Text Classification Assignment 1

Report Submitted to Prof. Arya Rahgozar

GNG5125

Lasya Bhaskara

Tulika Shukla

Xun Xun Shi

February 2, 2021

Department of Engineering

University of Ottawa

Ottawa, Canada

Contents

Table of Figures

Introduction

This report contains a developed solution towards the problem of predicting for the authorship of a book given a passage inside a book. Because this set of data is already annotated, this is a supervised machine learning problem. This report includes the investigation of four different supervised machine learning models (Naives Bayes, K Nearest Neighbor, Random Forest and SVM) along with two different preprocessing step (Bag of words and TF-IDF) on solving this problem. This report also further investigates the variance/bias associated with using each model, and selection of the optimal model that has both a low bias and low variance.

Data

We have chosen a range of books that were published between the 2000s the 1600s. We hypothesize that the books written in closer time will be harder to classify, and the features will be more similar.

The following books includes:

1. The Man Who Was Thursday - G. K. Chesterton (1908)/ Chesterton-Thursday
2. Alice's Adventures in Wonderland – Lewis Carroll (1800) / Carroll Alice
3. Stories – E. Bryant (1900-2000) / Bryant-stories
4. Paradise Lost – John Milton (1667) / Militon-Paradise
5. Hamlet – William Shakespeare (1609) /Shakespeare-Hamlet

Data Preparation for Book partition dataset:

To have a balanced data set, we have taken 200 passages containing 100 words from each book. Five different books from Gutenberg's digital library were selected to create the dataset. A function was developed to return the partitioned book as a data frame with 200 rows, each row containing 100 words. A total of 1000 records consisting of 200 passages from each book were generated.

Table 1. Data frame of the partitioned data. *Passage is X and bookName is Y.*

	passage	bookName
378	well known to the police , and I can see well ...	chesterton-thursday
350	the first engagement . When he came down very ...	chesterton-thursday
420	a gallop , without respect to the rather rugge...	chesterton-thursday
124	destroy the very idea of personal possession	chesterton-thursday
179	gas . But the truth was that by this time he h...	chesterton-thursday
...	...	...
784	to a stranger , that the true Anointed King Me...	milton-paradise
129	bring Their embryon atoms : they around the fl...	milton-paradise
618	. Thy sorrow I will greatly multiply By thy co...	milton-paradise
416	: still govern thou my song , Urania , and fit...	milton-paradise
557	envying stood , but could not reach . Amid the...	milton-paradise

Data Preprocessing and Data Cleansing:

Spacy:

Our choice of Spacy library as an open source library for preprocessing was motivated by the fact that it can perform tokenization in 0.2 milliseconds compared to nltk's 4 milliseconds. It creates pipelines which can automatically perform the tokenization including tagging, parsing, named entity recognition etc.

Motivation for using lemmatization and avoiding stop words:

Getting rid of the punctuations and performing Lemmatization on the text data was done using the Spacy library.

As a first step, lemmatization was performed. It is a process of converting many different forms to its root word. For example, words such as building, built, build etc will be converted to its base form or lemma 'build', based on the context of the words surrounding it. The nlp pipeline created using Spacy will automatically perform the tokenization, parsing and tagging processes for us. The code was executed with and without this preprocessing step and the accuracy has improved with preprocessing.

Stop words were not removed during preprocessing because these features assist in adding context to the classification. Since our data contains both past century books such as Shakespeare's Hamlet and militon (written in the 1600s), the common language is different than modern books. These stop words could serve as key feature to help distinguish these differences from modern books in this data set such as Bryant's stories, which was written in the 1900s. Removing them could potentially reduce the accuracy of the model in classifying such books. In our case, we did look at other feature engineering steps (such as TF-IDF) that will remove common words based on the frequency instead.

After the preprocessing, the data was split into 30% of test data and 70% of train data for the further processing.

Feature Engineering

Feature engineering is the process of converting raw textual data into numerical vectors

Bag of Words (BoW) and Term frequency – Inverse document frequency(TF-IDF) were the statistical measures used in order to perform text feature extraction on the book passages.

TF-IDF:

As a first choice, TF-IDF was used to transform X (Passages) into vectors. TF-IDF is a vectorization algorithm which is used to represent textual data in numerical vectors. It gives weights to the words depending on their frequency. Compared to bag of words, TF-IDF also calculates the inverse document frequency, which will factor in the frequency of the word to occur in all documents. This will take out very commonly used words.

Bag of Words:

Bag of words is a form of vectorization process for extracting features from the textual data. Like TF-IDF, this is another step of feature processing, but it does not take into account how often the words will appear across different documents. The term 'bag' implies that the order of the words appearing in the document is ignored. The model is only concerned with whether the word is present in a document, not with the location of the word in the document.

Modelling

The models

Supervised machine learning algorithms are trained using labeled data as an input, where the input is the feature, and the y put is previously annotated label. In this case, the X input is the word passage, and the Y output is the book author. Such algorithm then learns by comparing and monitoring the loss between the prediction output with correct output.

The following models were selected in order to identify the authorship of the passages.

Linear Support Vector Classifier (SVC):

A Linear Support Vector Classifier (SVC), uses a hyperplane that separates the data into groups. The objective is to fit to the data you provide, returning a "best fit" hyperplane that categorizes the data. The error margin in a support vector machine determines how harsh the penalty will be for wrong misclassified data. [1]

Random Forests:

Random forest is a supervised learning algorithm which operates on a large number of decision trees grouped together. Each individual tree returns a class prediction and the class with the most votes will be selected. Since the correlation between the models is low due to method such as strapping, this algorithm is less prone to overfitting compared to decision tree, making it a more desirable choice compared to algorithms such as decision trees. [1]

Naïve Bayes:

Naive Bayes (NV) Classifier uses Bayes' theorem to classify the passages. This is the only parametric models that was looked at in this project. Due to its class independence assumption, these classifiers are low bias and performs well on small dataset. [1]

KNN:

KNN(K-Nearest Neighbor) is a supervised learning algorithm which operates on minimalizing the distance between features. [1]

Motivation to perform grid search with cross validation

Bias and variance tradeoff:

There is a tradeoff between bias and variance. If a model is not complex enough, it could be great at generalizing trends resulting in low variance, but could suffer from high bias, where the accuracy of the overall prediction is not great. These are cases of underfitting, where both the

model's training and testing accuracy are low. Generally, increasing the complexity of an algorithm could decrease the bias in the data. However, greatly increasing complexity of the algorithm could also lead to reduction in the variance, yielding great results on the training set, but produces poorly on new data. The objective of grid-search with cross validation is to determine the optimal hyper parameter values that will yield low bias and low variance.

Hyper Parameter Tuning:

Hyper parameter tuning is the objective of finding the optimal combination of hyper parameters for a given machine learning algorithm. Such parameter combinations should that deliver great performance on not only the training set, but specifically the validation set. This can be achieved using a grid search.

Grid Search:

Grid-searching is a method of parameter tuning in search of an optimal hyper parameter combination for the model. During grid search, the model will predict on the training data with different parameter combinations. Monitoring the performance of each hyper parameter combination will provide insight on the model performance. Grid search can be combined with cross validation to monitor the performance on both the training and test set, which can reveal any effect the parameter value may have on the model's bias and variance.

Cross Validation:

Cross-validation is a technique of splitting the training data set into different partitions and evaluate the model's performance on each partition. Ten fold cross validation was performed for each grid search, where the training set has been partitioned to 10 equal parts, and training-testing was performed 10 times leaving a different part out each time. It is important to note that the 10-fold cross validation was used to split the training set, an individual test set was left out for evaluating the final model. This is also represented by the figure below.

