DTI5125[EG] Data Science Applications 2021

Group Assignment (1)

Under Supervision:

Prof. Dr. Arya Rahgozar

Group Members:

Abdelrahman Basha

Hussien Abdelraouf

Dina Ibrahim

Table of contents:

Abstract----------*3*
Introduction----------*3*
Methodology----------*3*
Dataset----------*3*
What is Gutenberg digital books ----------*4*
Implementation----------*4*
Result & Analysis----------*4*
Conclusion----------*21*

Abstract:

This report aims to produce classification predictions and compare them; analyze the pros and cons of algorithms and generate and communicate the insights and discuss the implementation steps of applying (preprocessing the data, transformation and evaluation,etc) strategies on three of multiclass classification algorithms (k-Nearest Neighbor, Support Vector Machine and Decision Tree algorithm). We will introduce the detailed steps of implementation of this strategy and discuss the achieved results. We used (five different samples of Gutenberg digital books) for implementation.

Introduction:

Text classification also known as text tagging or text categorization is the process of categorizing text into organized groups. By using Natural Language Processing (NLP), text classifiers can automatically analyze text and then assign a set of pre-defined tags or categories based on its content. Unstructured text is everywhere, such as emails, chat conversations, websites, and social media but it's hard to extract value from this data unless it's organized in a certain way. Doing so used to be a difficult and expensive process since it required spending time and resources to manually sort the data or creating handcrafted rules that are difficult to maintain. Text classifiers with NLP have proven to be a great alternative to structure textual data in a fast, cost-effective, and scalable way.

Methodology:

We prepared the data by creating random samples of 200 documents per book and limited them to 100 words. We preprocessed the data by removing stop words, doing stemming and lemmatization. We transformed it to BOW, TF-IDF, N-gram and LDA. We binarized the values to use them with the SVM, DT and KNN algorithms to get the accuracy. After that we made a cross validation with 10-Folds to evaluate them. We did an Error-Analysis for the champion model to get the miss-classified cases then we corrected it. We a made a bias-variance trade-off by calculating and analyzing them. Finally, we made a modification by increasing and decreasing the partitions and words to monitor the performance and accuracy of models.

Gutenberg project :

What is Gutenberg digital books

Project Gutenberg is a library of over 60,000 free eBooks.

Project Gutenberg (PG) is a volunteer effort to digitize and archive cultural works, as well as to "encourage the creation and distribution of eBooks." It was founded in 1971 by American

writer Michael S. Hart and is the oldest digital library. Most of the items in its collection are the full texts of books in the public domain. The Project tries to make these as free as possible, in long-lasting, open formats that can be used on almost any computer. As of 22 May 2021, Project Gutenberg had reached 65,405 items in its collection of free eBooks.

Gutenberg project features:

Project Gutenberg is zealously noncommercial, digitizes books in the public domain alone, and publishes an accurate rendition of the full electronic text.

Implementation:

This strategy implemented using Python programming language. The Natural Language Toolkit (or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. ... NLTK supports classification, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities. Scikit-Learn (sklearn library is used for building the models) and numpy library is used as a data structure. Matplotlib library is also used for visualizing the data, models' accuracy,etc.

Result & Analysis:

Preprocessing:

We made a preprocessing and cleaning using functions to convert the words to lower case, remove punctuation, special characters and stopwords, stemming to remove the (-ing, -ly, ...) and Lemmatisation (for converting the word to the root word ,e.g.. plays-playing-played to play).

```
def preprocessing_cleaning(  book_text , flg_stemm=False, flg_lemm=True   ):
  book_text = book_text.lower()                                  # convert to lower case
  tokenizer = RegexpTokenizer(r'\w+')                                  # remove Punctuation and special characters
  tokens = tokenizer.tokenize(book_text)                             # Remove stop words
  book_words = [w for w in tokens if not w in stopwords.words('english')]
  # return " ".join(filtered_words)
                                                #                Stemming (remove -ing, -ly, ...)
  if flg_stemm == True:
      ps = nltk.stem.porter.PorterStemmer()
      book_words = [ps.stem(word) for word in book_words]

                                                  #                Lemmatisation (convert the word into root word)
  if flg_lemm == True:
      lem = nltk.stem.wordnet.WordNetLemmatizer()
      book_words = [lem.lemmatize(word) for word in book_words]

  return book_words

# sentence = "At eight o'clock on Thursday morning Arthur didn't feel very good. French-Fries"
# print ( preprocessing_cleaning(sentence)  )
```

Partitioning:

We defined a function to get 200 random partitions of the books containing 100 words only and a loop to randomize and save the partition and the label.

```
def Partition_of_Books(  book_names , number_of_partitions  ,  number_of_words  ):
  book_partitions_list = [ ]     # list to save 200 partition for every book
  book_labels_list = [ ]         # list to save the label or book name for every partition
  for book in book_names:
    text = nltk.corpus.gutenberg.raw( book  )
    book_words = preprocessing_cleaning( text  )
    # tokenized_word =  nltk.word_tokenize(text)      # to split our text to words
    # text = nltk.Text( tokenized_word )
    # text = [word for word in text if not word in stopwords.words()]

    for x in range( number_of_partitions ):
      randoms  = np.random.randint( 0 ,  (len( book_words ) - number_of_words ) )  # to randomize the location where we get the partition
      partition_words = ( book_words[ randoms : randoms + number_of_words ] )   # save the partition
      partition_string = " ".join( partition_words )
      book_partitions_list.append( partition_string )
      book_labels_list.append( book )                                            # save the label
  return  book_partitions_list , book_labels_list
```

