Data Mining : Author Classification Task

Karan Manghi
kxm9436@rit.edu

Aamir Jamal
aj1243@rit.edu

Tejas Arya
ta2763@rit.edu

Akshay Kshirsagar
ak1865@rit.edu

1. INTRODUCTION

This paper highlights how we have implemented a Data Mining task, namely Classification based on authors. We have downloaded books from 2 authors who are, L Frank Baum and R.M Ballantyne. The books are downloaded from https://www.gutenberg.org/ Gutenberg, where books are freely available to download. The sections in the paper have following flow, Initial Phase describes how we have scraped datasets from Gutenberg, then we have explained the dataset, Next phase includes how we cleaned the dataset. Later we explain how we extracted the features from the data, Techniques for Feature Representation, Why this is Data Mining Task, A real life application of our project, Briefly explained the approaches we have evaluated, the challenges we faced, followed by the methodology used in Naive Bayes and Decision Tree. After this section We have elaborated the old and new approach, followed by Observations, Conclusions and References.

2. INITIAL PHASE

The steps in scraping data include:
Scraping Data - Used Gutenberg Package in python
Installation : git clone https://github.com/c-w/Gutenberg.git
Install dependencies:
sudo apt-get install libdb++-dev
pip install .
Scrape files:
1) from gutenberg acquire import load_etext
2) from gutenberg.cleanup import strip_headers
3) text = strip_headers(load_etext(FILENAME)).strip()
4) Cleaning - Removing name of author from the files
so that the results are not biased.
Part of Speech Tagging:
Packages used - NLP, openNLP
Methods -
Maxent_Word_Token_Annotator( ) - Annotator for words
Maxent_Sent_Token_Annotator( ) - Annotator for sentences
Maxent_POS_Token_Annotator( ) - for Part of Speech tagging
Compile and apply Annotator methods to the file. Combine annotations with the file. And display POS tagging with the words. - Large Tagged Token file created.
Save to a CSV.

3. DATA

3.0.1 Data Pre-processing

After scraping the site, we got text files for all books. Our data set contains 92 and 98 files for corresponding authors. They had to be cleaned , i.e, the author name should not be present in the text. Moreover, since the book was available on the gutenberg , so they included their note and policies at the end. We removed both of these since they are not significant in classifying the author's writing traits in any ways. Also we deleted the Preface and editor's note since it was not of much use either. All of this was done manually for over 190 text files. We tried to clean it using regex as well, but it was a bit convenient and had many edge cases where our program would not work that efficiently. Now we have clean files which we can clean as required for our feature extraction part.

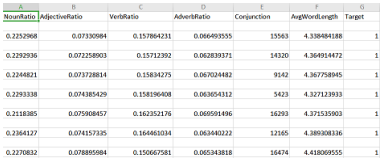
3.1 Data Extraction

3.1.1 Approach 1

The first approach that we took was to understand what particular things in the books would be able to differentiate between different authors. We discussed that a particular author might be using a particular number of verbs, adverbs or identifiers in his/her book or maybe he/she may be using more number of verbs than adverbs (and other such comparisons) in his/her book. So for this, we extracted the following from each authors' texts:
1) Adjective
2) Noun
3) Adverb
4) Verb
5) Conjunction
6) Average Word Length
For the above extraction we used the openNLP library in R. The library would take in text and return the results i.e the above types of words as well as some other types. We decided just to go with the above types because those are the words which majorly describe a sentence and the mood. The program would output a CSV file where each row would have a word and its type. Each type was further distributed in subtypes in the program. Hence as an example, we would get multiple types of adjectives and same for other types as well. The following are the sub-types of each of the types we used above:
Adjective : JJ, JJR , JJS
Noun : NN, NNS, NNP, NNPS
Adverb: RB, RBR, RBS
Verb : VB, VBD, VBG, VBN
Conjunction : CC, IN