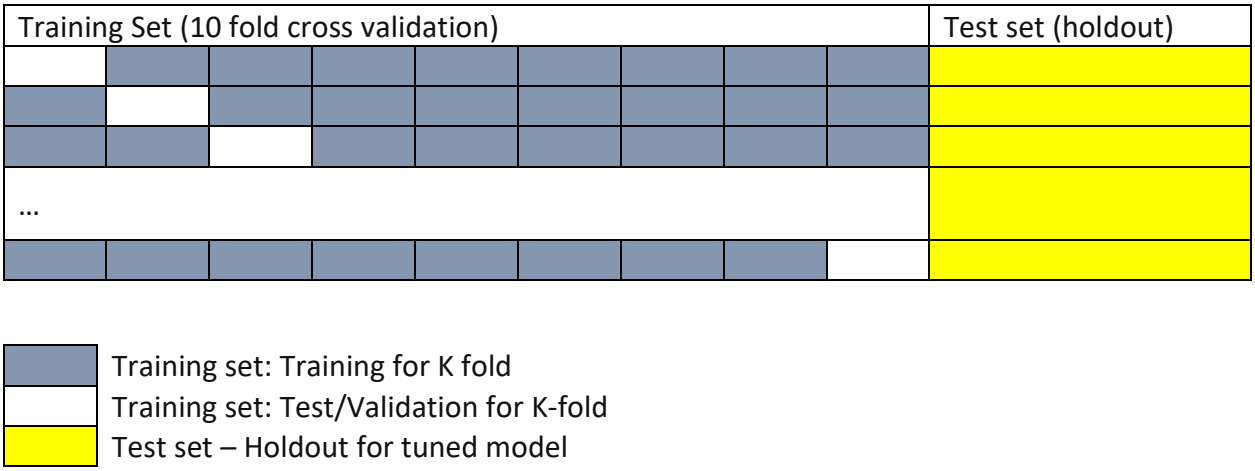


Figure 1. Train test data split schema.

Error plots:

In this report, error plots were used to determine the effect of incrementing the tuned parameter values on learning and test performance. Comparing the training and test data set's performance of these plots will give us insight on the parameter effect on each model's bias and variance.

Results of Grid search

SVC

One of the key parameters in SVC is the error margin width, where it is denoted here as the parameter C. This parameter determines how much penalty is given to points that are close to the hyper plane. The width of this margin is the parameter that was tuned, from a value of 1 to 10. The effect of using hinge loss Vs squared loss is also evaluated. The figure below demonstrates the effect of C value during hinge loss and squared hinge loss, for both cases when preprocessing was done with TF-IDF and bag of words.

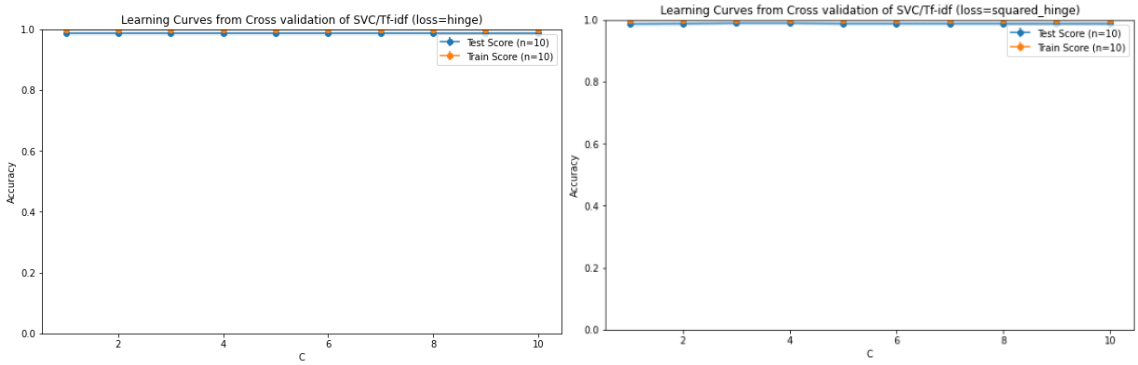


Figure 2. Error plot for TF-IDF - SVC with hinge loss and squared hinge loss . *The orange line indicates the training and the blue line indicates the test score, both aggregated across all 10-fold of cross validation.*

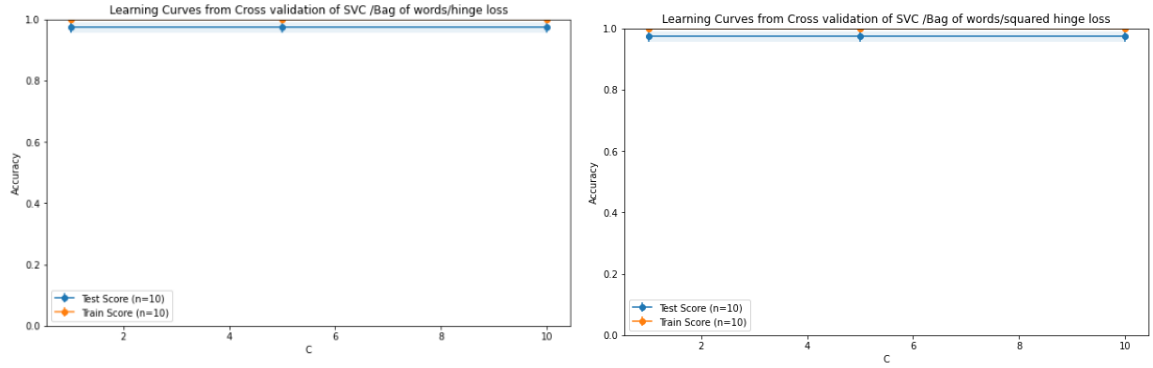


Figure 3. Error plot for Bag of words - SVC with hinge loss and squared hinge loss. *The orange line indicates the training and the blue line indicates the test score, both aggregated across all 10-fold of cross validation.*

In Figure 2 where the model uses SVC for preprocessing, there is neither an over fit nor an under fit for the changes in the parameter. The average train and test scores are almost overlapping for all ranges of C. Both trends when using hinge loss or squared hinge loss are very similar. Therefore, it can be concluded that any combination of C value to either hinge loss/ squared

hinge loss is an appropriate hyper parameter value, as it leads to model with low bias and low variance, based on the high-test accuracy and train accuracy.

However, when preprocessing with changed to bag of words, there is higher variance in the test set data (as shown by the wider blue band), along with a slight decrease in the mean test set error (as the blue line is lower than the orange line). It is important to point out that this indicates using Bag of words leads to a slightly higher bias and higher variance of a model compared to TF-IDF.

Using the highest mean test value (not visible on the graph), the optimal hyper parameter combination is therefore an error margin of 3 and a squared hinge loss with TF-IDF preprocessing.

KNN

For KNN, a grid search was performed to evaluate whether Manhattan distance (p=1) or Euclidian distance (p=2) should be used, and if the weight assigned should be uniform or inversely proportional to the distance.

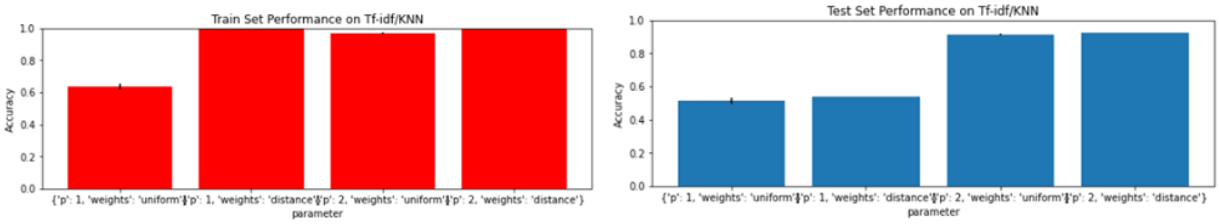


Figure 4. Error bar for TF-IDF - KNN with training error on the left and test error on the right. *P=1 indicates Manhattan distance while p=2 indicates Euclidean.*

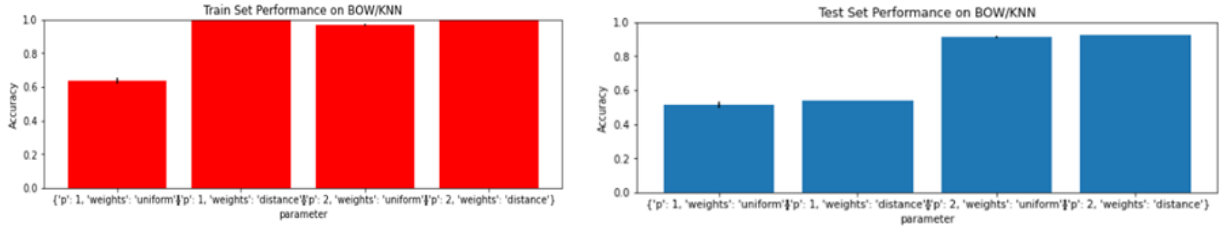


Figure 5. Error bar for TF-IDF - BOW with training error on the left and test error on the right. *P=1 indicates Manhattan distance while p=2 indicates Euclidean.*

The same trend was noticed for both preprocessing steps of TF-IDF and BOW.