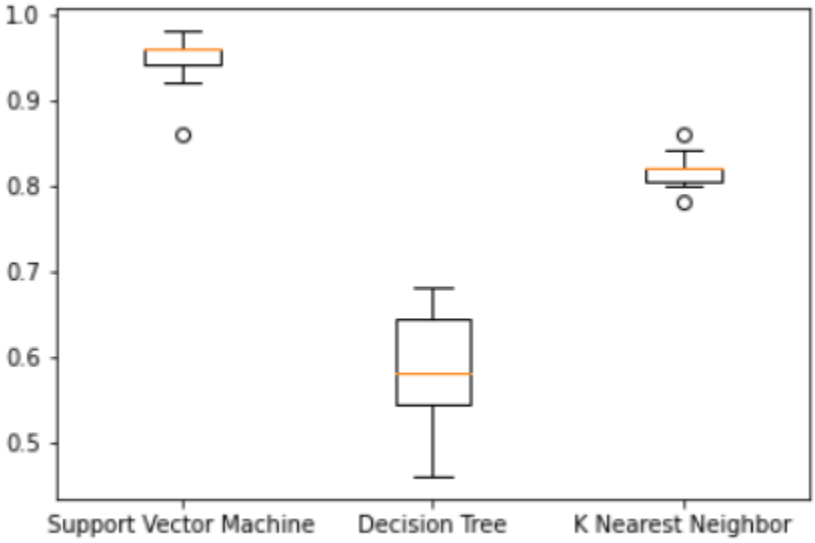
Feature engineering:

Bow and TF-IDF

We did it using Bow (bag of words) and TF-IDF. We calculated bag of words to get the frequency of every word in the text using CountVectorizer() . We also calculated the term-frequency (TF) and the inverse document frequency (idf) using TfidfTransformer().

```
[ ]  def Bow_Tf_Idf(   book_partitions_list  ):


        count_vect = CountVectorizer()
        X_train_counts = count_vect.fit_transform( book_partitions_list )  #
        tfidf_transformer = TfidfTransformer()
        X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)   # \
        # X_train_tfidf.shape
        return  X_train_tfidf
```

Figure for 10-Fold for (SVM , D_Tree , KNN) with TF_IDF



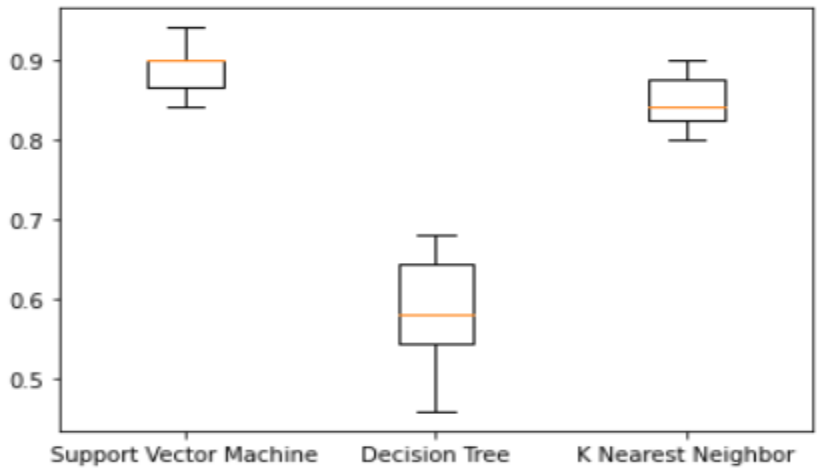
N-gram

We also used N-gram and calculated it as N-Grams model is one of the most widely used sentence-to-vector models since it captures the context between N-words in a sentence.

```
def ngram(book_partitions_list , n ):
  count_vect = CountVectorizer(ngram_range=(1,n))
  X_train_counts = count_vect.fit_transform( book_partitions_list )
  tfidf_transformer = TfidfTransformer()
  X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
  return  X_train_tfidf

c=ngram(X_train,3)
print(c.shape )
```

```
(700, 113672)
```

Figure for 10-Fold for (SVM , D_Tree , KNN) with N-gram



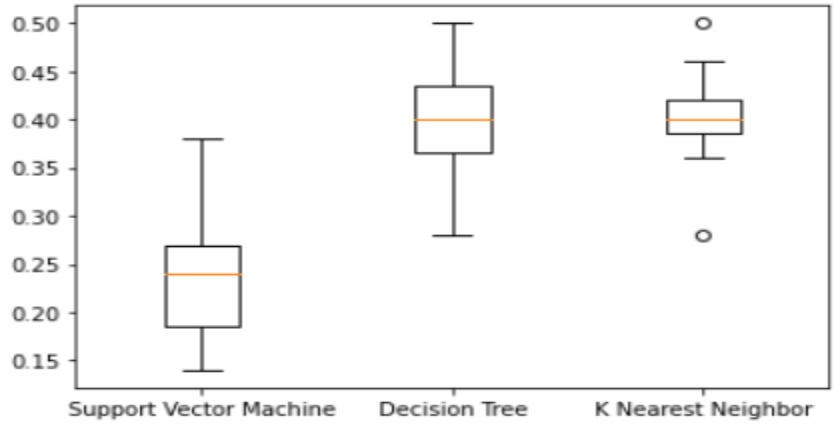
LDA (Latent Dirichlet Allocation)

SO as for LDA, we calculated LDA as it is an example of topic model and it is used to classify text in a document to a particular topic. It builds a topic per document model and words per topic model, modeled as Dirichlet distributions. Here we are going to apply LDA to a set of documents and split them into topics.

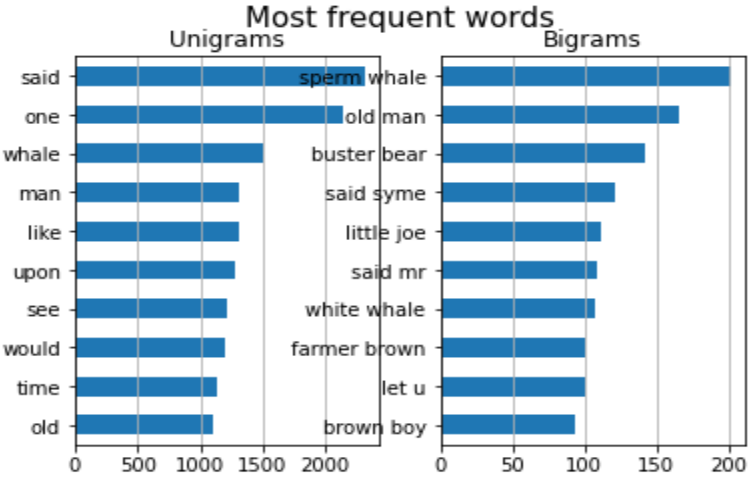
```
[12] def lda(book_partitions_list) :
       count_vect = CountVectorizer()
       X_train_counts = count_vect.fit_transform( book_partitions_list )
       LDA = LatentDirichletAllocation( n_components=7,random_state=42 )
       LDA.fit(X_train_counts)
       return LDA.transform(X_train_counts)

     t=lda(book_partitions_list)
     t.shape
```

```
(1000, 7)
```

