**A study about using machine learning classifiers KNN and SVM  
to perform authorship attribution analysis**

DS7003 Coursework / Student No. 1720146

Chung Hoi CHIU

Abstract: In this coursework, a simple machine learning classifier, KNN, and a relatively complicated machine learning classifier developed in the 1990s’, SVM, were used to perform authorship attribution analysis on certain 19th century texts, the authors of which are mainly Victorian Era novelists. The programming language for writing code to implement the analysis is R and the feature of the texts to be used for analysis consists of common words. This coursework found that although KNN is a simple classifier, its performance only slightly lags behind that of SVM. Valuable experiences were also grained from studying the dataset and coding with R. The code and one of the two data files used for doing this coursework can be found from <https://github.com/ericchchiu/u1720146_DS7003_courseworkCodeAndData> . The code is also contained in the Appendix of this coursework.

(Warning: The encoding should be changed to UTF-8 before running the code)

1. Introduction

Authorship attribution is about using computers to decide whether a piece of text was written by one of a group of people, and, if the answer is positive, to determine who among the group was the author.

This coursework will study how to use programming language R and machine learning classifiers KNN and SVM to perform authorship attribution analysis by using 19th century English writings extracted from a dataset entitled *Victorian Era Authorship Attribution* (The dataset can be downloaded from the UCI Machine Learning Repository <http://archive.ics.uci.edu/ml/datasets/Victorian+Era+Authorship+Attribution> . Eleven 4000 word portions extracted from the novels *A Tale of Two Cities*, *Middlemarch* and *Tess of the d'Urbervilles* which were obtained from the Gutenberg book corpus <https://www.gutenberg.org/> were also used). The feature of the writings being used to perform the analysis consists of words which are common in the writings.

*Victorian Era Authorship Attribution* is the only dataset in UCI Machine Learning Repository that was specifically produced to facilitate authorship attribution analysis. Its merits and shortcomings will be discussed in Section 2 below.

The method for finding the common words is described in Section 3. Most of the common words selected are function words, which are those words that have little substantial meaning, are used mainly to glue content words to form sentences and are difficult to coin new ones.

KNN is a classifier that is simple and easy to understand. Yet it is popular in authorship attribution studies. In last year's annual worldwide authorship attribution competition PAN 2019, KNN was a commonly used supporting method (Bacciu et al., 2019). However, SVM is the most popular classifier used in the competition. Eight of the eleven competing teams used SVM as the main method (Kestemont et al., 2019). These two methods will be discussed in Section 4.

R is a language specifically designed to facilitate statistical analysis (i.e. it is not a general-purpose language such as C++ or Java). Therefore, when coding with R to perform statistical analysis, whenever possible, its operators, or the operators of its extensions, that is, its packages, which are specifically produced for performing statistical calculations, should be used. Low tier general operators and methods, such as for loops and if statements should be avoided. A valuable experience learned from coding with R for doing this coursework will be discussed in Section 5.

2. The dataset

The dataset *Victorian Era Authorship Attribution* was produced in 2017/18 to write the master thesis *Benchmarking Authorship Attribution Techniques Using Over a Thousand Books by Fifty Victorian Era Novelist* (Gungor, 2018). This dataset consists of two csv files, one for training and one for testing.

The training file has the following salient features:

a. It is large. It has 53,678 cells of texts, each of which contains 1000 words (However, the texts are untidy. Please see below).

b. It is in csv format, in which the comma is the delimiter, while all punctuation is removed from the texts.

c. Some of the proper nouns and numbers are removed.

d. All characters are in lowercase.

e. Text cells are in the first column which is entitled 'text' and author index (1 to 50, with numbers 5, 7, 31, 47 and 49 withdrawn) are in the second column, which is entitled 'author'.

The test file consists of 38,810 lines x 1 column. Each of the 38,810 cells contains 1000 words. The thesis states that the test file contains texts written by authors George Eliot (index: 5), Jack London(7), Frances Hodgson Burnett (31), Sarah Stickney Ellis (47) and Thomas Nelson Page (49) (these numbers are missing from the author column of the training file, please see the immediately above paragraph). However, this test file contains only one column. Consequently, the file has little use for machine learning studies because the authors’ index (i.e. labels) as mentioned above, 5, 7, 31,47 and 49 are not provided. Furthermore, for reasons unknown, although this test file contains the writings of only five authors, it consists of 38810 lines. In contrast, although the training file contains writings of as many as 45 authors, it consists of only 53678 lines.

