.

```
for c in range(150):
    while True: # To stay in loop and change random number if it was used before
        x = randrange(0,len(filtered_words)-20) #generate random number from 0 to length of string - 100 (to be able to take last 100 words)
        if (x not in random_numbers): # make sure the number was not used before
            random_numbers.append(x) #add random number to list
            filtered_sentences.append(' '.join(filtered_words[x:x+20])) #add 100 words from random position to array.
            labels.append(texts[i][0:-4]) #append the label of the book while removing last 4 characters ".txt"
            break
```

Figure 2.40: Changing get-df to get 150 sentence for each author and 20 word per sentence

```
X_train, X_test, y_train, y_test = train_test_split(sample_x, label_x, test_size=0.7, random_state=42)
```

Figure 2.41: Splitting Data 30% training and 70% testing

```
from sklearn.model_selection import cross_val_score #cross validation for Naive bayes using TF-IDF
scores = cross_val_score(clf, X_train_tfidf, y_train, cv=10)
scores
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

Accuracy: 0.75 (+/- 0.08)
```

Figure 2.42: Showing Model's accuracy dropped from 0.95 to 0.75

2.5.7 Thresholds

There are three factors that affect the performance of the models. The three factors are the number of samples, the number of words per sample, and train test split. The more we decrease each of the number of samples and words and the training data split. The threshold for the number of samples is 150 per book (750), for the number of words per sample the threshold is 10 per sample, and finally the threshold for the training and testing data split is 30% for the training data.

2.5.8 Bias and Variability

In the first scenario we set the number of words per each chunk to 30 we got the bias and variance shown in Fig 2.43. In the second scenario we set the number of words per chunk 100 and we got bias and variance shown in Fig 2.44.It turned out that the more words we feed into the model the bias increases and the variance decreases [2].

```
y_train_mapped = numpy.array(y_train_fact)
y_test_mapped = numpy.array(y_test_fact)
mse, bias, var = bias_variance_decomp(clf, X_new_tfidf, y_train_mapped, X_test_tfidf, y_test_mapped, loss='mse', num_rounds=100, random_seed=1)
# summarize results
print('MSE: %.3f' % mse)
print('Bias: %.3f' % bias)
print('Variance: %.3f' % var)
```

```
MSE: 4.579
Bias: 2.792
Variance: 1.787
```

Figure 2.43: Showing Model's accuracy dropped from 0.95 to 0.75

```
y_train_mapped = numpy.array(y_train_fact)
y_test_mapped = numpy.array(y_test_fact)
mse, bias, var = bias_variance_decomp(clf, X_new_tfidf, y_train_mapped, X_test_tfidf, y_test_mapped, loss='mse', num_rounds=100, random_seed=1)
# summarize results
print('MSE: %.3f' % mse)
print('Bias: %.3f' % bias)
print('Variance: %.3f' % var)
```

```
MSE: 4.744
Bias: 2.950
Variance: 1.794
```

Figure 2.44: Showing Model's accuracy dropped from 0.95 to 0.75

2.6 Future Work

In the future, we can use different models like a deep learning model. Other than the model we can try different transformation methods such as LDA and word-embedding. We can use different combinations of books and increase the number of partitions.

Chapter 3

Conclusions

In conclusion, the naive bayes model had the best performance with the SVM coming close. However, the more we increased in the training data the better all the models became with very high accuracy in most of them greater than 90%. The TF-IDF transformation turned out to be the best transformation for this kind of problem which is in text classification. In the error analysis we concluded that the some samples were wrongly predicted because the words in the samples were also frequently found in other books.

References

[1] Y. Ma, Y. Li, X. Wu, and X. Zhang, "Chinese text classification review," in *2018 9th International Conference on Information Technology in Medicine and Education (ITME)*, IEEE, 2018, pp. 737–739.

[2] T. G. Dietterich and E. B. Kong, "Machine learning bias, statistical bias, and statistical variance of decision tree algorithms," Citeseer, Tech. Rep., 1995.