Figure for 10-Fold for (SVM , D_Tree , KNN) with LDA



Visualizing the Data:



Binarizing Target

We defined a binarizing function in order to make the values with 0 and 1.

```
book_names = [ 'chesterton-thursday.txt' , 'edgeworth-parents.txt', 'melville-moby_dick.txt' , 'burgess-busterbrown.txt' , 'whitman-leaves.txt' ]

def Binarizing_Target(   book_labels_list ):
  book_labels_list = np.array( book_labels_list )
  for book in book_names:
    book_labels_list [ book_labels_list  ==  book ]  = int( book_names.index( book  ) )
  # if type(book_labels_list[0] ) is str:
  return book_labels_list
```

The algorithms

By using SVM we achieved 99.67%, 60% with Decision tree classifier and 96% with k-nearest neighbors as depicted in Figure. Table-1, shows accuracy summary per each of the models. Table-2, displays the confusion matrix for each of the models.

Algorithm	Binarized Models Accuracy
SVM	99.67%
Decision tree	60%
K-nearest neighbors	96%
Champion Model	SVM

Table-1: Models Accuracies

```
Accuuracy is      0.9966666666666667
Confusion Matrix
 [[55  0  0  0  0]
 [ 0 71  0  0  0]
 [ 0  0 56  0  1]
 [ 0  0  0 53  0]
 [ 0  0  0  0 64]]
Accuuracy is      0.6
Confusion Matrix
 [[45  0  0  0 10]
 [ 0  0  0  0 71]
 [ 0  0 27  0 30]
 [ 0  0  0 45  8]
 [ 0  0  1  0 63]]
Accuuracy is      0.96
Confusion Matrix
 [[54  0  0  1  0]
 [ 0 70  0  0  1]
 [ 0  1 51  2  3]
 [ 0  0  0 53  0]
 [ 1  2  0  1 60]]
Champion Model is : svm
```

Table 2-: Confusion matrix :