Further attention should be paid to the following features of the training file:

a. The author ID stated in the table contained in p.p. 25-26 of the thesis does not match the index as stated in the author column of the training file. After conducting about a week of research, I found the indices of the training file, in general, match the names of the authors if they are rearranged in alphabetical order. Therefore, for example, George Eliot's writings were not moved to the test file, as the thesis said. They are located at lines having author index 14.

b. The texts are not tidy. The untidiness can be observed by the following comparisons:

"that we shall meet him at the assemblies, and that Mrs. Long promised to introduce him." (Jane Austen)  
that wa â all meet him at the and that mrs long promised to introduce him

"How good it was in you, my dear Mr. Bennet! But I knew I should persuade you at last. (Jane Austen)  
how good it was in you mj dear mr but i knew i should persuade ou at last

All who come near him honor him. To stand before him is like standing before a prophet of God" — Mirah ended with difficulty, her heart throbbing —"falsehoods are no use." (George Eliot)  
all who come near him honor him to stand before him is like standing before a prophet of god â ended with difficulty her heart throbbing â are no use

c. The thesis states that the texts were obtained from the huge database GDELT project. Accessing this database is free. However, the tool for accessing this database, BigQuery provided by Google, is not free. Nevertheless, Google provides free credit of 300USD to new users, which must be used in the first year after subscribing to this service. I therefore subscribed to this service and made SQL queries to retrieve books so that I could evaluate the quality of texts of the 3.5 million books published in the period 1800 to 2015 and contained in the database GDELT. The following is one of the SQL queries that I made:

#legacySQL  
SELECT BookMeta\_Date, BookMeta\_Title, BookMeta\_Creator, BookMeta\_FullText  
FROM   
(TABLE\_QUERY([gdelt-bq:internetarchivebooks], 'REGEXP\_EXTRACT(table\_id, r"(\d{4})") BETWEEN "1877" AND "1877"'))   
WHERE  
BookMeta\_Title CONTAINS 'Daniel Deronda'  
AND  
BookMeta\_Creator CONTAINS 'Eliot, George'

I confirmed that the quality of the texts is poor. Below is an example:

Pride and Prejudice, Jane Austen:  
"" But I can assure you/' she added, "" that Lizzy does not lose much by not suiting his fancy; for he is a most disagreeable, horrid man, no$ at all worth pleasing. So high and so conceited that there was no enduring him ! He walked here, and he walked there, fancying himself so very great ! Not handsome enough to dance with! 1 wish you had been there, my \* dear, to have given him one of your set downs\* I quite detest the man."" CHAP- ( 19 ) CHAPTER IV.

Notes for using BigQuery to access GDELT:   
i. Do not select the column BookMeta\_FullText too early. The charge is based on data searched, not the output. Before retrieving a book from the database, one should perform free Google searches to identify the book’s name and year of publication and then use SQL code to perform preliminary searches to make sure the book is there.  
ii. The purpose of the line #legacySQL located at the beginning of the above SQL code is to signal BigQuery that the traditional SQL code is being used.

d. Although the title of the dataset contains the term 'Victorian Era' and the title of the thesis is *Benchmarking Authorship Attribution Techniques Using Over a Thousand Books by Fifty Victorian Era Novelists*, approximately a half of the authors of the texts of the dataset are not Victorian Era novelists. For example, Charles Darwin and John Muir, the father of National Park, are not novelists, and Jane Austen is a Regency Era novelist. The dataset also contains texts of about 20 American novelists, who, according to convention, would not be classified as Victorian Era novelists even though their novels were published during the Victorian Era. Recently, I found an article entitled *Open Set Authorship Attribution Toward Demystifying Victorian Periodicals* (arXiv:1912.08259. Date of submission to arXiv: 17 December 2019) of which the producer of the dataset *Victorian Era Authorship Attribution*, Abdulmecit Gungor, is a co-author. All 46 authors of a corpus mentioned in the article are Victorian Era novelists.