3.2 Feature Representation

Then from the CSV file which we got, we ran a python code to get the occurrences of each of the above types in the text. We then ran that program in a loop to go through all books of a particular author and give us a csv file with the types as the columns and the count of those types in each row. Each row would correspond to a particular book. And the last column would be a 0 or 1 depending on which author that book belongs to. The CSV file would look as follows :



NounRatio	AdjectiveRatio	VerbRatio	AdverbRatio	Conjunction	AvgWordLength	Target
0.2252908	0.07330984	0.157864231	0.066493555	15563	4.338484188	1
0.2292936	0.072258903	0.15712392	0.062839371	14320	4.364914472	1
0.2244821	0.073728814	0.15834275	0.067024482	9142	4.367758945	1
0.2293338	0.074385429	0.158196408	0.063654312	5423	4.327123933	1
0.2118385	0.075908457	0.162352176	0.069591496	16293	4.371535903	1
0.2364127	0.074157335	0.164461034	0.063440222	12165	4.389308336	1
0.2270832	0.07885984	0.150667581	0.065343818	16474	4.418069555	1

Figure 1: Feature Representation

In the above figure, we can see the format of the CSV file. Each column represents a feature that we exracted and the last column represents the author and is represented by either 0 or 1.

```r
```{r}
r<-list.files("D:\\ProjectBooksBigData\\Books_Ballantyne\\")
p <- "D:\\ProjectBooksBigData\\Books_Ballantyne\\"
```

```{r}
i=0
for (variable in r) {
  temp = paste(p,variable,sep="")
  print(temp)
  f <- readLines(temp)
  f <- paste(f, collapse = " ")
  file <- as.String(f)
  word_ann <- Maxent_Word_Token_Annotator()
  sent_ann <- Maxent_Sent_Token_Annotator()
  POS <- Maxent_POS_Tag_Annotator()
  file_annotations <- annotate(file, list(sent_ann, word_ann, POS))
  file_doc <- AnnotatedPlainTextDocument(file, file_annotations)
  tags <- tagged_words(file_doc)
  p2 = "D:\\ProjectBooksBigData\\CSV_Ballantyne\\"
  val2 = paste(i,".csv",sep = "")
  val3 = paste(p2,val2,sep="")
  write.csv(tags,val3)
  i= i +1
}
```
```

Figure 2: Annotation Code

The above is a snippet from the code which is used to parse through the text and create a CSV file which has the words annotated with its type. This code is basically the heart of the feature extraction part.

4. EXPLAIN WHY DATA MINING TASK

In this project we had to first extract the features/knowledge from the data and then we had to make the model learn what features are typical for a particular class and then we got outputs on the test data as to which book corresponds to which author. Also, we used the classification approach. Because of all of the above reasons, this was a data mining task. We basically had to mine knowledge out of just plain text books and see what features would make sense and which ones wouldn't.

5. REAL LIFE APPLICATION

This project that we have does a 2 class classification. But this approach can easily be extended to a multi-class classification. We just have to add data from more authors and keep different class labels for the other authors. In real life, this approach can be used to identify authors if we are given only a paragraph from a book given that we previously know a list of all the authors.

6. APPROACH

We had multiple approaches:

6.1 Hypothetical Approach

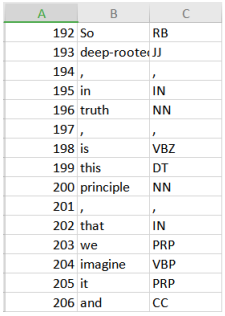
We discussed and came up with an alternate approach to solving the problem. This was just a thought and since it was too complicated, we did not go ahead and implement it. We still discuss it here. So, the idea was, a better way to find unique writing traits of an author would be to look at how the author uses words. What we mean by that is, how an author chains the different types of words together to form a sentence. This would give us insights about the writing style of the author and might be a better classification mechanism. But since this would require an understanding of Natural Language Processing and was out of the scope of this course, we just left it as a thought. An interesting thought though!