For Manhattan(p=1) distance, the average mean of cross-validation test data has low accuracy (0.5) for both uniform and inverse-distance weights. But for different reasons. Uniform weights have low average training accuracy and low-test accuracy, indicating underfitting (high bias and low variance). While inverse-distance weights have high average training accuracy and low-test accuracy, which indicates that the model is overfitting (low bias and high variance).

For Euclidean (p=2) distance, both uniform and inverse-distance weights have high training accuracy and high-test accuracy. These indicates good models that can generalize to new data. Therefore, Euclidian distance should be used with either random or inverse-distance.

Grid search performed has shown that the optimal hyper parameters values is using a p=2 (Euclidean distance) and using inverse-distance weights. Under these parameters combinations, the KNN/TF-IDF and for bag of words grid search yield similar results.

Random Forests

Random forest is a group of decision trees. The parameters optimized here is the number of trees that is needed for a great model with low bias and low variance. The effect of bootstrapping is also investigated to determine whether bootstrap should be used in the final model.

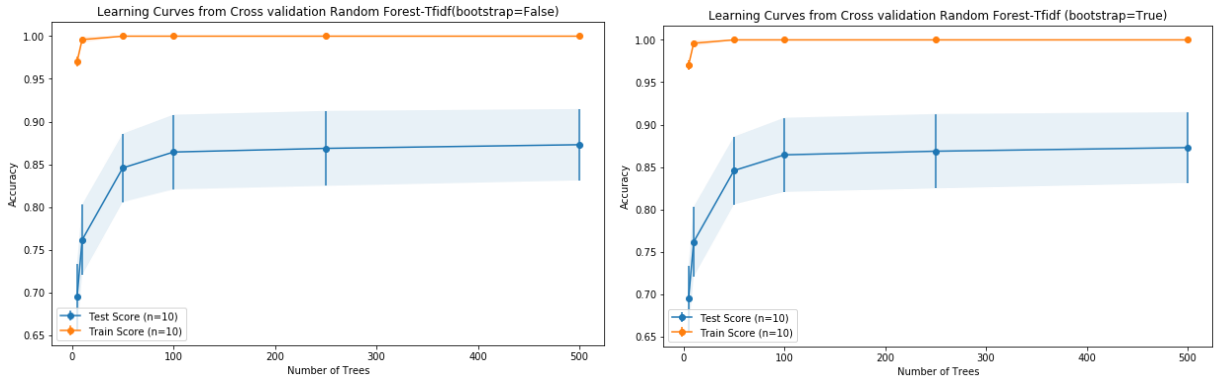


Figure 6 Error plot of 10-fold cross validation result for different number of trees for a random forest classifer with TF-IDF. *Left plot is without bootstrapping and right plot is with bootstrapping. The orange line indicates the training and the blue line indicates the test score, both aggregated across all 10-fold of cross validation.*

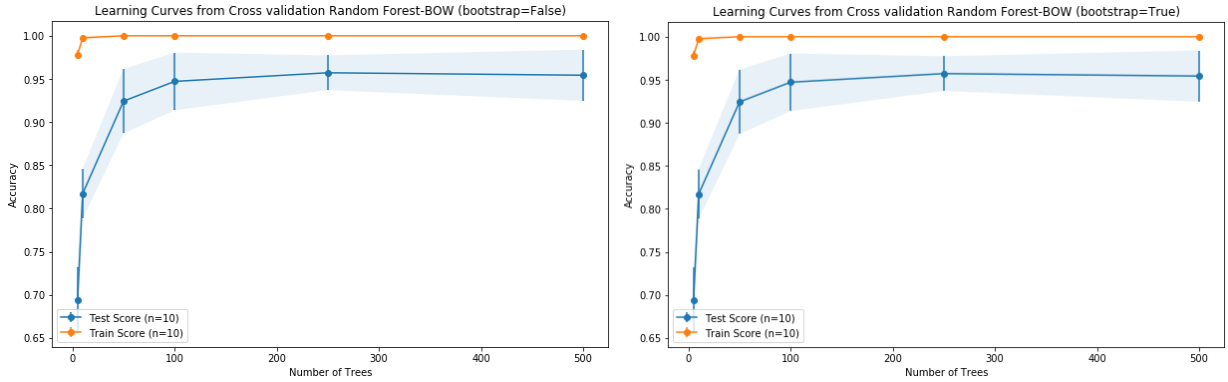


Figure 7. Error plot of 10-fold cross validation result for different number of trees for a random forest classifier with Bag of Words. *Left plot is without bootstrapping and right plot is with bootstrapping. The orange line indicates the training and the blue line indicates the test score, both aggregated across all 10-fold of cross validation.*

Across all models, the left and right plots follow similar pattern, indicating that bootstrapping does not play an important role on the model's performance. The test set data average increases drastically until 100 trees, indicating that at least 100 trees are needed to avoid underfitting.

For Random forests/TF-IDF the plotted graph, the mean test score seems to be stabilized after 100 trees, with addition of trees playing very little effect on the test data. This means that adding more trees may not greatly increase the performance of the trees after 100 trees, but there is very little sign of overfitting with increasing trees from 100 to 500 (no decrease in test data). However, given that these models have a high training accuracy (>95%) and lower testing accuracy compared to Bag of words (~85%), these model could not fully capitulate the true relationship between the features and predictions, and may be underfitting the model with low

bias (compared to bag of words). Therefore, the optimal number of trees selected for TF-IDF/random forest should be 500, and the effect of bootstrapping is not significant.

For the bag of words plots, the average test set accuracy significantly increases to 95% after 100 trees, indicating a model with fairly low variance and low bias. It was observed that as tree increased from 250 to 500, the shaded blue area increases, and the average test mean accuracy slightly decreases. This means that while increasing the number of trees from 250 to 500 may have decreased model bias, it has simultaneously increased the model variance, and may be more prone to over fitting. This case is true for both with bootstrap and without bootstrap. Therefore, 250 trees should be selected for the bag of words. Therefore, the optimal number of trees selected for BOW/random forest should be 250, and the effect of bootstrapping is not significant.

Grid search performed has shown that the optimal hyper parameters values is using 250 trees with bootstrapping for Bag-of-words with random forest, whereas when TF-IDF is used, the optimal number of trees should be 500 with bootstrapping.

Naïve Bayes

Navies bayes is a probability-based classifier, but the problem can arise for assigning a probability to a new data point that has not been previously observed by the classifier. The term Laplace smoothing is an added term that dictates the probability assigned to these data points. The effect of varying this parameter from 0 to 1 is investigated on the cross-fold accuracy.

