# CLASSIFICATION

Assignment 1

uOttawa

Faculté de génie
Faculty of Engineering

**Ahmed Hallaba** **Bassel Hamshary**
**Kamel El-Sehly**

MAY 30, 2021
UNIVERSITY OF OTTAWA
Ottawa, Canada

## Table of Contents

## Table of Figures

## 1. Objectives

The overall objective is to produce classification predictions and compare them, analyze the pros and cons of algorithms, and generate and communicate the insights.

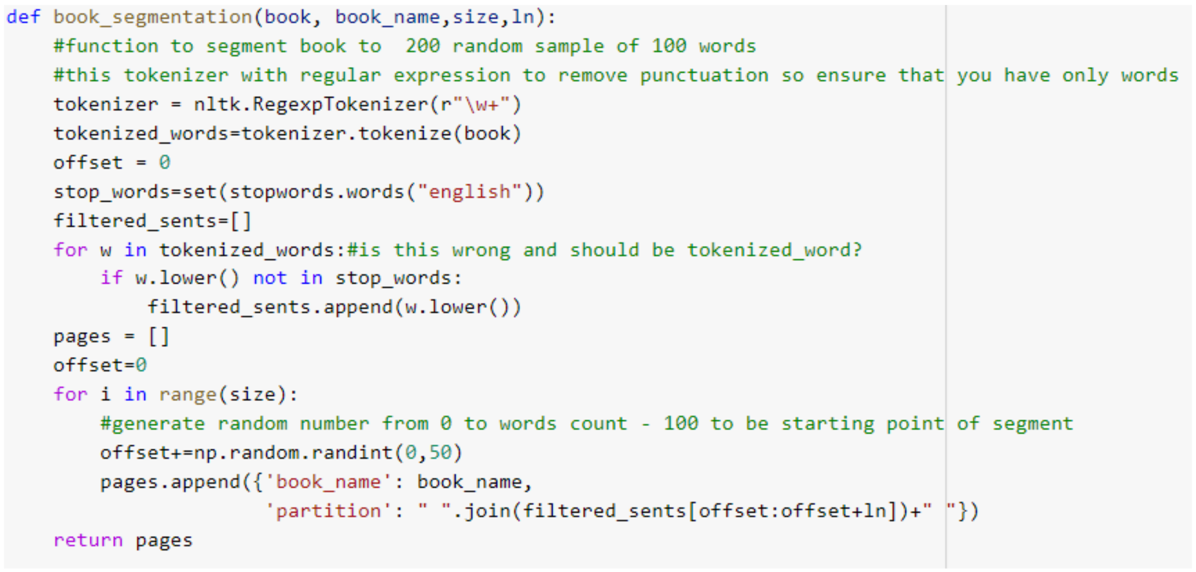
## 2. Requirements

Take five different samples of Gutenberg digital books, which are of five different authors, that you suspect are of the same genres and are semantically the same.
Separate and set aside unbiased random partitions for training, validation, and testing. Gauge the bias and variability of the models to decide the champion model. Then play with the features and other factors that provide you with leverages to make it harder for the model to predict and bring the accuracy down for about 20% and then check the bias and variability.

## 3. Procedures

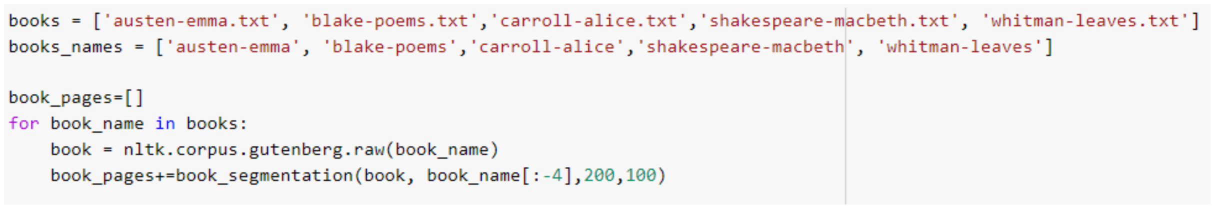
### 3.1. Data Preparation

Create random samples of 200 documents of each book, representative of the source input, and prepare records of 100 words for each document, label them as per the book they belong to.



```python
def book_segmentation(book, book_name,size,ln):
    #function to segment book to  200 random sample of 100 words
    #this tokenizer with regular expression to remove punctuation so ensure that you have only words
    tokenizer = nltk.RegexpTokenizer(r"\w+")
    tokenized_words=tokenizer.tokenize(book)
    offset = 0
    stop_words=set(stopwords.words("english"))
    filtered_sents=[]
    for w in tokenized_words:#is this wrong and should be tokenized_word?
        if w.lower() not in stop_words:
            filtered_sents.append(w.lower())
    pages = []
    offset=0
    for i in range(size):
        #generate random number from 0 to words count - 100 to be starting point of segment
        offset+=np.random.randint(0,50)
        pages.append({'book_name': book_name,
                      'partition': " ".join(filtered_sents[offset:offset+ln])+" "})
    return pages
```