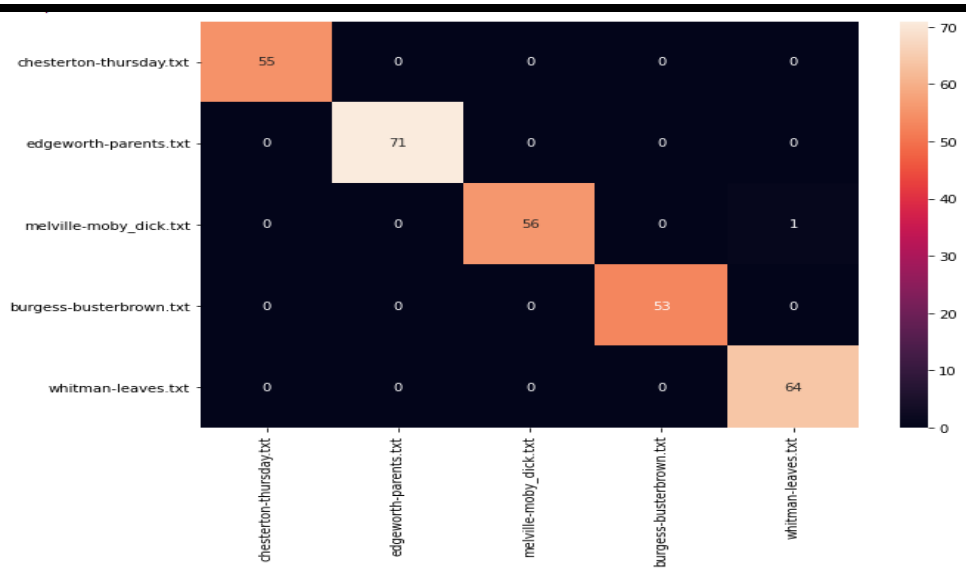


Figure 1: Confusion Matrix for SVM

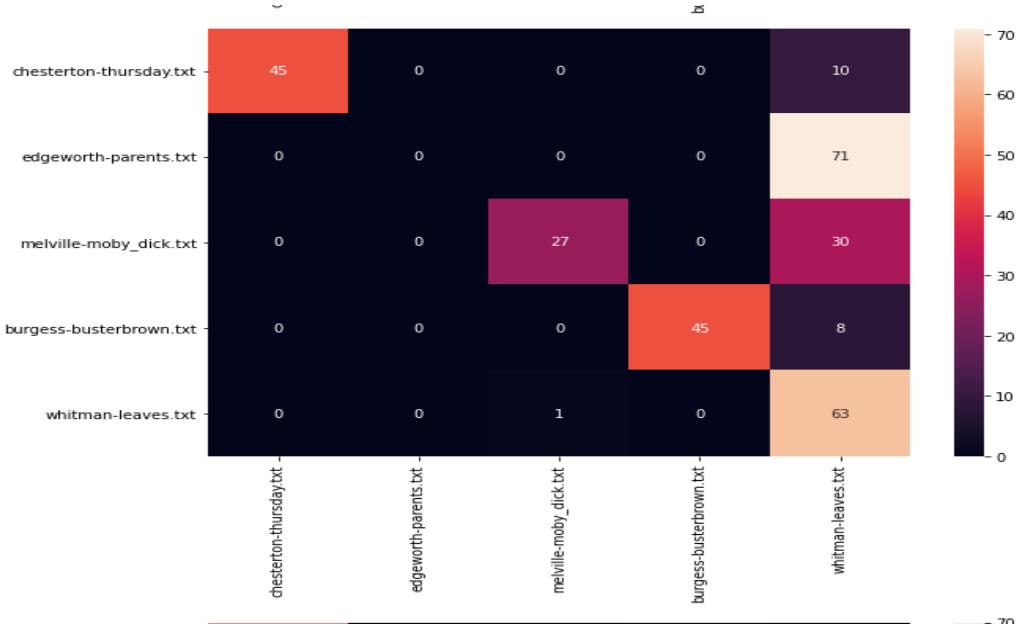


Figure 2: Confusion Matrix for Decision tree

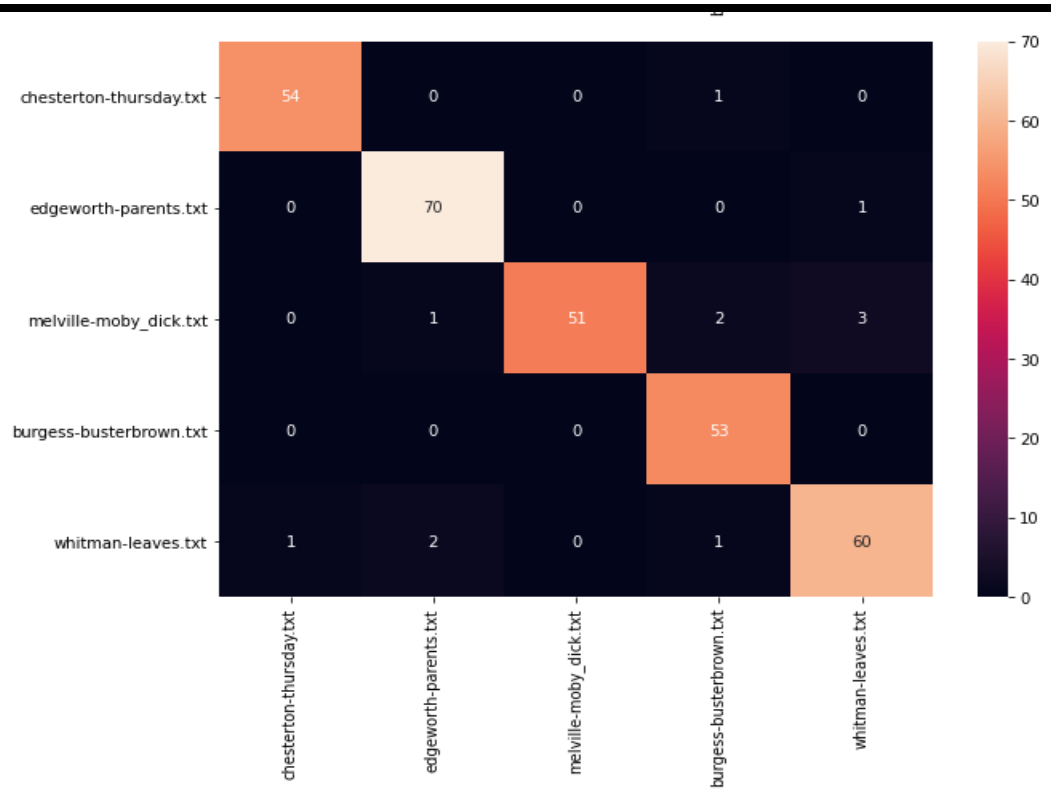


Figure 3: Confusion Matrix for K-nearest neighbors

Code:

```
def Build_Models( x , y , model_str = "svm" ):
  model_str = str.lower( model_str )
  y = np.array(y)
  model = None
  if model_str == "svm":
    model = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', SGDClassifier(loss='hinge', penalty='l2',
                          alpha=1e-3, random_state=42,
                          max_iter=5, tol=None)  )    ,           ])
                                    #Training
    model.fit(    x   ,  y )
  elif model_str == "d_tree":
    model = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('model', DecisionTreeClassifier(criterion = "gini",
            random_state = 100,max_depth=3, min_samples_leaf=5)   ), ] )
                                  #   # fitting the data in the pipe
    model = model.fit( x   ,  y  )

  elif model_str == "knn":
    model = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('model', KNeighborsClassifier(n_neighbors=7)   ),         ])
                                                 # Fitting our train data to the pipeline
    model.fit(  x  , y   )
  return model


def compute_confusion_matrix( actual , predicted  ):
  arr = np.zeros( ( len( book_names) , len( book_names) )  )
  print( len( book_names)  )
  for x in range( len( actual ))  :
      arr[ actual[x] , predicted[x] ] += 1
  return np.int_( arr )


def plot_confusion_matrix(   confusion_matrix ,  classes ):
  df_cfm = pd.DataFrame( confusion_matrix , index = classes, columns = classes)
  plt.figure(figsize = (10,7))
  cfm_plot = sn.heatmap(df_cfm, annot=True)
```

Evaluation models code:

```
model_names = [  "svm"  , "d_tree"   ,     "knn"    ]
acc=[]

y_train_binarized = Binarizing_Target( y_train )
y_test_binarized = Binarizing_Target( y_test )

y_test_binarized =  [ int (i, base=16) for i in y_test_binarized ]

#print( y_train_binarized )
#print(  y_test_binarized )

models = [ ]
for x in  range(len(model_names)) :
  models.append( Build_Models( X_train   , y_train_binarized  ,  model_names[x] )   )
  predicted = models[x].predict(Binarizing_Target( X_test ))
  predicted = [int(x) for x in predicted]
  acc.append(accuracy_score( y_test_binarized ,   predicted ) )
  print( "Accuuracy is     " ,  accuracy_score( y_test_binarized ,   predicted )  )
  confus_mtrix = confusion_matrix(  y_test_binarized ,   predicted )
  print( "Confusion Matrix    \n" , confus_mtrix   )
  # plot_confusion_matrix( model  ,   np.array ( partition_test ) ,   np.array( predicted ) )
  # print(  book_names [  int(predicted[0] )  ]  )
  # np.mean( predicted == book_names_bin  )
  plot_confusion_matrix(  confusion_matrix(   y_test_binarized , predicted )  , book_names )
  # print(    compute_confusion_matrix(   y_test_binarized , predicted ) )


print( "Highest Accuracy Model is : " +   model_names [ np.argmax(acc) ] )
```

After Evaluation process with 10-Folds Cross Validation:

We found that the accuracy got enhanced for both svm and decision tree as shown below.