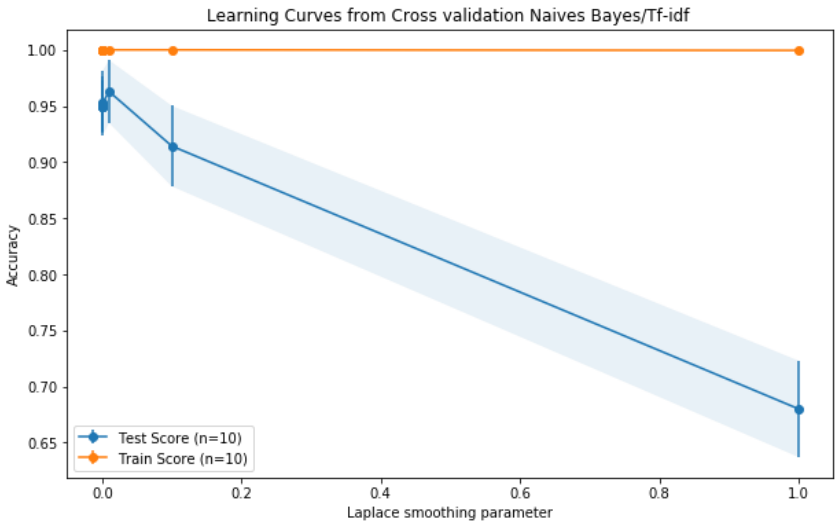


Figure 8. Error plot of 10-fold cross validation result for different value of smoothing parameter for a Naives bayes classifies with TF-IDF. *The orange line indicates the training and the blue line indicates the test score, both aggregated across all 10-fold of cross validation.*

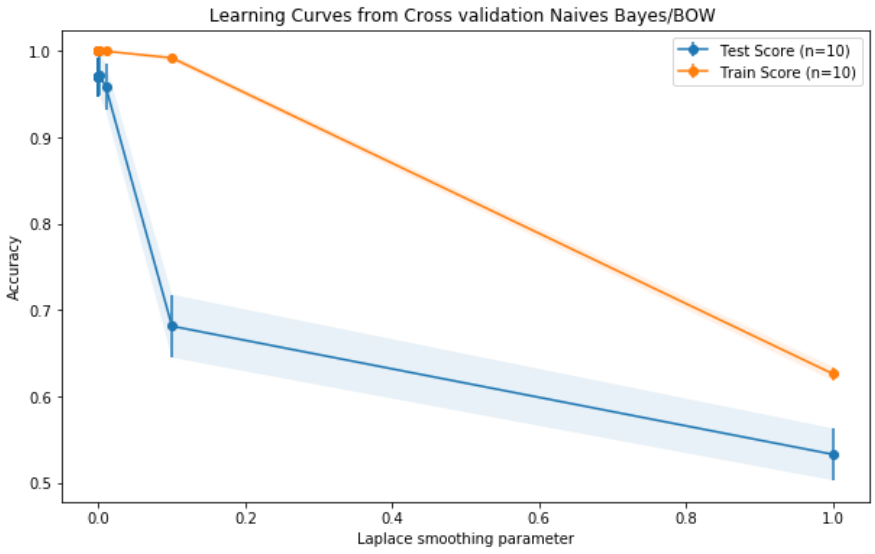


Figure 9. Error plot of 10-fold cross validation result for different value of smoothing parameter for a Naives bayes classifies with bag of words. *The orange line indicates the training and the blue line indicates the test score, both aggregated across all 10-fold of cross validation.*

For Naïve bayes with bag of words, Initially, it was observed that with the increase in smoothing parameter, there is an increase in average test performance, this means the model is decreasing both bias and variance. However, the mean test score seems to drop off significantly was we increase the parameter past 0.001. This means that using a smoothing value greater than 0.001 will do great on training data but bad in test data. Therefore, we can conclude that using smoothing parameter higher than 0.001 will lead to overfitting, where the model will have low bias but high variance. The optimal parameter value is therefore 0.001. These results are similar to Naïve Bayes with TF-IDF except the optimal is 0.001 instead of 0.01 as in the case of TF-IDF.

Summary of Optimal Parameters:

The table below demonstrates the summary of the best parameters aggregated from grid-search.

Table 2. Optimal Hyper parameters compared across TF-IDF and BOW with various algorithms.

		SVC	Random Forest	Naïve Bayes	KNN
Hyper parameter tuned		• C • type of loss	• Whether to use boot strap • type of loss	• Laplace smoothing parameter	• type of distance monitoring (p) • weights
Optimal Combination	TF-IDF	• C=3 • loss = squared hinge	• Use Bootstrap • 500 Trues	• Smoothing parameter = 0.01	• p = 2 (Manhattan) • weights based on inverse distance
	BOW	• C=1 • squared hinge	• Use Bootstrap • 250 Trees	• Smoothing parameter = 0.001	• p = 2 (Manhattan weights using inverse distance

Using each of the selected hyper parameters as the final tuned parameter, we can evaluate each model on the left out testing set. The results of this evaluation follow as below.

Testing results

ROC and Confusion Matrix

A Receiver Operating Characteristic curve is a graph that shows the performance of a classification model for different classes based on the true positive and false positive rates. A Confusion demonstrates the performance of a classification model in a N x N matrix, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well the classification model is performing and what kinds of errors it is making.

In this analysis, both ROC and confusion matrix are used for a through analysis on the model performance and the type of misclassification that were made.

KNN:

The following figures are the ROC and confusion matrix results of the test data that were predicted by the KNN using Manhattan distance and inverse distance, with both TF-IDF and Bag of words as preprocessing steps.

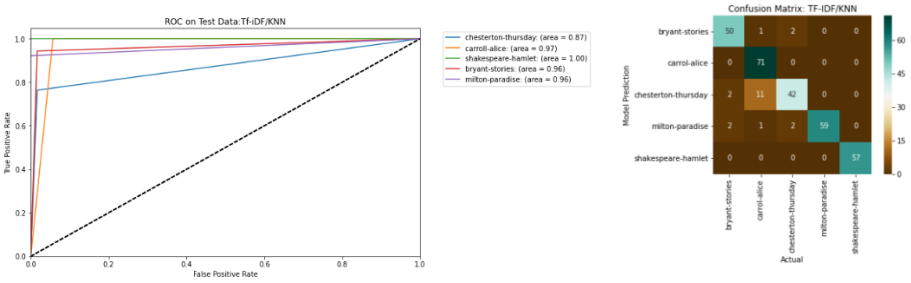


Figure 10. ROC and confusion matrix of test data for TF-IDF with KNN.

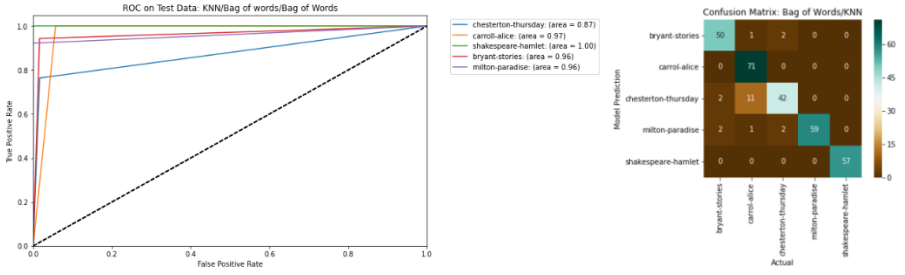


Figure 11. ROC and confusion matrix of test data for Bag of Words with KNN.

Both KNN with TF-IDF and Bag of words yielded the same result. The confusion matrix shows fairly good results. Most of the values are mainly in the diagonal and the classification report shows an accuracy of 0.93. There are a lot of wrong values specifically in the columns 2, row 3.

This means that the classifier is often mislabeling the book chesteron-thursday into carroll-alice. It's possible that there are overlapping words used in these two books, as these books were written in the same era. The classification report shows an accuracy of 0.93 which means it is a pretty good classifier.

The ROC for KNN/TF-IDF demonstrates that area under the curve are all fairly high, with the exception of chesterton-thursday. This means that this classifier is having difficulty predicting for chesterton-Thursday, an observation previously identified in the confusion matrix. Moreover, ROC area under the curve is high. Overall, these results are good.

Random Forests:

The following figures are the ROC and confusion matrix results of the test data that were predicted by the Random Forest, using 500 and 250 trees respectively for TF-IDF and Bag of words.

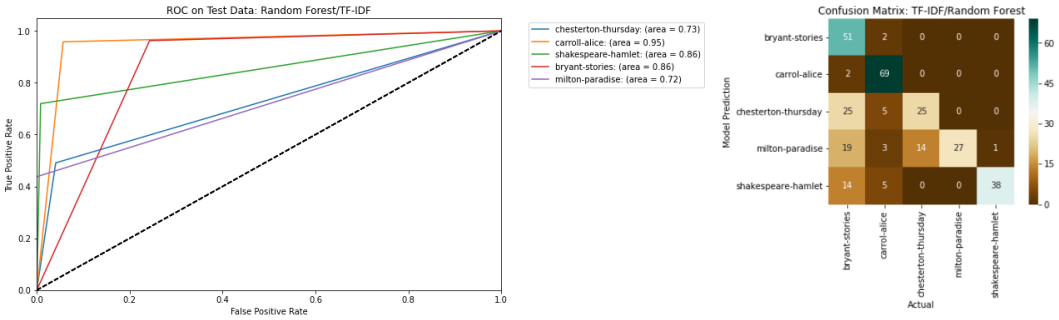


Figure 12. ROC and confusion matrix of test data for TF-IDF with Random Forest.