Figure 1: Book segmentation function: clean, tokenize, and generate random samples



```python
books = ['austen-emma.txt', 'blake-poems.txt','carroll-alice.txt','shakespeare-macbeth.txt', 'whitman-leaves.txt']
books_names = ['austen-emma', 'blake-poems','carroll-alice','shakespeare-macbeth', 'whitman-leaves']

book_pages=[]
for book_name in books:
    book = nltk.corpus.gutenberg.raw(book_name)
    book_pages+=book_segmentation(book, book_name[:-4],200,100)
```

Figure 2: Reading books and generate 200 samples of 100 word

## 3.2. Data Preprocessing

Remove stop-words and punctuation characters. Stop words are the words that do not affect the meaning of the sentence. In this stage, the data is being cleaned and ready to train the model.

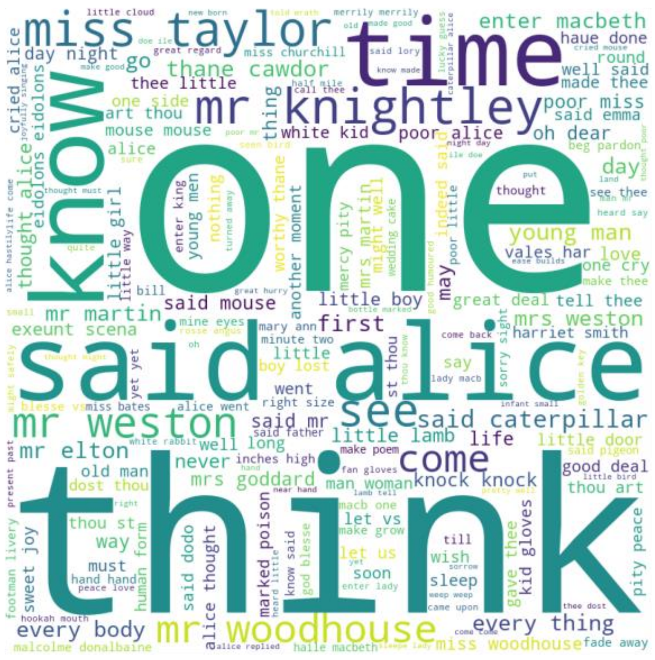
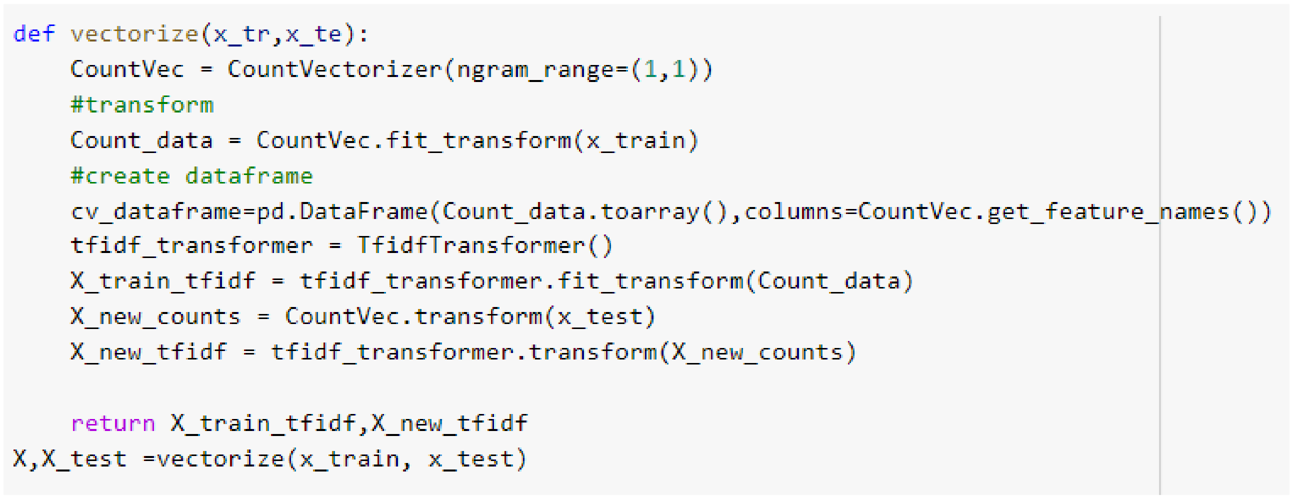


Figure 3: Stop words

## 3.3. Data Transformation

Data transformation is an important step before training the model, as we transform the data in a way we want the model to be trained on. In our case, the data was transformed to a set of words, each word with its specific weight according to its frequency of occurrence.