3. The feature used

When performing authorship attribution studies, one should select one or more features of the texts to study. Before the computer era, scholars usually used the method of finding whether unusual features of an author's writings also occurred in the target text. Examples of such unusual features include rare words and unusual sentence structures. This method has two shortcomings. The first is that if the target text does not contain unusual features, then the study cannot proceed further. The second is that, if such unusual features of an author attract attention of scholars, they will similarly attract the attention of admirers, forgers and imitators of the author.

Therefore, authorship attribution studies should focus on the common features of texts. Currently, the most popular features are character n-gram and word n-gram (in that order). Last year, all eleven of the teams participated in PAN 2019 used these features (Kestemont et al., 2019). The character n-gram approach to the text ' rare words ' would produce the following features: | r| |ra| |ar| |re| |e | | w| |wo| |or| |rd| |ds| |s | if n is two, which is also termed bi-gram. Word bi-gram would produce from the text 'on an evening in the latter part of may' the following features: |on an| |an evening| |evening in| |in the| |the latter| |latter part| |part of| |of may| .

However, I will not use n-gram features to do this coursework, because the texts of the dataset are quite untidy, and it will lead to many meaningless n-gram elements.

I will use common words, most of which will be function words. Function words include conjunctions, determiners, particles, prepositions, etc. They are closed-class words. It means new function words are not easy to invent. When isolated, each function word does not carry substantial meaning. They work mainly as glue to join content words to form sentences.

Although function words carry little substantial meanings, they occur very frequently in speaking and writing. On the other hand, people do not pay attention to these words and tend to use them subconsciously. For example, if people are asked to count the number of 'f's in the following sentence and only count once:

Finished files are the result of years of scientific study combined with the experience of many years.  
(obtained from: <https://www.smart-jokes.org/count-number-f-sentence.html> )

Most people cannot get the right answer because they always pay little attention to function words (the answer is six).

Function words are also included in the stop word lists of search engines. They are called stop words because the engine would stop searching with these words since they carry little substantial meaning. In most branches of text mining, such as document summarization, sentiment analysis and spam email detection, stop words are filtered out before analysis can proceed.

However, function words are a treasure to authorship attribution studies because they are used so often and used subconsciously. Their use betrays one aspect of a person's style, and this aspect is not conspicuous and is difficult to imitate.

Nevertheless, I will not compile a function word list or stop word list and pick up those words from the data for analysis, because the border between function words and content words, like other linguistic features, is not clear cut. For example, the prepositions 'above', 'between' and 'through' have a relatively high degree of substantial meaning. The modal auxiliary verbs such as 'can', ‘could’ and 'shall', in addition to grammatical functions, also have the function of indicating the speaker's moods. Therefore, I will use the following objective method to pick up words for analysis: first find words that appear in texts of every candidate authors' texts and then among them pick up those words which occur at least 0.05% times the number of total words of the candidate authors' texts.

4. The classifiers KNN and SVM

As mentioned in Section 1, in authorship attribution studies, KNN is a simple and still popular classifier and SVM is the most popular classifier. General discussions of these two methods can be found in James, et al. 2013 (KNN at p.p. 39-42 and SVM at p.p. 337-368). Deeper discussions of these two methods can be found in Hastie, et al., 2009 (KNN p.p. 14-18 and SVM p.p. 417-458). However, I will only discuss certain features of these two methods which are interesting or require attention when the two classifiers are used to perform authorship attribution studies:

a. A simple authorship attribution example can be used to provide a clear explanation of KNN:

Below are four texts each of which consists of the same three distinct words (how are you) and nine tokens in different combinations (just three distinct words are used because it is difficult to imagine beyond three dimensions). These four texts are represented by the following informally represented vectors:

text A1: 2[how]+3[are]+4[you] text A2: 4[how]+4[are]+1[you]  
text B1: 4[how]+3[are]+4[you] text B2: 3[how]+3[are]+3[you]

Author A wrote texts A1 and A2 and author B wrote texts B1 and B2.