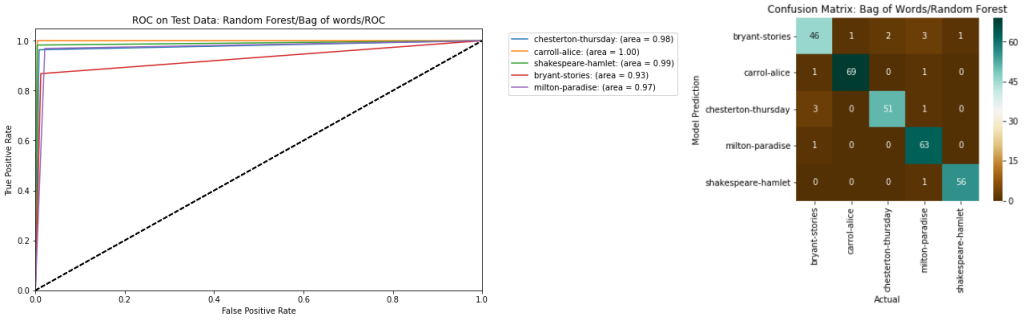


Figure 13. ROC and confusion matrix of test data for Bag of Words with Random Forest.

For Random Forests/TF-IDF, the confusion matrix does not seem too promising, a lot of numbers are not on the diagonal whereas for bag of words, most of the numbers are on the diagonal axis showing that it performs a lot better than the model preprocessed with the TF-IDF.

The classification report for Random Forest/TF-IDF provides only a 0.72 score of accuracy while for bag of words, it shows that a higher accuracy of 0.96 on the test data. Bag of words with Random Forest has a much better performance than with Tf-idf, where it previously scored an accuracy of 0.71.

The areas covered by the ROC for Random forest/TF-IDF plot for each class show that the model was better at classifying some writing styles than the others. The average area covered by the ROC curve is 0.87 which shows that the model is not ideal for classification whereas the ROC for Random forest/Bag of words the plot is based on true and false positive rates for each class. This show that the model can best classify carroll-alice. All books have are >0.9 which is good.

Naïve Bayes:

The following figures are the ROC and confusion matrix results of the test data that were predicted by the Naïve Bayes, using a smoothing parameter of 0.01 and 0.001 respectively for TF-IDF and Bag of words.

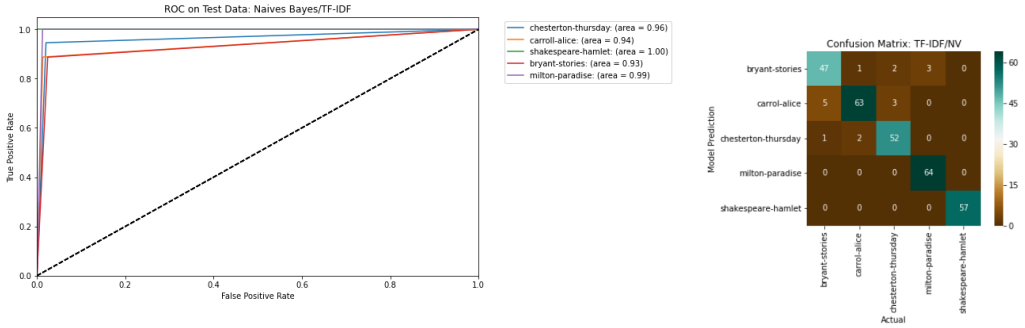


Figure 14. ROC and confusion matrix of test data for TF-IDF with Naives Bayes.

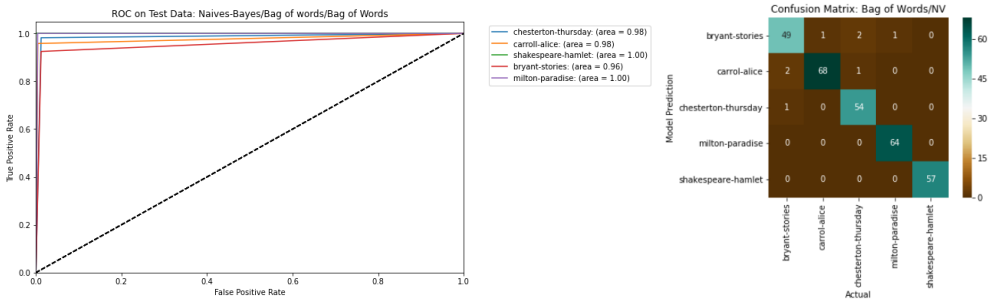


Figure 15. ROC and confusion matrix of test data for Bag of Words with Naives Bayes.

For both Naïve Bayes/TF-IDF and Naïve Bayes/Bag of words, the confusion matrix has most of the values in the diagonal, which is good. For Naïve Bayes with TF-IDF, the classification report shows an average value of 0.94, which is a pretty good model. However, Naïve Bayes with Bag of words yielded an average result of 0.97, deeming it a better preprocessing choice compared to Tf-idf.

The area under the curve is very high for both TF-IDF and bag of words, all greater than 0.95, making this model a great one for this problem. Also, the model is more biased at classifying 16th century classics such as Shakespeare-Hamlet and Milton-paradise (indicated by its' high area and therefore higher accuracy), while it could misclassify books with modern writing styles.

SVC:

The following figures are the ROC and confusion matrix results of the test data that were predicted by the SVC, using a margin penalty of 3 and squared hinge loss for both TF-IDF and Bag of words.

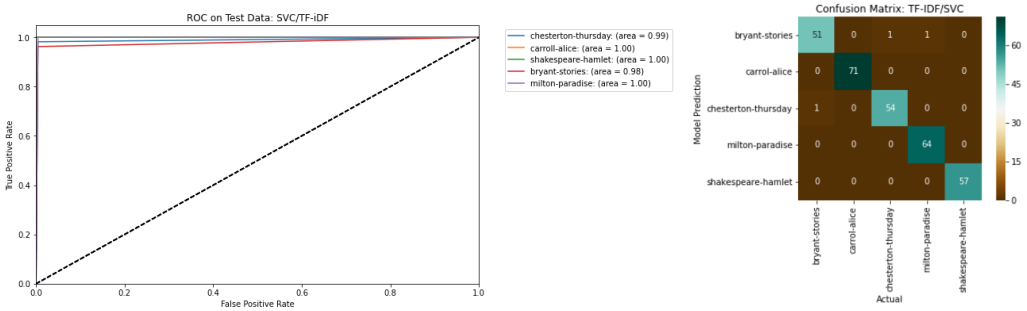


Figure 16. ROC and confusion matrix of test data for TF-IDF with SVC.

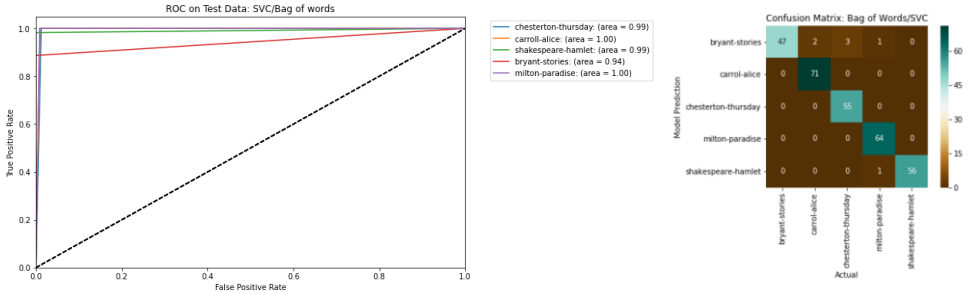


Figure 17. ROC and confusion matrix of test data for Bag of Words with SVC.