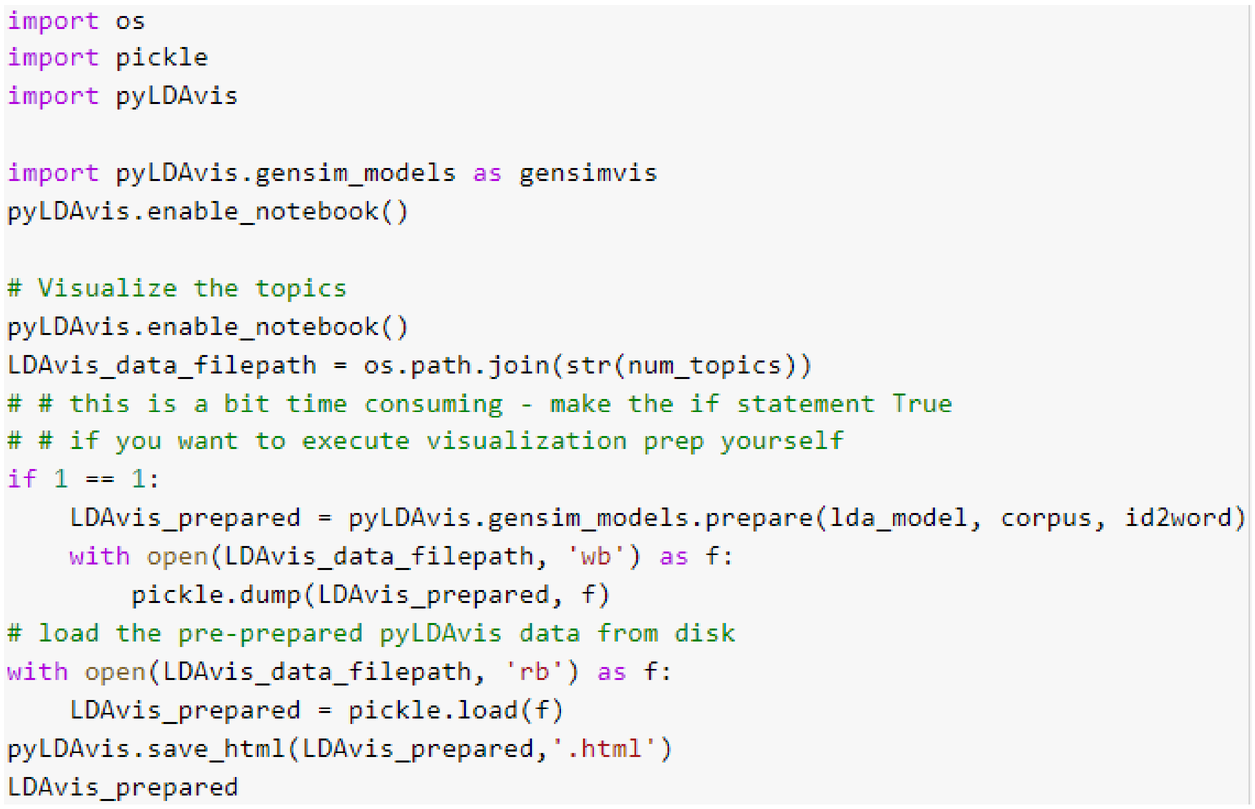


```python
def vectorize(x_tr,x_te):
    CountVec = CountVectorizer(ngram_range=(1,1))
    #transform
    Count_data = CountVec.fit_transform(x_train)
    #create dataframe
    cv_dataframe=pd.DataFrame(Count_data.toarray(),columns=CountVec.get_feature_names())
    tfidf_transformer = TfidfTransformer()
    X_train_tfidf = tfidf_transformer.fit_transform(Count_data)
    X_new_counts = CountVec.transform(x_test)
    X_new_tfidf = tfidf_transformer.transform(X_new_counts)

    return X_train_tfidf,X_new_tfidf
X,X_test =vectorize(x_train, x_test)
```

Figure 4: Vectorize function: apply BOW and TF-IDF transformation on the dataset

In this function we took the training dataset, collect all the unique words, and get the frequency of occurrence of each word using BOW and countVectorizer respectively. After that, each word was assigned a weight using TF-IDF based on the frequency of occurrence of each word, high occurrence rate is assigned to low weights and vice-versa.



```python
import os
import pickle
import pyLDAvis

import pyLDAvis.gensim_models as gensimvis
pyLDAvis.enable_notebook()

# Visualize the topics
pyLDAvis.enable_notebook()
LDAvis_data_filepath = os.path.join(str(num_topics))
# # this is a bit time consuming - make the if statement True
# # if you want to execute visualization prep yourself
if 1 == 1:
    LDAvis_prepared = pyLDAvis.gensim_models.prepare(lda_model, corpus, id2word)
    with open(LDAvis_data_filepath, 'wb') as f:
        pickle.dump(LDAvis_prepared, f)
# load the pre-prepared pyLDAvis data from disk
with open(LDAvis_data_filepath, 'rb') as f:
    LDAvis_prepared = pickle.load(f)
pyLDAvis.save_html(LDAvis_prepared,'.html')
LDAvis_prepared
```