Using KNN to find the likely author among authors A and B of the following text T:

2[how]+3[are]+4[you]

one can first find the distances between the target text T and each of the four texts with their authors known. The type of distance used is usually Euclidean distance. The distances can be found by using a vector distance calculator (<https://www.calculatorsoup.com/calculators/geometry-solids/distance-two-points.php>):

Euclidean distance between T and A1: 0 / A2: 3.741657 / B1: 2 / B2: 1.414241

If the K in KNN (K-nearest neighbours) is 1, only the nearest data point will be used (i.e. text A1) to classify and therefore Author A will be chosen as the likely author. If K is 3, the nearest three data points will be used (i.e. texts A1, B1 and B2) and therefore Author B will be chosen.

In real authorship attribution studies, each text/document will generate a data point/vector and each element of the feature used (for example, function word 'the') will form a dimension of the vectors. The number of dimensions is usually in the magnitude of hundreds or thousands.

The above very simple example shows that if the number of K is too small, the model will lean to the variance side of the bias-variance trade-off. This means that the model derived from the training set will have a significant change even if there is only a slight change in the data points close to that representing the target text. On the other side of the spectrum, if the number of K is too large, the model will lean to the bias side of the bias-variance trade-off. This means that the model derived from the training set will be stable, but, since there are too many unsuitable elements (noise) of the training set was used to develop the model, the model will be suitable only to describe the training set itself. In another jargon, the model so developed will be overfitting. We need to tune the value of K to render the model not too variable and not too biased.

b. Some special features of SVM:

i. KNN uses all vectors formed from the training set data (in authorship attribution, it means texts of all candidate authors) simultaneously to perform a multi-class classification on a target text. In contrary, in SVM, each time, only vectors of a pair of classes (in authorship attribution, it means a pair of the candidate authors) are used to form a hyperplane to divide the n-dimensional space into two parts and then perform a binary classification on a target text. This means that, in a SVM operation, when the one-vs-one approach is adopted (another approach is one-vs-all), and there are i classes of data (in authorship attribution, it means i number of candidate authors), a target text will be classified i(i-1)/2 times to decide to which class (which candidate author) this target text should be attributed.

ii. SV, support vector, indicates that not all vectors produced with the training data will be used to define the hyperplane, only those that are best to be used would be used, which means that changing of those vectors that are not support vectors will not change the hyperplane. In this aspect, therefore, SV of SVM is like NN (nearest neighbours) of KNN. We need to tune a C value (i.e. cost) of an SVM model. The lower the C value, the narrower the margins on both sides of the hyperplane those vectors inside which will be used as the support vectors. This means that the model will lean more on the variance side of the variance-bias trade-off. On the other hand, the higher the C value will cause the model to lean more on the bias side of the variance-bias trade-off.

iii. SVM uses various kernel functions to transform vector values to a designated form before they are used to form a hyperplane. However, in authorship attribution, the linear kernel (i.e. no transformation) is invariably used. The following passage will explain this phenomenon:

In this data set, there are a very large number of features relative to the number of observations. This suggests that we should use a linear kernel, because the additional flexibility that will result from using a polynomial or radial kernel is unnecessary.  
(James, et. al, 2013, p. 367)

In authorship attribution, the number of features extracted from texts are always higher than the number of texts (observations). Furthermore, a document feature matrix / dataframe is also sparse (i.e. full of zeros. Please see the picture contained in subsection 5.b.vii below).

(Other features about implementing SVM, such as tuning and cross-validation, will be discussed in the next section.)

d. Why KNN and SVM are called non-parametric machine learning algorithm

Non-parametric machine learning algorithms do not make strong assumptions on the distribution of the data that they study. In contrast, parametric machine learning algorithms such as linear regression and Naive Bayes make strong assumptions.

e. Why KNN is said a lazy machine learning algorithm (Please see next section).

5. The R code for doing this coursework

Four R code files were produced for doing this coursework and are appended to the end of this coursework. (Warning!) To input these files to the console of an RStudio and run them from it, after a file was input, one should click File -> Reopen with Encoding and change the encoding to UTF-8 if the default encoding is not UTF-8 (If the RStudio is run on a Windows PC, it is likely that the default encoding is ISO 8859-1). Use UTF-8 is because the code contains a non-ASCII character â which is the name of a column of a dataframe and the column must be deleted.