Evaluation (10-Fold Cross Validation) for SVM , KNN , D_Tree

```
[ ]  # Models

     svm = SGDClassifier(loss='hinge', penalty='l2',  alpha=1e-3, random_state=42,  max_iter=5, tol=None)
     d_tree = DecisionTreeClassifier(criterion = "gini", random_state = 100,max_depth=3, min_samples_leaf=5)
     knn = KNeighborsClassifier(n_neighbors=7)


     X_train_tfidf  =  Bow_Tf_Idf(  book_partitions_list )


     models = [  svm   , d_tree ,  knn  ]
     model_names = [ "Support Vector Machine" , "Decision Tree"  ,  "K Nearest Neighbor" ]
     Model_scores = [ ]
     Model_accuracy = [ ]

     for x in range( 3 ):
       Model_scores.append ( cross_val_score(  models[x] ,    X_train_tfidf ,  book_labels_list ,   cv = 10, scoring= 'accuracy')  )

     for x in range( len( Model_scores ) ) :

       print(       model_names[ x ]  ,   " Scores is  "  , Model_scores[ x ]  )
       print("Accuracy: %0.2f (+/- %0.2f)" % (  Model_scores[ x ].mean()  ,   Model_scores[ x ].std() * 2)   )
       print( "////////////////////////////////////////// \n ")
       Model_accuracy.append(  Model_scores[ x ].mean()   )
```

Algorithm	Binarized Models Accuracy (10-Folds)
SVM	100%
Decision tree	66%
K-nearest neighbors	95%
Champion Model	SVM

Output :

```
Support Vector Machine  Scores is    [1.   1.   0.99 1.   0.99 1.   1.   1.   1.   0.99]
Accuracy: 1.00 (+/- 0.01)
//////////////////////////////////////////

Decision Tree  Scores is    [0.69 0.64 0.64 0.65 0.69 0.57 0.72 0.67 0.72 0.62]
Accuracy: 0.66 (+/- 0.09)
//////////////////////////////////////////

K Nearest Neighbor  Scores is    [0.94 0.96 0.94 0.99 0.9  0.95 0.94 0.97 0.94 0.93]
Accuracy: 0.95 (+/- 0.05)
//////////////////////////////////////////

Champion Model is :  Support Vector Machine
```

Error analysis :

According to the results after running K-fold on the SVM, DT, and KNN, we found that the highest mean accuracy is the Support Vector Machine. So, we considered the SVM as the champion Model to do our error analysis on.

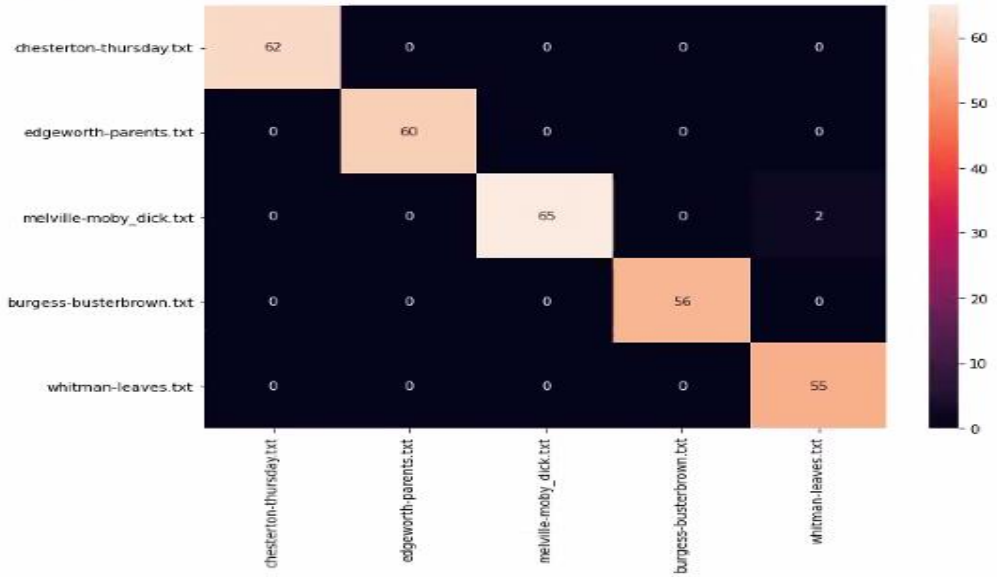
We defined a function to detect the mis-classified cases then we identified the correct labels for these records.

```
def detect_misclassified( actual , predicted  ):

  error_partition = [ ]
  for x in range( len( actual ))  :
    if(   actual[x] !=  predicted[x] ):
      error_partition.append( book_partitions_list[x]   )
      print( "" )
      print( "Misclassified = " , book_names  [predicted[x]]   )
      print( "Correct Classified = " , book_names[ actual[x] ]  )

  # print( "Error Partitions is :  \n" , error_partition   )


detect_misclassified(  y_test_binarized ,   predicted   )
```

```
Misclassified =  edgeworth-parents.txt
Correct Classified =  melville-moby_dick.txt
```

We trained the SVM (champion model) with our data , and then we computed the accuracy and we found that the acc is 93.33%.

```
Accuuracy is      0.9333333333333333
Confusion Matrix
 [[62  0  0  0  0]
 [ 0 60  0  0  0]
 [ 0  0 65  0  2]
 [ 0  0  0 56  0]
 [ 0  0  0  0 55]]
```



Specifying the characteristics of the records:

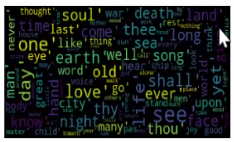
For the mis-classified cases, we first calculated the word cloud for these wrong records, we also calculated the word cloud for the predicted wrong books.

```
wc = wordcloud.WordCloud( background_color='black',    max_words=60 ,    max_font_size=35  ) # Here we computed the word cloud for the wrong partitions to know more about the highest frequency words
wc = wc.generate(str( error_partition  ))
fig = plt.figure(num=1)
plt.axis('off')
plt.imshow(wc, cmap=None)
plt.show(    )
```

Error partition

we computed the word cloud for the wrong partitions to know more about the highest frequency words.

```
all_error_books = [ ]
for x in range( len( error_books) ):          # Here we want to know more about the word frequeny in the wrong books that the algorithm has misclassified the partitions to them

  text = nltk.corpus.gutenberg.raw( error_books[x]  )
  book_words = preprocessing_cleaning( text  )
  all_error_books.append( book_words )


wc = wordcloud.WordCloud( background_color='black',    max_words=100 ,    max_font_size=35  )
wc = wc.generate(str( all_error_books  ))
fig = plt.figure(num=1)
plt.axis('off')
plt.imshow(wc, cmap=None)
plt.show()
                                #          we found that there are many words in the wrong partitions are similar to the words existed in the wrong books so the algorithm classified them to the wrong books.
```



Error books

Here we want to know more about the word frequency in the wrong books that the algorithm has misclassified the partitions to them.