Figure 5: LDA

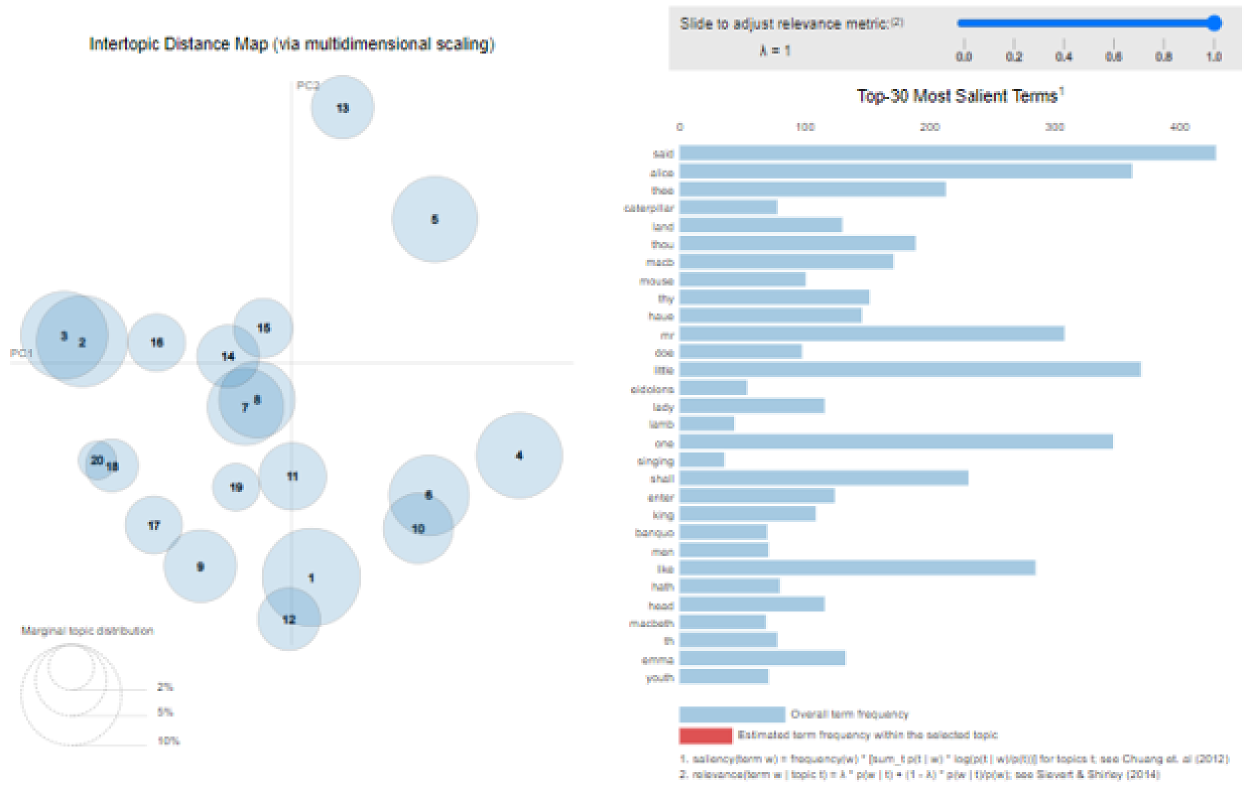
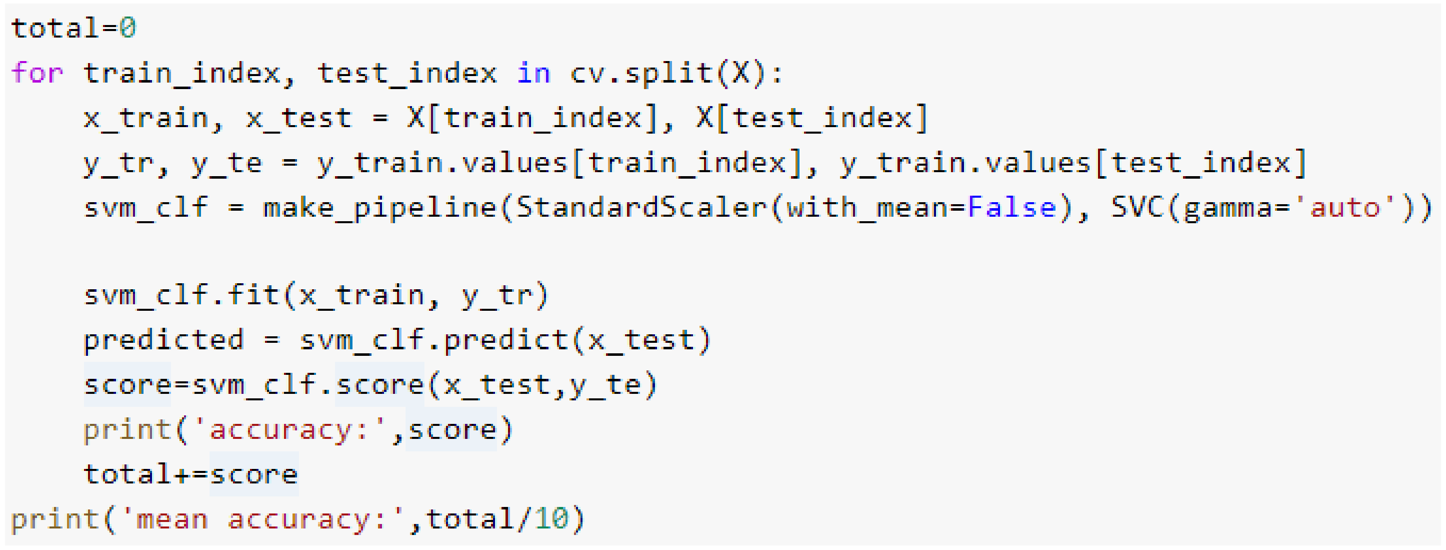