For both TF-IDF/SVC and Bag of words/SVC, the majority of the values of the confusion matrix is in the diagonal. The classification report shows an accuracy of 0.99 and F1 score of 1.0 for SVC/TF-IDF whereas for SVC/Bag of words the average accuracy is 0.98 value. These are very amazing results. The difference in preprocessing in this case did not seem to impact the classifier results greatly (1% difference). It's possible that TF-IDF may have improved the testing data slightly through getting rid of common/Non-discriminating features (words).

The ROC curve also shows how 16th century classics could be classified better than the modern style writings. The areas under the curve is high (perfect with 1.0) for Shakespeare-Hamlet and Milton-paradise, demonstrating that the model can always correctly classify 16th century classics

in the test dataset. The area under the curve is a lower for other modern books (Chesterton Thursday with 0.99 and Bryant-stories with 0.98), showing that it can occasionally misclassify modern writing styles. These results validate our hypothesis that book written in closer times share more common features and is harder to distinguish.

Summary of Test Results

During this report, we aggregated the final test accuracy score from the classification report on the table below.

Table 3. Accuracies of the different models with Bag of Words and TF-IDF features.

	SVC	Random Forests	Naïve Bayes	KNN
TF-IDF	99%	71%	94%	93%
Bag of Words	96%	96%	97%	93%

From these test accuracies, it was concluded that SVC with TFIDF is the best model for this problem, with an accuracy of 99% on the test dataset.

Error Analysis:

Error Analysis is used to determine the type of errors made by the model and identify the causes for such a misclassification, to define an approach to fix them in the future.

In this report, the error analysis is performed on the three passages that were wrongly classified in the SVC-TFIDF method. Please refer to figure 15 for the three misclassified data points.

The following figures demonstrate the word cloud of three passages, and unigram and bigram of the commonly used words in two of these passages.

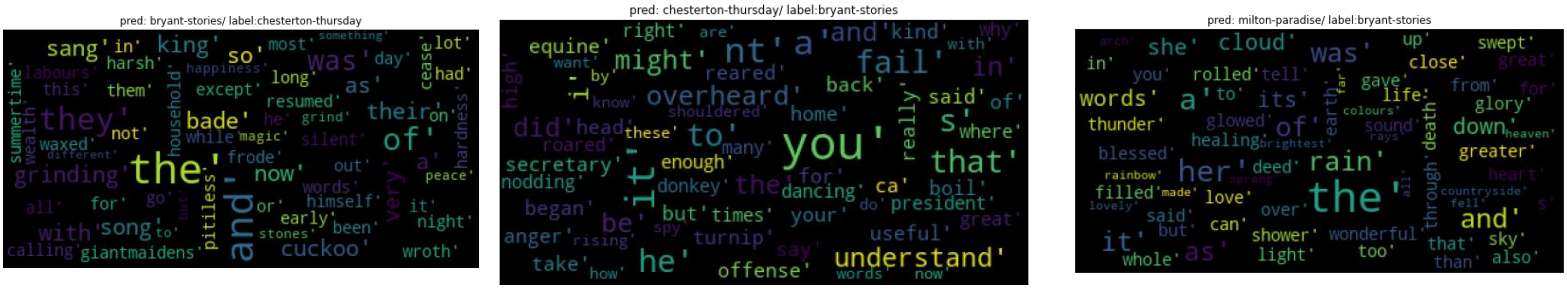


Figure 18. Word cloud of the three mislabeled passages in the final test data predicted by TF-IDF/SVC.

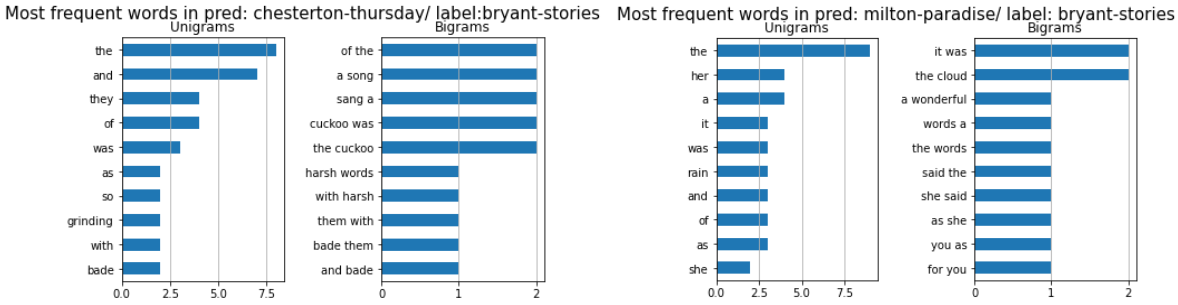


Figure 19. Unigram and Bigrams of two selective mislabeled passages in the final test data predicted by TF-IDF/SVC.

From first glance, there are multiple stop words that occurs in these word clouds such as “the”, “you” or “that”. Moreover, the unigrams reveal that the most common word in two of the passages are “the”, followed by “and” and “she” subsequentially. This means that these passages are probably heavy in stop words, making it hard to distinguish between modern book Chesterton-Thursday and Bryant stories. It is important to note that TF-IDF probably did not take away these words because they did not appear in all passages, as words such as “you” was actually “thou” in old time books such as Shakespeare.

The hypothesis that not taking away the stop words will make it easier for the classifier to distinguish between modern and century time books were backed up by our results, as there was zero misclassification of Shakespeare and Milton books into modern day books. However, such

decision is not without a tradeoff, making it difficult to distinguish between books written in similar times.

Case study on using short paragraphs for testing

In this case study, the impact of the dataset length on the accuracy was investigated. A function was first created to convert a given passages into short passages of varying lengths. A dataset with varying length (from 10 words to 100 words) is then trained using the optimized model of TF-IDF with SVC, and the accuracy on the test set was reported.

This following is the relationship between number of words in each passage of the dataset and the test set accuracy, using TF-IDF and SVC with a penalty margin of 3.

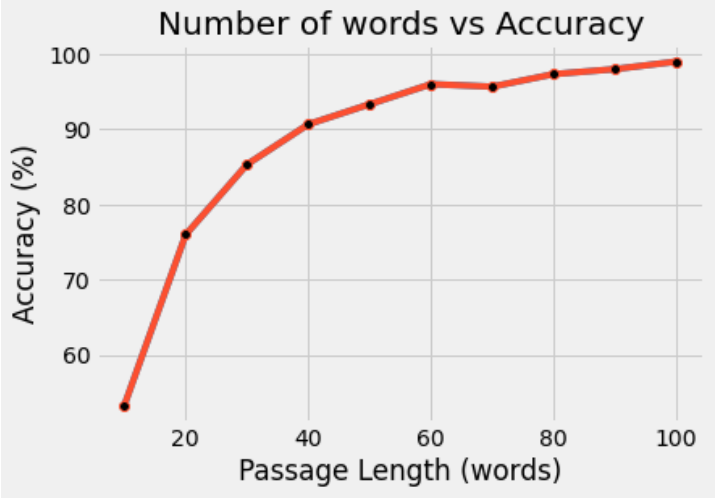


Figure 20. Testing Accuracy on dataset with varying passage length.

This figure shows that with increasing words in the passage, the model is able to the predict the corresponding book with higher accuracy. While the training accuracy decreases with less words, the model can still classify the test dataset of 40 words and greater at an accuracy of 90% and greater. Therefore, it is recommended that this classifier is used on predicting for passages with at least 40 words or more.

Conclusion

During this analysis, our group tuned and compared 3 different algorithms with 2 feature selection to solve the problem of predicting for authorship. The optimal combination was SVC with TF-IDF, with a penalty margin of 3 and using squared hinge loss. We were able to predict the author of a passage with 99% accuracy on new data. Repeating this experiment with smaller data set reveals that this classifier can still predict the passage accurately with a 90% test score or higher, as long as the number of words in the passage contains 40 words or more. We also discovered that the certain feature selection could cause class bias, as not taking out stop words caused us difficulty in distinguishing books written in similar time but has allowed for very great prediction in classifying books written in different times.