Conclusion:

We found that there are many words in the wrong partitions are similar to the words existed in the wrong books so the algorithm classified them to the wrong books.

Bias and Variability:

We calculated the bias and variance as the bias is the amount that a model's prediction differs from the target value, compared to the training data and the variance indicates how much the estimate of the target function will alter if different training data were used. We consider the highest accuracy as the lowest bias and the lowest standard deviation as the lowest variance (SD is the square root of the Variance**)**.

```
# Here we consider the highest accuracy as   >>>   the lowest bias
# and the Lowest standard deviation as       >>>   the Lowest Variance


Models_highest_accuracy
Models_Less_std
Model_index

print(   Models_highest_accuracy)
print(Models_Less_std )

Highest_accuracy_index = np.argmax(  Models_highest_accuracy  )
Lowest_std = np.min( Models_Less_std )


print( "Lowest Bias is              >>>>>>>>>    "  , np.max(  Models_highest_accuracy ) )
print( "Lowest Standard deviation is >>>>>>>>>    " ,  np.min(  Models_Less_std ) )

print( model_names[ Model_index[ Highest_accuracy_index  ]  ] )
```

```
[0.992, 0.9870000000000001, 0.726]
[0.007483314773547889, 0.0100498756211209, 0.02481934729198172]
Lowest Bias is              >>>>>>>>>     0.992
Lowest Standard deviation is >>>>>>>>>     0.007483314773547889
Support Vector Machine
```

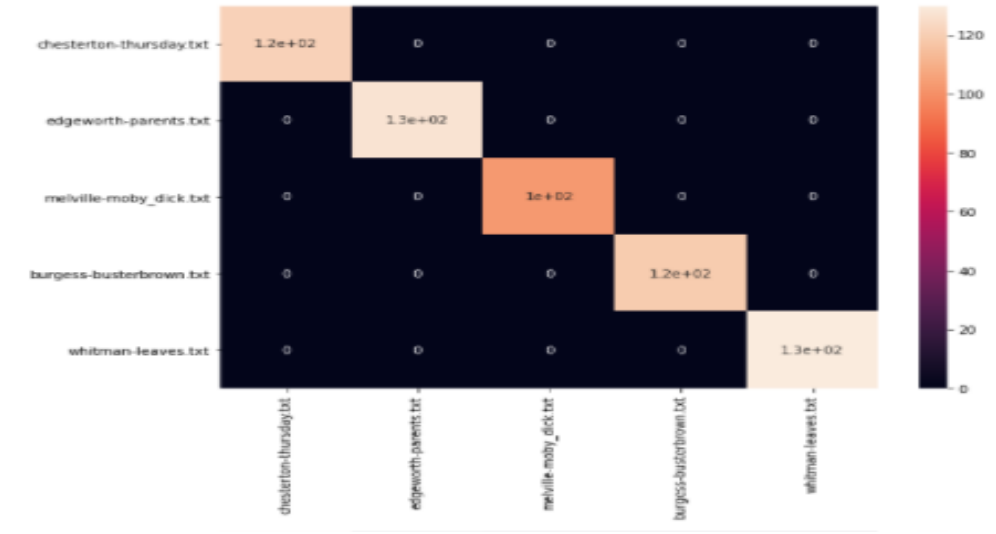
Identifying, measuring and control the machine's thresholds of factors of prediction hardship:

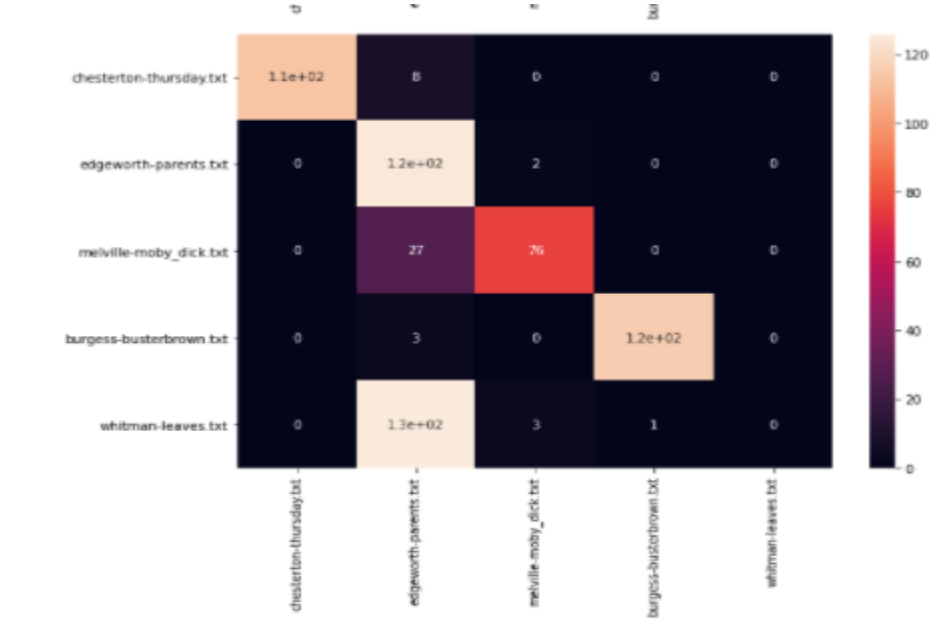
First-step: we increased the number of partitions and words to 400,200 respectively and we made an evaluation again and it achieved 100%,71.67% and 95.5% for SVM, Decision tree and KNN respectively.