Figure 6: LDA Visualization

## 3.4. Training & Validation

After preparing and transforming of the data, now the data is ready to be trained. The training models aim to find the name of the book from an input sentence to it. The data is trained by many algorithms and validated through 10 folds cross validation technique to get the best accuracy, according to the accuracy performance of each algorithm we will be able to choose the champion model for this task.

### 3.4.1. Support Vector Machine Model

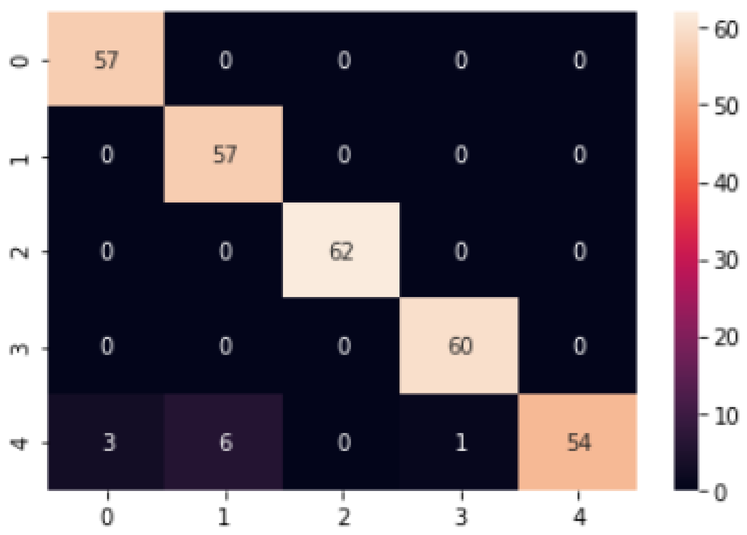


```
total=0
for train_index, test_index in cv.split(X):
    x_train, x_test = X[train_index], X[test_index]
    y_tr, y_te = y_train.values[train_index], y_train.values[test_index]
    svm_clf = make_pipeline(StandardScaler(with_mean=False), SVC(gamma='auto'))

    svm_clf.fit(x_train, y_tr)
    predicted = svm_clf.predict(x_test)
    score=svm_clf.score(x_test,y_te)
    print('accuracy:',score)
    total+=score
print('mean accuracy:',total/10)
```

Figure 7: SVM Model

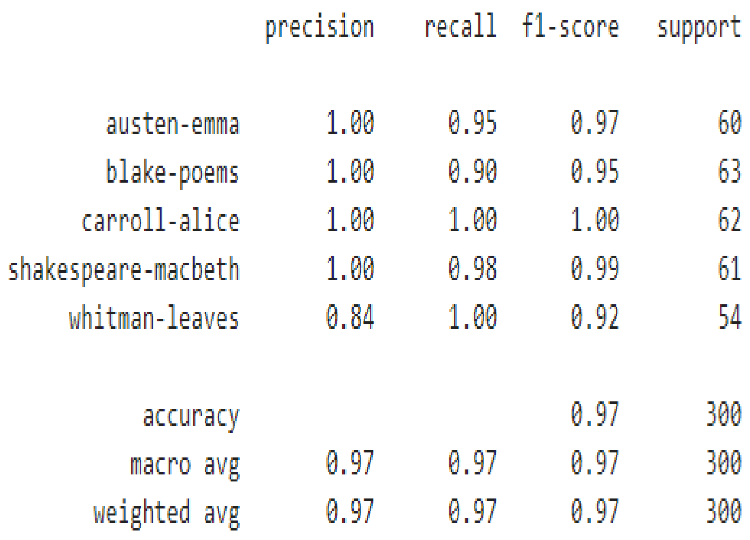
```
Confusion matrix
```



| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 57 | 0 | 0 | 0 | 0 |
| 1 | 0 | 57 | 0 | 0 | 0 |
| 2 | 0 | 0 | 62 | 0 | 0 |
| 3 | 0 | 0 | 0 | 60 | 0 |
| 4 | 3 | 6 | 0 | 1 | 54 |

```
Score 96.66666666666667 %
```