README

Problem Statement

Take five different samples of Gutenberg digital books, which are of five different genres and authors, that are semantically different. Is it possible to predict for the authorship of a book given a passage?

Abstract

Passaged were taken out of five different samples of Gutenberg digital books, with the objective to find a classifier that would yield the best prediction when given a book passage. These books was written throughout a variety of times, between the 1600s to 2000s, meaning that the words used could greatly vary. Sample processing

Goal

The goal is to determine the optimal algorithm (with the optimal parameters) that would be able to correctly classify the author of a given passage.

Hypothesis

We hypothesize that using proper feature engineering, appropriate supervised machine learning method with intensive parameter tuning, we could accurately predict the authorship of the passage.

We hypothesize that using the four different algorithms namely, SVC (Support Vector Classifier), Random Forests, Naive Bayes and K Nearest Neighbours with, different pre-processing techniques like - TF-IDF and Bag of Words and then chose our champion model having the highest accuracy.

Project Setup

1. Load Google colab - https://colab.research.google.com/

2. Create a new notebook

3. Upload the xlsx file in the file section

Note

The xlsx file that you upload will vanish if the session expires.

If uploading the file from local remember to have the file in the folder where the .py file is located.

Exact path should be mentioned along with the file name in order to upload the file correctly

Order of the cells should be followed during execution for optimzed output and avoiding the re-runs.

```
df1 = pd.read_excel(r'\books.xlsx')
```

Process

Import all the required libraries.

Step 1: Data

Function to return partitioned book as a data frame with 200 rows , each row containing 100 words

1) We use the word_tokenize package from nltk to tokenize the package

2) Using a while loop, we partition the book until the end of passage :

We grab 100 of the next token at a time

Using regex, we then calculate the number of tokens (n) out of the 100 that are not words but rather punctuations etc.

However, it is possible that the next(n) tokens also have some punctuations. So we look ahead and once again we calculate the number of tokens(n2) that are not words

We repeat this to get the index (100+n+n2+..) that would allow for 100 words in the passage.

This passage is then added onto the BookDf dataframe.

We repeat this process from the previous index counter.

3) If the dataframe has less than 200 rows, this means that there were not enough number of words in the book (<20000) for 200 partitions of 100 words each. This function will still return a data frame back but gives a warning.

4) If the total length has greater than 200 rows, we used numpy's random number generator to get200 random non-repeating numbers between 0 and max number of book rows. We then return this book back to the user.

The book partition is done into 200 random partitions of 100 words each.

We can see our partitions by calling df1

Note

Please make sure book.xlsx is in the proper location and change following path to correspond to the file location.

Using the returnLabeledBook would generate a different combination of training/testing dataset each time. This could impact the analysis results.

In order to retain consistent results each time the notebook is rerun, we are loading the data set from books.xlsx excel file.

If there is an error in loading data from this excel, then different results may not match he documentation due to stochastic randomness in grouping the training/testing set.

Data Processing and Cleansing

Getting rid of the punctuations and Spacy Lemmatization

- Our choice of Spacy library was motivated by the fact that it can perform tokenization in 0.2 milliseconds compared to nltk's 4 milliseconds.
- We are performing a lemmatization in the code. It is a process of converting many different forms to it's root word.
- The nlp pipeline created using Spacy will automatically perform the tokenization, parsing and tagging processes for us.
- We have reran this code without these preprocessing step which lead to lower accuracy.
- We are not removing stopwords during preprocessing because these features assists in adding context to the classification. Because our data contains past century books such as Shakespeare's Hamlet, the common language are different then than modern books. These stopwords could serve as key feature to help distinguish these differences. Removing them could potentially reduce the accuracy of the model in classifying such books.

```
[ ]  def return_split(passage):
       return (re.sub(r'[^\w\s]', '', str(passage).lower().strip()))
```

```
[ ]  df1['passage']=df1['passage'].apply(return_split)
```

```
[ ]  nlp=spacy.load('en_core_web_sm')
```

```
[ ]  def lemmatization(x):
       X_list=[]
       for token in nlp(x):
         lemma=token.text
         X_list.append(lemma)
       return " ".join(X_list)
```

```
     df1['passage']=df1['passage'].apply(lemmatization)
```

Then perform the Training/Testing Data Split on the book partitions dataframe using the following –

```
[ ]  x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,train_size=0.7,random_state=42) ## random state 42 for rerunning purposes
```

Step 2 : Feature Engineering

As a first choice, we are using TF-IDF to transform our X (Passages) into vectors

TF-IDF is a vectorization algorithm which is used to represent textual data in numerical vectors . It gives weights to the words depending on their frequency.

Compared to bag of words, TF-IDF also calculates the inverse document frequency, which will factor in the frequency of the word to occur in all documents. This will take out very commonly used words.

The 'Fit' is used to identify the vocabulary and frequency whereas the 'Transform' is used for conversion to a vector.

```
[ ]  vec = TfidfVectorizer(binary=True, use_idf=True)
     tfidf_train_data = vec.fit_transform(x_train)
     tfidf_test_data = vec.transform(x_test)
```

Step 3 : Model Selection

SVC Algorithm with TF-IDF

The purpose of a Linear SVC i.e Support Vector Classifier is to return a hyperplane that seperates the data into groups.

Hyper parameter tuning

We are performing hyper parameter tuning on – 'C' and 'type of loss'

Grid search and 10 fold cross validation

- We use a grid search to identify the best combination of the hyper parameters that provides the best accuracy results
- During this grid search we try each of the parameter combination on a 10-fold cross validation
- In cross validation we are splitting the training data again into 10 separate parts, and holding one part out for testing each time.

The following function concatenates the grid_search results into a more readable dataframe with the below results

test scores of CV1,CV2,CV3,CV4,CV5,CV6,CV7,CV8,CV9,CV10

train scores of CV1,CV2,CV3,CV4,CV5,CV6,CV7,CV8,CV9,CV10

average train score from 10 fold cross validation

average test score from 10 fold cross validation

The corresponding parameter combinations for these features

```
[ ] def gridsearchResultstoDf(grid_search):
        results=pd.DataFrame(columns={'mean test score','mean train score','param', 'CV1 train loss','CV1 test loss', 'CV2 train loss','CV2 test loss
        results=results[['mean test score','mean train score','param' , 'CV1 train loss','CV1 test loss', 'CV2 train loss','CV2 test loss', 'CV3 trai
        for mean_test_score,mean_train_score,params,testsplit0,testsplit1,testsplit2,testsplit3,testsplit4,testsplit5,testsplit6,testsplit7,testsplit
            z=[mean_test_score,mean_train_score,params ,trainsplit0,testsplit0,trainsplit1,testsplit1,trainsplit2,testsplit2,trainsplit3,testsplit3,t
            results.loc[len(results)]=z
        return results
```

Grid search results returned for Tf-idf and SVC algorithm.

All results are sorted by the worse to best test scores averaged from the 10-Fold Cross Validation. In some instances, multiple records can have equal (best) score. In such case, we are selecting the first best combination we come across, because any parameter combinations with yield the same test scores.

```
[ ] results=gridsearchResultstoDf(grid_search)
```

```
[ ] results.sort_values(by='mean test score')
```

Learning Curve

These are graphic representations for selecting the best parameter combination. We look to see whether increasing a parameter value will yield over fitting.

We plot a learning curve using the following code for different values of hyper parameter using the grid search to evaluate the best parameters.

```
plt.figure(figsize=(10,6))
temp_tree_cv_results.sort_values(by='C',inplace=True)
resultsSVC=temp_tree_cv_results[temp_tree_cv_results['loss']=='hinge']
plt.fill_between(resultsSVC['C'],resultsSVC['mean test score']-resultsSVC['test sdv'],resultsSVC['mean test score']+resultsSVC['test sdv'],alpha=
plt.fill_between(resultsSVC['C'],resultsSVC['mean train score']-resultsSVC['train sdv'],resultsSVC['mean train score']+resultsSVC['train sdv'],al
plt.errorbar(resultsSVC['C'],resultsSVC['mean test score'],resultsSVC['test sdv'],label='Test Score (n=10)',fmt='-o')
plt.errorbar(resultsSVC['C'],resultsSVC['mean train score'],resultsSVC['train sdv'],label='Train Score (n=10)',fmt='-o')
plt.ylabel('Accuracy')
plt.xlabel('C')
plt.legend()
plt.ylim(0,1)
plt.title('Learning Curves from Cross validation of SVC/Tf-idf (loss=hinge)')
```

Implementation of the Tf-idf and SVC algorithm with the optimal hyper parameters determined from grid search on the test data.