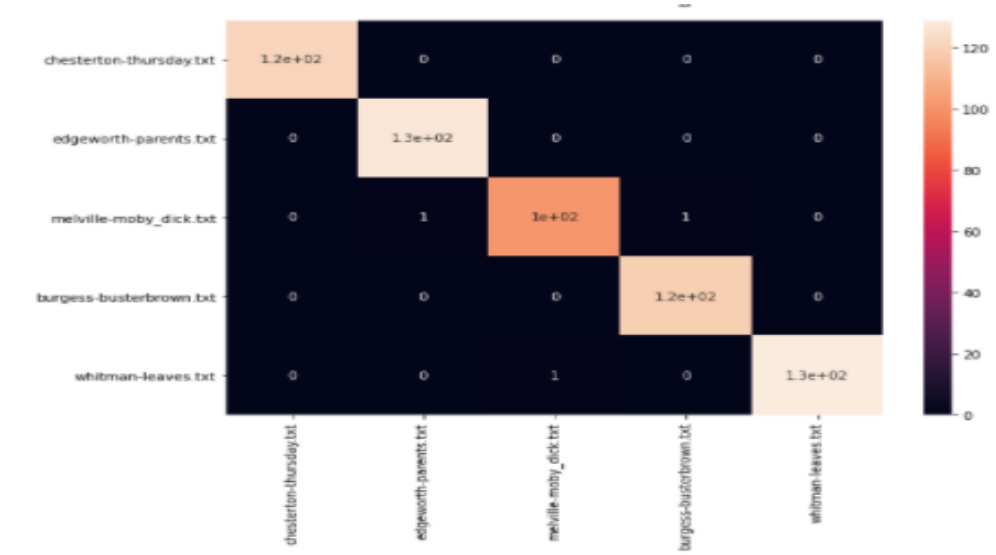
```
# set number_of_partitions per book = 400  partitions
# set number_of_words per partition = 200  words
number_of_partitions = 400
number_of_words = 200
book_partitions_list , book_labels_list =  Partition_of_Books(  book_names ,  number_of_partitions  ,  number_of_words   )

# print( len( book_partitions_list  ) )
# print( len(  book_labels_list ) )
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(  book_partitions_list , book_labels_list  ,  test_size=0.3, random_state=42)

    Evaluate_Models(  X_train,   y_train, X_test, y_test )
```

```
Accuuracy is      1.0
Confusion Matrix
 [[121   0   0   0   0]
 [  0 127   0   0   0]
 [  0   0 103   0   0]
 [  0   0   0 119   0]
 [  0   0   0   0 130]]
Accuuracy is      0.7166666666666667
Confusion Matrix
 [[113   8   0   0   0]
 [  0 125   2   0   0]
 [  0  27  76   0   0]
 [  0   3   0 116   0]
 [  0 126   3   1   0]]
Accuuracy is      0.995
Confusion Matrix
 [[121   0   0   0   0]
 [  0 127   0   0   0]
 [  0   1 101   1   0]
 [  0   0   0 119   0]
 [  0   0   1   0 129]]
Highest Accuracy Model is : svm
```


chesterton-thursday.txt
edgeworth-parents.txt
melville-moby_dick.txt
burgess-busterbrown.txt
whitman-leaves.txt
1.2e+02
1.3e+02
1e+02
1.2e+02
1.3e+02
120
100
80
60
40
20
0


chesterton-thursday.txt
edgeworth-parents.txt
melville-moby_dick.txt
burgess-busterbrown.txt
whitman-leaves.txt
1.1e+02
8
1.2e+02
2
27
76
3
1.2e+02
1.3e+02
3
1
120
100
80
60
40
20
0

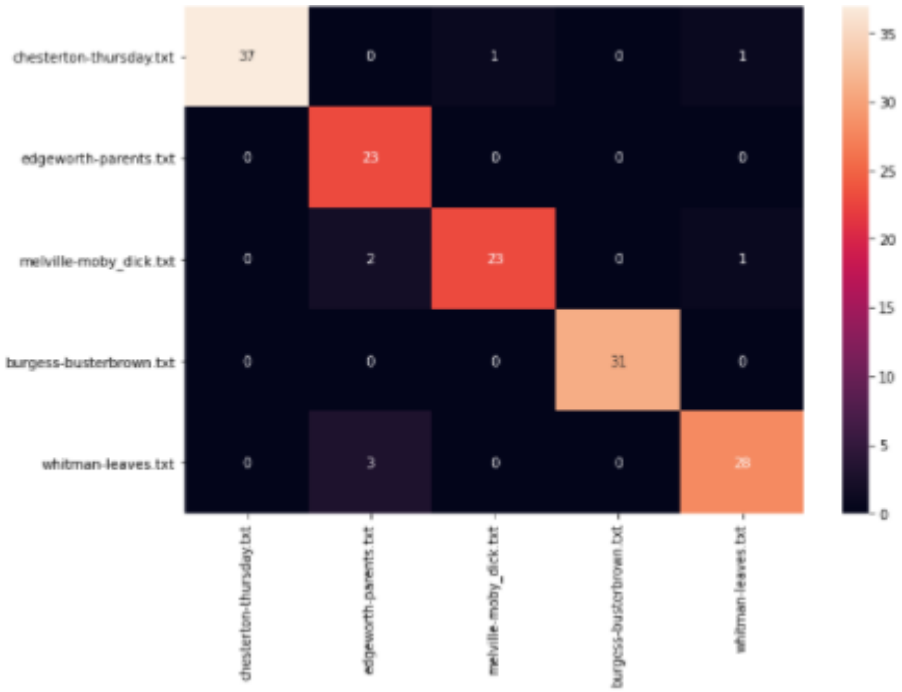

chesterton-thursday.txt
edgeworth-parents.txt
melville-moby_dick.txt
burgess-busterbrown.txt
whitman-leaves.txt
1.2e+02
1.3e+02
1
1e+02
1
1.2e+02
1
1.3e+02
120
100
80
60
40
20
0