Figure 8: SVM Confusion Matrix



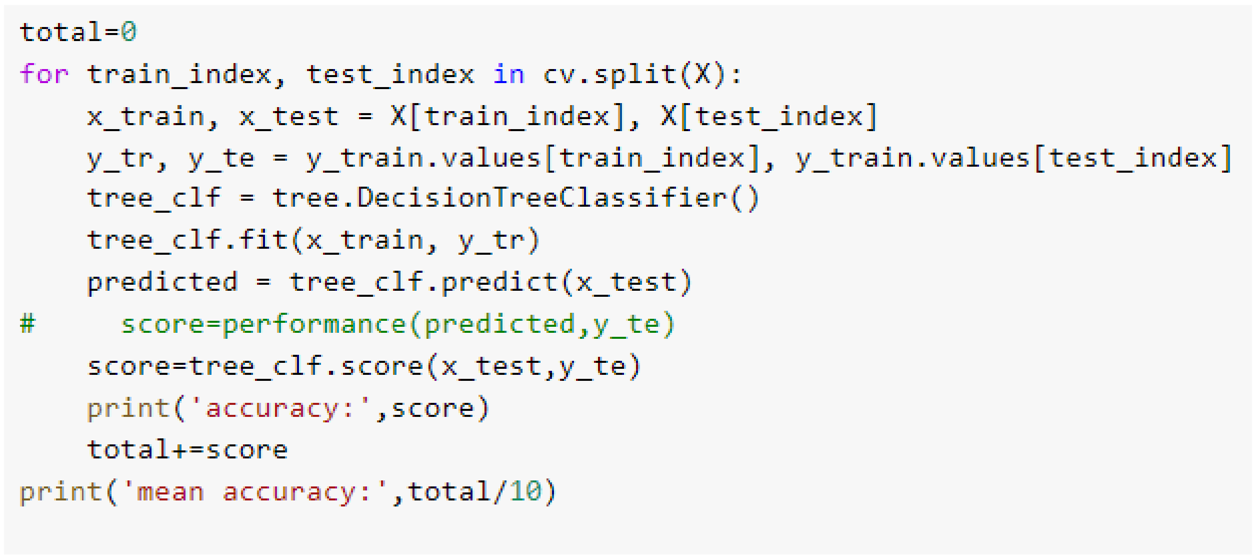
```
                     precision    recall  f1-score   support

         austen-emma      1.00      0.95      0.97        60
         blake-poems      1.00      0.90      0.95        63
       carroll-alice      1.00      1.00      1.00        62
shakespeare-macbeth       1.00      0.98      0.99        61
      whitman-leaves      0.84      1.00      0.92        54

            accuracy                          0.97       300
           macro avg      0.97      0.97      0.97       300
        weighted avg      0.97      0.97      0.97       300
```

Figure 9: SVM Performance Metrics

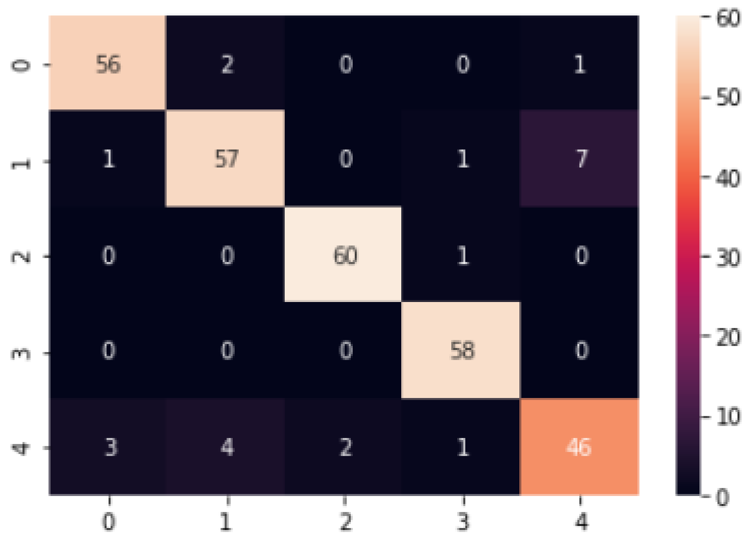
### 3.4.2. Decision Tree Model



```
total=0
for train_index, test_index in cv.split(X):
    x_train, x_test = X[train_index], X[test_index]
    y_tr, y_te = y_train.values[train_index], y_train.values[test_index]
    tree_clf = tree.DecisionTreeClassifier()
    tree_clf.fit(x_train, y_tr)
    predicted = tree_clf.predict(x_test)
#     score=performance(predicted,y_te)
    score=tree_clf.score(x_test,y_te)
    print('accuracy:',score)
    total+=score
print('mean accuracy:',total/10)
```