These four files are briefly described as follows:

a. code 1 (JaneAustenAndJohnMuirTokenizers3Hrs.r)

This code needs more than three hours to run!

I have very little knowledge about R and this is the first R code that I wrote for this coursework. I have some experience in Java and Scala. Therefore, I tried to write code in Java style and avoid using packages of R. I used the package 'tokenizers' only to convert every token into an independent character vector and to use the package 'class' to perform KNN calculations. I used the general for loop to traverse the tokens and interim dataframes produced while the programme is running, and the if operator to perform compare, select or discard functions. The aim of these manoeuvres is to construct a dataframe with each text represented by a row and each common word by a column (Please see Section 3 above). After struggling for about 10 days, I produced this code. It is very difficult to understand. Even worse, it needs more than three hours to run (code 2, which will be discussed below and which performs the same tasks as this code, takes less than 50 seconds to run of which about 20 seconds are for inputting data).

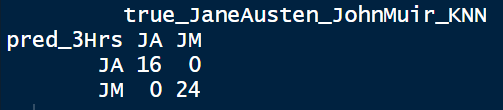
This experience gave a solid understanding of the advice that one should endeavour to code in the declarative style (to provide 'What to do' instructions) and avoid the imperative style (to provide 'How to do' instructions) and choose the methods and operators accordingly. The operator DocumentTermMatrix of the tm package of R is a typical declarative type operator: to produce a matrix with documents forming the rows and terms forming the columns (How to do it is up to the package tm and R to decide). The following snippet of code extracted from my code 1 is a typical imperative style code:

for (i in 5: nrow(y)) {  
 if (i%%4 == 1) {  
 df <- y[i,]  
 }   
 else if (i%%4 != 0) {  
 df <- rbind(df, y[i,])  
 }  
...

The following is an interpretation of the above code: For row number 5 to row number of the last row of the dataframe y, if the remainder of dividing the row number with 4 is 1, form a dataframe df with the row of the dataframe y; if not but if the remainder of dividing the row number with 4 is not 0, append the row of the dataframe y to the dataframe df...

The declarative style code is more readable and more effective. The methods and arrangements used behind the scenes by declarative type operators such as 'DocumentTermMatrix' should be advanced and effective.

The output obtained from running this 3-hour+ code is shown below:

  
KNN all correct

b. code 2 (JaneAustenAndJohnMuirTmDtm.r)

This code (and code 1 above) trains a KNN model with the designated features (i.e. number of occurrences of common words) of training texts written by Jane Austen and John Muir and uses the model to ascertain who among the two authors wrote the test texts. 400,000 (400 cells x 1000) words were chosen from each of the two authors’ writings.

This is the first task of this coursework. Therefore, only two authors with very different writing styles were chosen. Jane Austen is a Regency Era novelist and John Muir, the father of the American National Parks, is a naturalist.

A brief description of what this code does is shown below:

i. Input data from the file Gungor\_2018\_VictorianAuthorAttribution\_data-train.csv (259.361MB, 53679 rows x 2 columns (text, author). Please see Section 2 for more details of this dataset).

ii. Form two corpa with texts written by Jane Austen (JA) and John Muir (JM) respectively.

iii. Form two document term matrices with texts represented by rows and common words as the columns. The number in a cell represents the number of occurrences of the common word in a text.

An important note: the default setting of the DocumentTermMatrix operator makes it ignores tokens with fewer than 3 characters. Therefore, words such as 'as', 'is' and 'we' are ignored by this operator. However, authorship attribution treasures these words. Therefore, the following control statement should be used to regulate DocumentTermMatrix:

control= list(wordLengths = c(1, Inf))

(please see code 2 for how to do it)