6.2 Final Approach

After the last idea, we thought hard and came up with another idea which would be possible to implement given the time constraint. The first idea we came up with highly depends on the number of words that the author has in his/her text. What if some texts of a particular author are small and the rest are very long texts. Then in that case the algorithm would get confused if it is really that author. So we thought of using ratios and averages instead of the actual number of words. We used the following:

1) Noun Ratio
2) Adjective Ratio
3) Verb Ratio
4) Adverb Ratio
5) Conjunction
6) Average word length

We used the same library i.e OpenNLP in R which we used in the first idea to get the above features. We fit the Naive Bayes and Decision Tree models and got the results.



	A	B	C
192	So	RB
193	deep-rooted	JJ
194	,	,
195	in	IN
196	truth	NN
197	,	,
198	is	VBZ
199	this	DT
200	principle	NN
201	,	,
202	that	IN
203	we	PRP
204	imagine	VBP
205	it	PRP
206	and	CC

Figure 3: Token,Tags

In the above figure we can see the format of the CSV file that we receive from the annotation code and how each word in the text corresponds to a certain type. We use these types to create our CSV file which we will feed to the Naive Bayes and the Decision Tree.

7. CHALLENGES

1) The first major challenge that we faced was to get hold of 2 prolific authors who had an abundance of freely available books. We were able to find 2 such authors on project gutenberg which helped us to comprise the dataset for training.
2) After we decided our dataset, the next problem that we faced was selecting appropriate features on the basis of which authors could be distinguished. We tried to keep the features as generic as possible and not specific to the selected two authors, so that our model could classify on any other dataset without much modification. We also had to take a few different combinations of the features in order to select the features which gave us the best possible accuracy. For example, we decided to use the average sentence length as a feature, but then realized that it was overfitting the data as we got good result for the training data but not so good when used on testing data.
3) One of the problems was installing libraries NLP and openNLP. It requires the rJava support which is not in r-cran project and has to be installed in the machine on which R is running. We installed the package rJava. Even installing the package did not solve the problem as path for rJava wasn't set correctly. While transferring data from DocumentTermMatrix to csv, the transfer containers were incompatible. The data was tokenized sentences and POS tagging which was our initial idea to be used as features to feed to the Decision Tree and Naive Bayes. This problem was overcome by converting the DocumentTermMatrix to a Matrix and that Matrix to a DataFrame.

8. METHODS:

8.1 Naive Bayes

For Naive Bayes, we read the csv file. It contained some blank lines which were removed. Next we shuffled the data so when we split the data into test and train set, it would be distributed as opposed to all the train data belonging to one class and the test data belonging to another class. Then we used the library to call the naive bayes function and predicted the output to calculate the corresponding accuracy.

8.2 Decision Tree

We first read the CSV file into a table in R. Then after reading, we realised that there were blank spaces in the CSV file and in the R table they were being represented as NA. So we omitted the NAs in the table. Then we split the data into train and test. We did a 80:20 split on the data. Next we passed the training data to the decision tree algorithm. We first used the "party" library in R to fit a decision tree. But for some reason it was making a decision tree and giving outputs in the form of continuous data and not discrete values. We spent some time trying to fix it but couldn't. The following is the plot of the tree made using "party" :

9. RESULTS

9.1 Old Approach

We have attached results for 2 methods, namely Naive Bayes and Decision Tree.

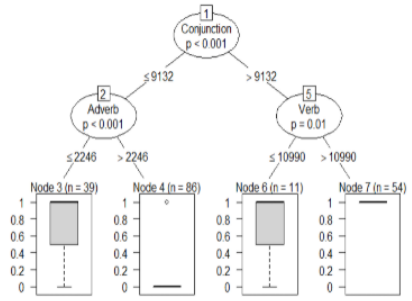


Figure 4: Using "party" library"