Figure 10: Decision Trees Model

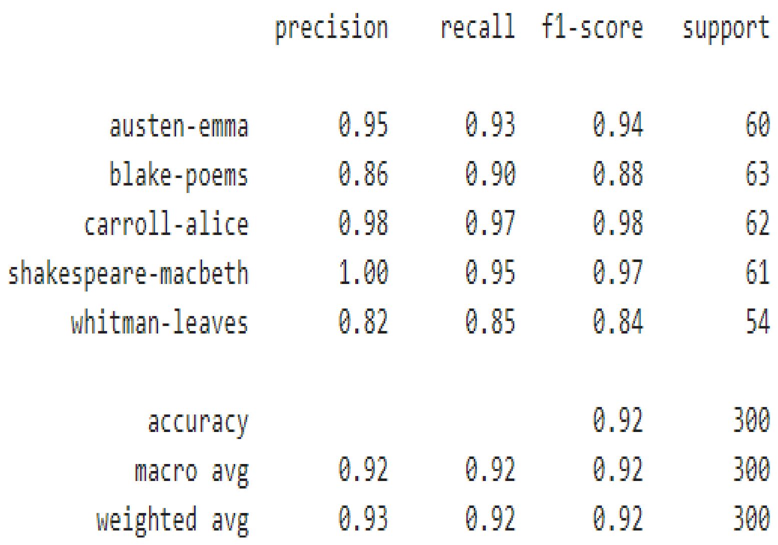
```
Confusion matrix
```



|  | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 56 | 2 | 0 | 0 | 1 |
| 1 | 1 | 57 | 0 | 1 | 7 |
| 2 | 0 | 0 | 60 | 1 | 0 |
| 3 | 0 | 0 | 0 | 58 | 0 |
| 4 | 3 | 4 | 2 | 1 | 46 |

```
Score 92.33333333333333 %
```

Figure 11: Decision Tree Confusion Matrix



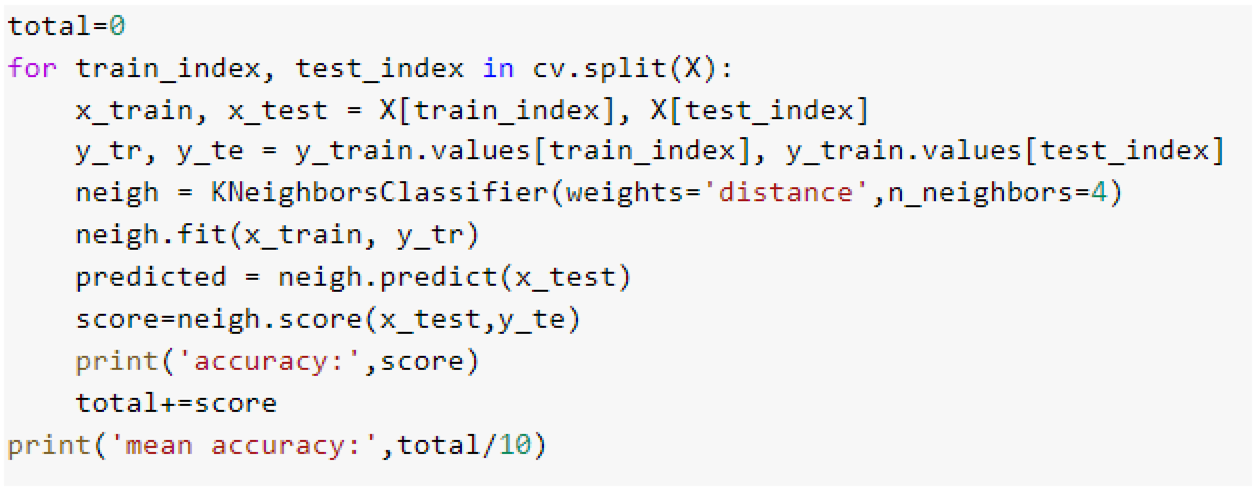
```
                    precision    recall  f1-score   support

        austen-emma      0.95      0.93      0.94        60
        blake-poems      0.86      0.90      0.88        63
       carroll-alice      0.98      0.97      0.98        62
shakespeare-macbeth      1.00      0.95      0.97        61
     whitman-leaves      0.82      0.85      0.84        54

           accuracy                          0.92       300
          macro avg      0.92      0.92      0.92       300
       weighted avg      0.93      0.92      0.92       300
```

Figure 12: Decision Tree Performance Metrics

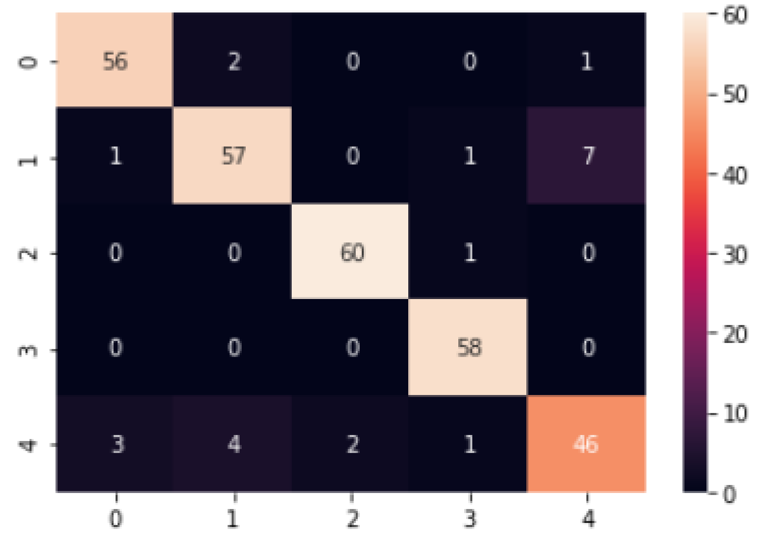
### 3.4.3. K-Nearest Neighbour Model



```
total=0
for train_index, test_index in cv.split(X):
    x_train, x_test = X[train_index], X[test_index]
    y_tr, y_te = y_train.values[train_index], y_train.values[test_index]
    neigh = KNeighborsClassifier(weights='distance',n_neighbors=4)
    neigh.fit(x_train, y_tr)
    predicted = neigh.predict(x_test)
    score=neigh.score(x_test,y_te)
    print('accuracy:',score)
    total+=score
print('mean accuracy:',total/10)
```