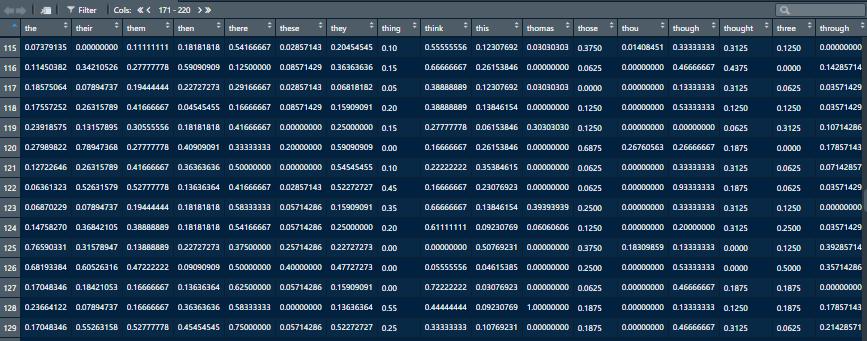
iv. Retain columns representing words which appear at least once in texts of JA and also at least once in texts of JM. The total number of occurrences of the word is not less than 0.05% of the total number of tokens of the texts of JA and JM.

v. Further delete some columns of single characters and the character â. They are the products of OCR scanning errors. For example, some OCRs convert non-ASCII characters such as em-dash '—' to â.

vi. Aggregate every 4 rows to one and add the numbers accordingly, which means every newly formed row representing a text of 4000 tokens long. JA and JM each has 100 rows.

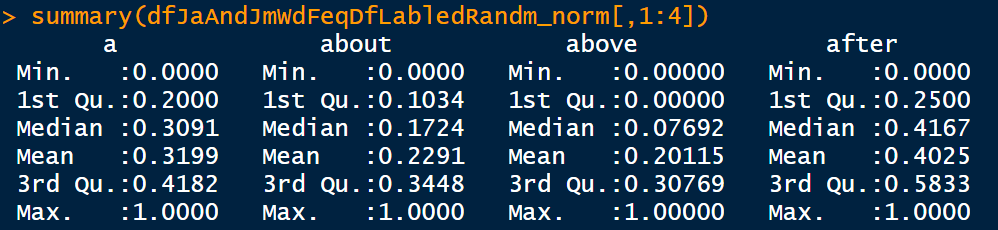
vii. Label, randomise and normalise the dataframe:

Part of the resulting dataframe is shown below:



(each 4000-token long text is represented by a row and each common word a column.)

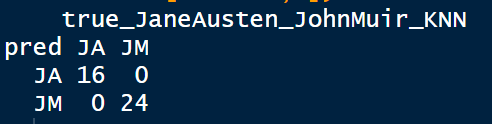
A summary of the first four columns of the normalised dataframe is shown below:



Note: the length of every text here is the same (4 x 1000 = 4000 tokens). If lengths of texts vary, one should convert the number of occurrences of each word in a text to 'term frequency' (This is a special term. It equals (number of occurrences of a word) / (length of a text)) first before normalisation over columns), or use Cosine distance instead of Euclidean distance to perform calculation, or use tf-idf (term frequency–inverse document frequency) weighting.

viii. Divide the rows into training set (80%, 160 rows) and test set (20%, 40 rows)

ix. Use the KNN operator of the class package to perform KNN calculations on the data and the result is shown below:

  
KNN all correct

It is a common practice that initially the value of K should be set at the square root of the number of datapoints (which here means the number of texts) of the training set. Therefore, initial K = sqrt(160) = 12

(SVM will be used and discussed in the next subsection)

c. code 3 (ThreeVictorianEraFemlNovlstsTmDtm.r)

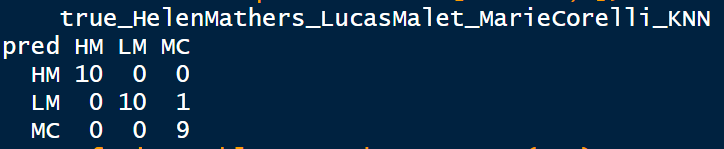
The nature of this code is similar to code 2 above except for the following:

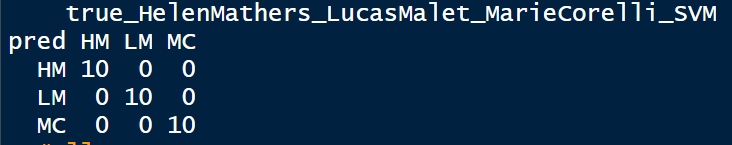
i. To make the tasks more challenging, instead of just choosing two authors as in code 1 and code 2, in this code, three authors with similar writing style were chosen. They are female popular novelists Helen Mathers, Lucas Malet and Marie Corelli. All were born in the 1850s'.