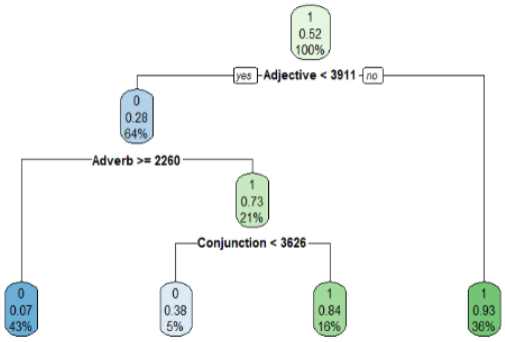


Figure 5: Using "rpart" library"

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 11  1
         1  6 12

               Accuracy : 0.7667
                 95% CI : (0.5772, 0.9007)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : 0.01905

                  Kappa : 0.5455

 Mcnemar's Test P-Value : 0.13057

              Precision : 0.9167
                 Recall : 0.6471
                     F1 : 0.7586
             Prevalence : 0.5667
         Detection Rate : 0.3667
   Detection Prevalence : 0.4000
      Balanced Accuracy : 0.7851

       'Positive' Class : 0
```

Figure 6: Naive Bayes Old Approach

In the above image we can see the results from the Naive Bayes using the old approach i.e where we are only considering the count of the types. We get an accuracy of 76% with 91% precision and 64% recall. The precision looks really good and that means we picked up a lot of relevant items.

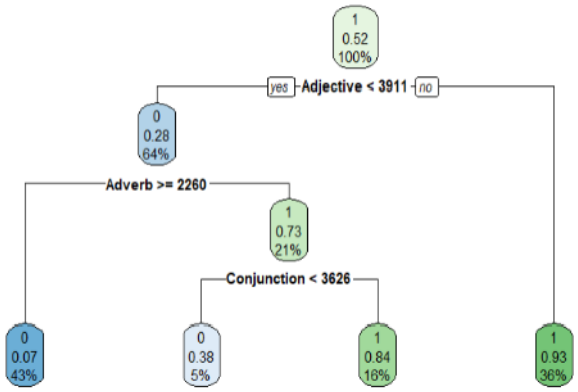


Figure 7: Decision Tree Old Approach
In the above image we see the decision tree generated using the old approach. We see that the tree first splits on Adjective, then on Adverb and finally on conjunction.

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 13  6
         1  0 11

               Accuracy : 0.8
                 95% CI : (0.6143, 0.9229)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : 0.006664

                  Kappa : 0.6137

 Mcnemar's Test P-Value : 0.041227

            Sensitivity : 1.0000
            Specificity : 0.6471
         Pos Pred Value : 0.6842
         Neg Pred Value : 1.0000
             Prevalence : 0.4333
         Detection Rate : 0.4333
   Detection Prevalence : 0.6333
      Balanced Accuracy : 0.8235

       'Positive' Class : 0
```

Figure 8: Decision Accuracy Old Approach
In the above image we can see the results from the Decision Tree using the old approach i.e where we are only considering the count of the types. We get an accuracy of 80%. Also we get a very high sensitivity which again means that we picked up a lot of relevant items.

9.2 New Approach

After the decision tree construction we used used the "predict" method in the "rpart" library to predict based on the test data. We got an output but that output was in a different format and was not comparable to the ground format. So first we converted the predicted output to a numeric format.But that's not all. The levels in this format were different. 1s were represented by 2 and 0s by 1s. So then we changed the 1s in the output to 0 and the 2s to 1s.These formats were still not comparable. So we converted both the ground truth as well as the predict function output to factors and then passed it to the confusion matrix function of the "caret" library to get the accuracy and all other factors. The accuracy would vary on each run because of the test and training split as the shuffling function was shuffling data in a different way each time. This could be solved by using a constant seed each time but we felt that it wasn't very important so we moved forward.