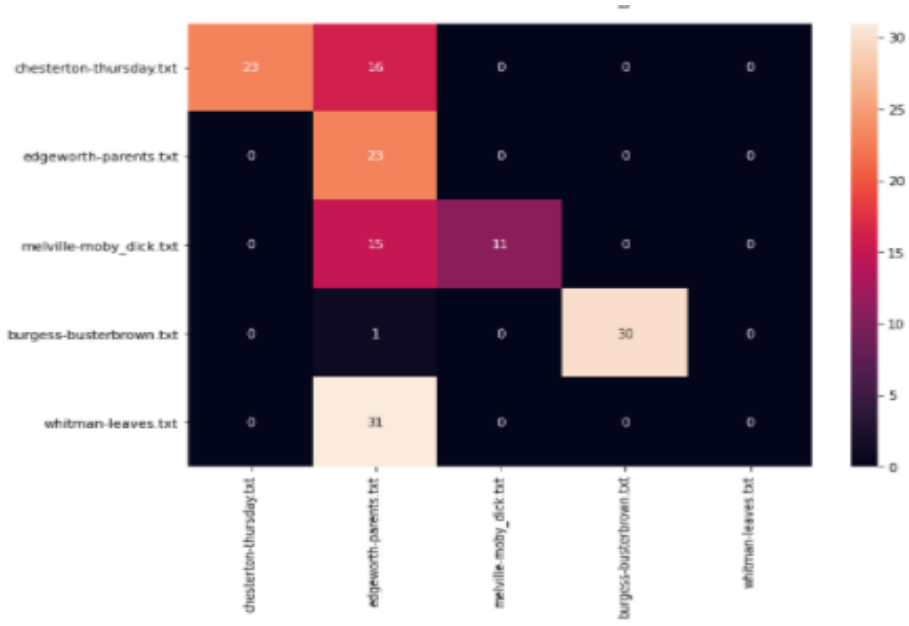
Second-step: we decreased the number of partitions and words to 100,50 respectively and we made an evaluation again and it achieved 94.67%,58% and 83.33% for SVM, Decision tree and KNN respectively.

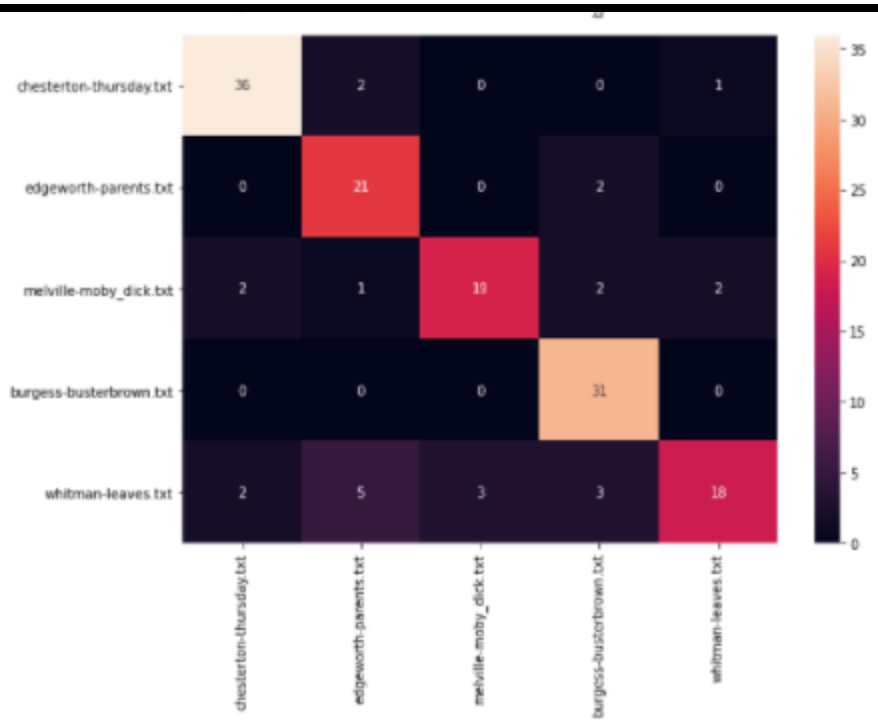
```
[ ]
    # set number_of_partitions per book = 100  partitions
    # set number_of_words per partition = 50  words
    number_of_partitions = 100
    number_of_words = 50
    book_partitions_list , book_labels_list =  Partition_of_Books(  book_names ,  number_of_partitions  ,  number_of_words   )

    # print( len( book_partitions_list  ) )
    # print( len(  book_labels_list ) )
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(  book_partitions_list , book_labels_list  ,  test_size=0.3, random_state=42)

    Evaluate_Models(  X_train,   y_train, X_test, y_test )
```

```
Accuuracy is      0.9466666666666667
Confusion Matrix
 [[37  0  1  0  1]
 [ 0 23  0  0  0]
 [ 0  2 23  0  1]
 [ 0  0  0 31  0]
 [ 0  3  0  0 28]]
Accuuracy is      0.58
Confusion Matrix
 [[23 16  0  0  0]
 [ 0 23  0  0  0]
 [ 0 15 11  0  0]
 [ 0  1  0 30  0]
 [ 0 31  0  0  0]]
Accuuracy is      0.8333333333333334
Confusion Matrix
 [[36  2  0  0  1]
 [ 0 21  0  2  0]
 [ 2  1 19  2  2]
 [ 0  0  0 31  0]
 [ 2  5  3  3 18]]
Highest Accuracy Model is : svm
```


chesterton-thursday.txt
edgeworth-parents.txt
melville-moby_dick.txt
burgess-busterbrown.txt
whitman-leaves.txt
37 0 1 0 1
0 23 0 0 0
0 2 23 0 1
0 0 0 31 0
0 3 0 0 28
35
30
25
20
15
10
5
0


chesterton-thursday.txt
edgeworth-parents.txt
melville-moby_dick.txt
burgess-busterbrown.txt
whitman-leaves.txt
23 16 0 0 0
0 23 0 0 0
0 15 11 0 0
0 1 0 30 0
0 31 0 0 0
30
25
20
15
10
5
0



Conclusion:

We implemented the steps (preparing till verifying and validating) using by using the SVM and DT algorithms. In addition to, implementing the KNN algorithm for text classification. We found that SVM provides the best performance among them from accuracy perspective.