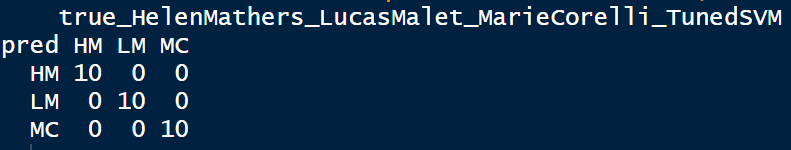
ii. The number of words chosen from each author is reduced from 400,000 (400 cells x 1000) to 200,000 (200 cells x 1000)

iii. In addition to KNN, SVM is used for classification

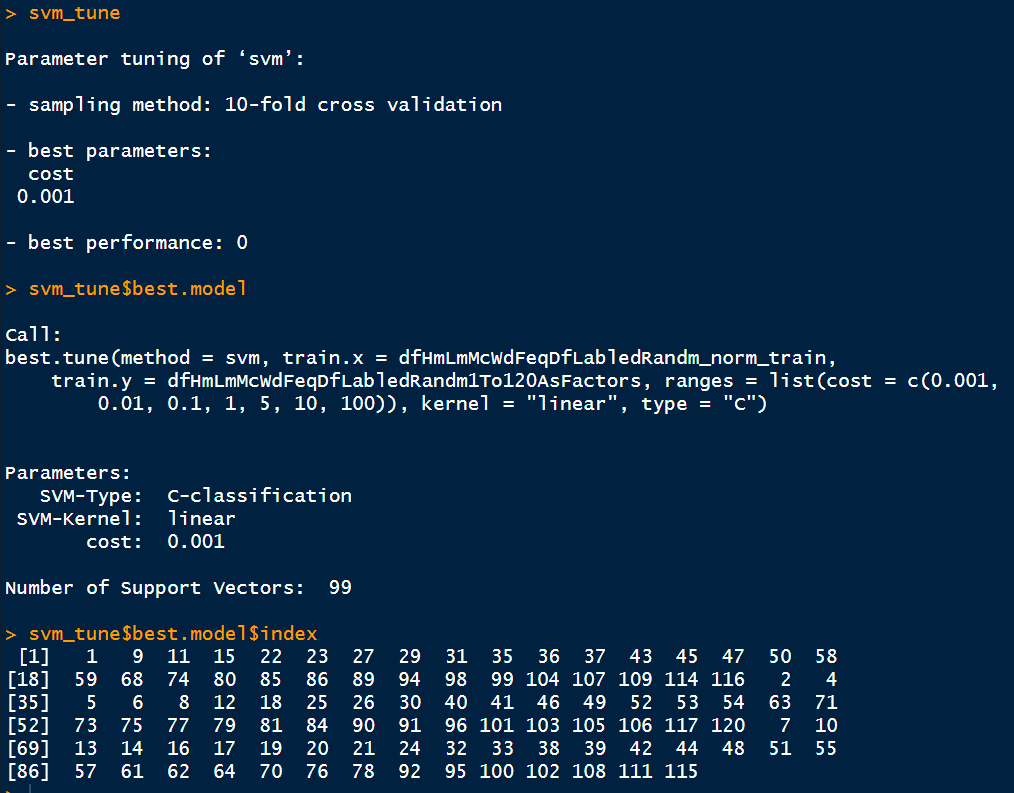
Cross tables produced by KNN and SVM (tune automatically and tune manually) are shown below:

  
KNN correct rate = 29/30 = 96.7%

  
SVM (tuned automatically) all correct

  
SVM (tuned manually) all correct

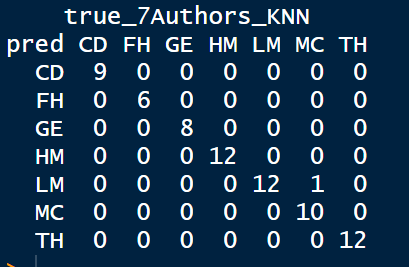
iv. The following picture shows information concerning manual tuning of the SVM