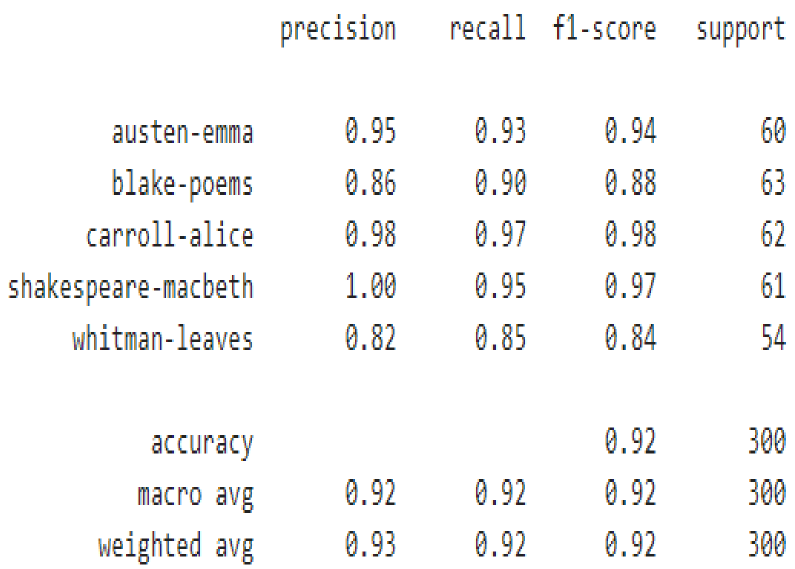
Figure 13: K-Nearest Neighbour Classifier Model

```
Confusion matrix
```



```
Score 92.33333333333333 %
```

Figure 14: K-Nearest Neighbour Confusion Matrix



```
                    precision    recall  f1-score   support

      austen-emma        0.95      0.93      0.94        60
      blake-poems        0.86      0.90      0.88        63
    carroll-alice        0.98      0.97      0.98        62
shakespeare-macbeth      1.00      0.95      0.97        61
   whitman-leaves        0.82      0.85      0.84        54

         accuracy                            0.92       300
        macro avg        0.92      0.92      0.92       300
     weighted avg        0.93      0.92      0.92       300
```

Figure 15: K-Nearest Neighbour Performance Metrics

### 3.5. Hyperparameter Tuning

#### 3.5.1. SVM

To tune the SVM model, Gamma is the key parameter for increasing or decreasing of accuracy, where the gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'.[1]

| | |
|---|---|
| • Default parameters give mean accuracy | 99 % |
| • In SVM using gamma='auto' mean accuracy | 95 % |
| • Using kernel='sigmoid' mean accuracy | 96 % |

#### 3.5.2. Decision Tree

In the Decision Tree algorithm, the depth of the tree is the key parameter to tune the accuracy. As depth decreases, the accuracy decreases.

| | |
|---|---|
| • Default parameters mean accuracy | 95 % |
| • max_depth = 1 gives | 35 % |
| • max_depth = 5 gives | 82 % |
| • max_depth = 10 gives | 90 % |

#### 3.5.3. K-Nearest Neighbour Model

In the K-Nearest Neighbour, the number of classes (neighbours) and the separated distances between them are the parameters that can tune the accuracy performance. As the distance between classes increases the accuracy increases.

| | |
|---|---|
| • Default parameters gives mean accuracy | 61 % |
| • n_neighbors = 1 gives mean accuracy | 96 % |
| • n_neighbors = 3 gives mean accuracy | 85 % |
| • weights = 'distance' increases accuracy to | 71 % |

## 4. Conclusion

To sum up, the target of the task is to predict the name of the book that contains the input sentence to the model, this problem was tackled by transforming the data into separated words using the techniques (BOW & TF-IDF), after that, the training algorithms of three different techniques (SVM, Decision Tree & K-Nearest Neighbour) using cross validation of 10 folds, to get the most suitable algorithm for that task. Based on the above-mentioned results for the three models, and the transformation techniques that were followed, it is found that the best algorithm for this task is the SVM Model with accuracy 96.6 %.

[1]Gamma Parameter: http://scikit-learn.org/