We train the model using the vectorized X train data and Y train. We predict the results using the vectorized X test data.

This tuned model will be used to predict the true test/holdout data.

```
[ ] clf=LinearSVC(C=3,loss='squared_hinge')
    clf.fit(tfidf_train_data,y_train)
    predictions= clf.predict(tfidf_test_data)
```

Testing Results

Confusion matrix and Classification report for the Tf-idf/SVC(C=3,loss='squared_hinge') classifier's performance

```
print(confusion_matrix(y_test,predictions))
```

```
[[51  0  1  1  0]
 [ 0 71  0  0  0]
 [ 1  0 54  0  0]
 [ 0  0  0 64  0]
 [ 0  0  0  0 57]]
```

```
print(classification_report(y_test,predictions))
```

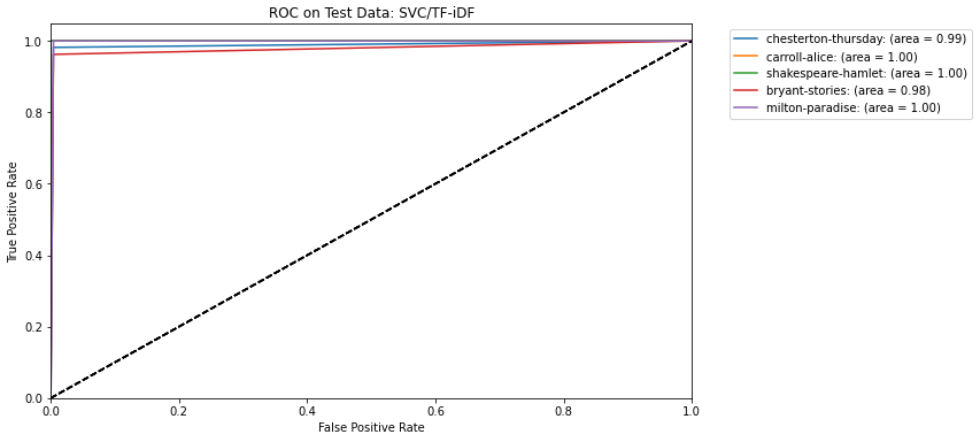
	precision	recall	f1-score	support
bryant-stories	0.98	0.96	0.97	53
carroll-alice	1.00	1.00	1.00	71
chesterton-thursday	0.98	0.98	0.98	55
milton-paradise	0.98	1.00	0.99	64
shakespeare-hamlet	1.00	1.00	1.00	57
accuracy			0.99	300
macro avg	0.99	0.99	0.99	300
weighted avg	0.99	0.99	0.99	300

ROC Curve

A Receiver Operating Characteristic curve is a graph that shows the performance of a classification model for different classes based on the true positive and false positive rates.

The areas under the curve is high (actually perfect with 1.0) for Shakespeare-Hamlet and Milton-paradise, demonstrating that the model can always correctly classify 16th century classics. The area under the curve is a lower for other modern books (chesterton-thursday with 0.99 and byrant-stories with 0,98), showing that it can occasionally misclassify modern writing styles.

We plot an ROC curve using the accuracy generated by the classification report. And we do this process for both the techniques – TF-IDF and Bag of words.



Error Analysis

Error Analysis is used to determine the type of errors made by the model and identify the causes for such a misclassification, in order to define an approach to fix them in the future.

The following is the list of misclassified labels

```
totalListoferrors.groupby(by=['predictions','labels']).count()
```

		input
predictions	labels	
bryant-stories	chesterton-thursday	1
chesterton-thursday	bryant-stories	1
milton-paradise	bryant-stories	1

We then Use the Wordclouds for visualizing the type of words used in the passages that were misclassified to identify errors

```
[ ] wordc = wordcloud.WordCloud(background_color='black', max_words=100,
                                max_font_size=35)
    wc = wordc.generate(str(nltk.tokenize.word_tokenize(totalListoferrors['input'].iloc[0])))
    fig = plt.figure(num=1,figsize=(10,6))
    plt.axis('off')
    plt.imshow(wc, cmap=None)
    plt.title('pred: bryant-stories/ label:chesterton-thursday' )
    plt.show()

    wc = wordc.generate(str(nltk.tokenize.word_tokenize(totalListoferrors['input'].iloc[1])))
    fig = plt.figure(num=1,figsize=(10,6))
    plt.axis('off')
    plt.imshow(wc, cmap=None)
    plt.title('pred: chesterton-thursday/ label:bryant-stories')
    plt.show()

    wc = wordc.generate(str(nltk.tokenize.word_tokenize(totalListoferrors['input'].iloc[2])))
    fig = plt.figure(num=1,figsize=(10,6))
    plt.axis('off')
    plt.imshow(wc, cmap=None)
    plt.title('pred: milton-paradise/ label:bryant-stories')
    plt.show()
```

We use Unigrams and Bigrams see the words that occur most frequently - alone(unigram) or together in pairs of two(bigrams)

We Repeat the same process for all combinations of the following algorithms -

- **SVC/Bag of Words**
- **Random Forest/TF-IDF**
- **Random Forest/Bag of Words**

- **Naive Bayes/TF-IDF**
- **Naive Bayes/Bag of Words**
- **KNN/TF-IDF**
- **KNN/Bag of Words**

And then compare the accuracies of all the models and chose our champion model

Manipulating the Algorithm

In addition to having the best model, we try to manipulate the values of the words in the passage in order to make it harder for the model to predict.

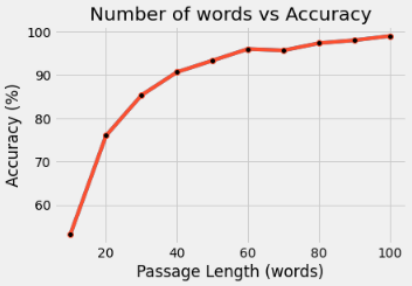
We use the following function to convert the given passages in the data set into short passages of varying lengths for testing the accuracy of the model

```
[ ]  import re
     import nltk

     def short_passages(n):
       main_set = []
       regex_pattern = re.compile(r'[A-Za-z0-9]+|^\'\w+')
       counter = 0
       result_set = []
       passage = ''
       for i in range(len(df1)):
         words = nltk.word_tokenize(df1['passage'][i])
         for each_word in words:
           if bool(re.match(regex_pattern, each_word)) == True:
             result_set.append(each_word)
             counter += 1
           else:
             result_set.append(each_word)
           if counter >= n:
             passage = " ".join(str(item) for item in result_set)
             counter = 0
             result_set = []
             main_set.append(passage)
             break
         short_passage = pd.Series(main_set)
       short_passage = short_passage.rename('passage')
       df1['short_passages'] = short_passage
       return df1
```

Plotting a graph for the Passage words vs Test Accuracy

This plot shows that with increasing words in the passage, we are able to predict the corresponding book with higher accuracy . However, this is a strong classifier, as it is able to classify the test datset of 40 words and greater at an accuracy of 90% and greater.

```
plt.style.use('fivethirtyeight')

plt.title('Number of words vs Accuracy')
plt.xlabel('Passage Length (words)')
plt.ylabel('Accuracy (%)')
plt.plot(df_words_vs_accuracy['Passage Length (words)'],df_words_vs_accuracy['Number of words vs Accuracy']*100)
plt.plot(df_words_vs_accuracy['Passage Length (words)'],df_words_vs_accuracy['Number of words vs Accuracy']*100, marker ='o', markerfacecolor='black')
plt.show()
```



Reference

1. Géron, A. (2020). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. Beijing: O'Reilly.