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 10  2
         1  3 15

               Accuracy : 0.8333
                 95% CI : (0.6528, 0.9436)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : 0.001939

                  Kappa : 0.6575

 Mcnemar's Test P-Value : 1.000000

            Precision : 0.8333
               Recall : 0.7692
                   F1 : 0.8000
            Prevalence : 0.4333
        Detection Rate : 0.3333
  Detection Prevalence : 0.4000
     Balanced Accuracy : 0.8258

       'Positive' Class : 0
```

Figure 9: Naive Bayes New Approach
In the above image we can see the results from the Naive Bayes using the new approach i.e where we are using ratio and averaged features. We get an accuracy of 83% with 83% precision and 76% recall. The precision is good enough again meaning that we picked up a lot of relevant items.

```
```{r}
print(confusionMatrix(data = prediction_numeric,reference = test_authors$Target))
```

Confusion Matrix and Statistics

          Reference
Prediction 0 1
         0 13 0
         1 4 13

               Accuracy : 0.8667
                 95% CI : (0.6928, 0.9624)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : 0.0004563

                  Kappa : 0.738

 Mcnemar's Test P-Value : 0.1336144

            Sensitivity : 0.7647
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.7647
             Prevalence : 0.5667
         Detection Rate : 0.4333
   Detection Prevalence : 0.4333
      Balanced Accuracy : 0.8824

       'Positive' Class : 0
```

Figure 10: Decision Tree Result Approach
In the above image we can see the results from the Decision Tree using the new approach i.e where we are using ratio and averaged features. We get an accuracy of 86%

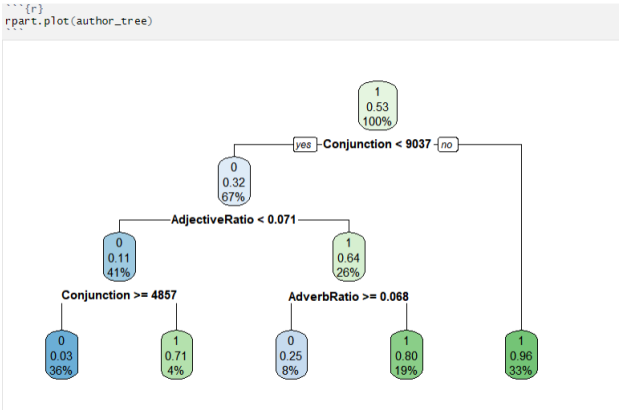


Figure 11: Decision Tree New Approach

In the above image we see the decision tree generated using the new approach. We see that the tree first splits on conjunction and then on adjective ratio and then on both conjunction and adverb ratio as children.

10. OBSERVATION

Initially when we used the features as number of verbs, number of adjectives, number of adverbs used etc, we got our average accuracy as 80 percent for decision tree and by using Naive Bayes we got the average accuracy as 76 percent. Then we realized that the feature selected (number of adjectives/nouns/verbs/adverbs) is depending on the length of the book considered as the dataset. So the same author could also have a lot of varying data in the features as per the book length. So we shifted to better features for classification and changed the absolute numbers (noun/verb/adverb/adjective) to ratio with respect to the total number of words available. This gave us features which were more author specific to the author and independent of the content length. After shifting to the new features, we got the new average accuracy as 86 percent using decision tree and 83 percent average accuracy for Naive Bayes classifier.

11. CONCLUSION

According to the above results that we have mentioned, we see that Decision Tree(86 percent) generates slightly better results than Naive Bayes(83 percent). Also, the sensitivity, specificity, precision and recall of the results is not bad and hence that way we know that the results are good. There is definitely more scope for improvement and that might happen only if we use more features based on Natural Language Processing like chaining of words, how they are related to each other, occurrence of words one after the other and so on.

12. REFERENCES

1) https:/rstudio-pubs-static.s3.amazonaws.com118341
_dacd8e7a963745eeacf25f96da52770e.html
2) https:/rstudio-pubs-static.s3.amazonaws.com118341
_dacd8e7a963745eeacf25f96da52770e.html
3) https:/rpubs.comrthemanNLP101
4) https:/smart-statistics.compart-speech-tagging-r
5) http:/martinschweinberger.dedocsarticlesPosTagR.pdf
6) https:/rpubs.comlmullennlp-chapter