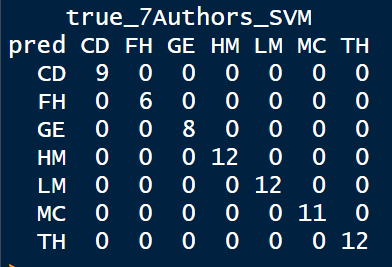
  
  
The above picture shows that tuning was performed by a 10-fold cross validation, which means that the training data is divided into 10 portions, after which the operation is run 10 times. During each time, a portion of the 10 portions is used as the validation data (similar to test data) and then the best one is selected out of the results. The cost is selected to be 0.001 (the cost can be any number from 0 to infinity since SVM-Type is defined as a C-classification type). With the cost set at 0.001, 99 vectors (out of the total 120 vectors) were selected to be support vectors and the row indices of the 99 selected vectors are shown.

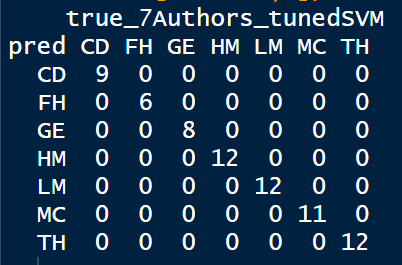
d. code 4 (DickensEliotHardyAndFourOthersTmDtm200LinesEachAndEliotImmBlr80End80GutenbergMiddlemarch5Portions.r)

i. To make the tasks even more challenging, I added four more authors' writings to the three authors' I used in code 3, which means that the writings of seven authors were used: Charles Dickens, Fergue Hume, George Eliot, Helen Mathers, Lucas Malet, Marie Corelli and Thomas Hardy. Initially, I further reduced the words per author from 200,000 to 75,000 (75 cells x 1000) but the results became very poor. I therefore increase the number of words for each author to 200,000.

ii. Cross tables produced by KNN and SVM (both tuned automatically and manually) are shown below:

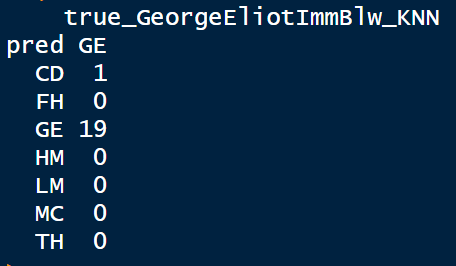
   
KNN correct rate = 69/70 = 98.6%

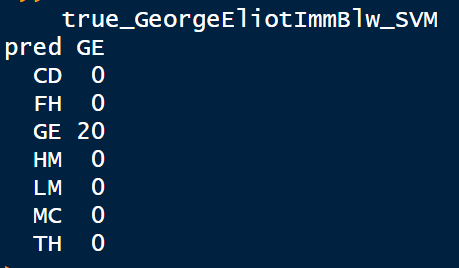
  
SVM (tuned automatically) all correct

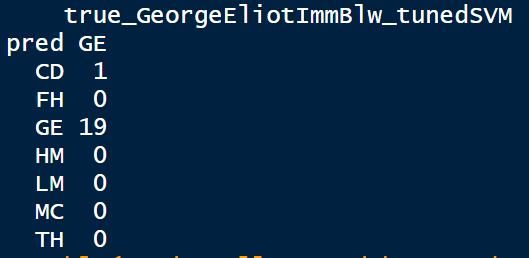
  
SVM (tuned manually) all correct

iii. In previous studies, test data were obtained from the same dataset from which the training data were produced (80% for training and 20% for testing). To make the tasks more challenging, I used data obtained from outside the original dataset to test the models. In the original dataset, the following 200 lines of George Eliot's writings were used: 13670 to 13869. Therefore, I used lines 13870 to 13949 (immediate below line 13869) and lines 16286 to 16365 (at the end of George Eliot's portion) to test. The results are as follows:

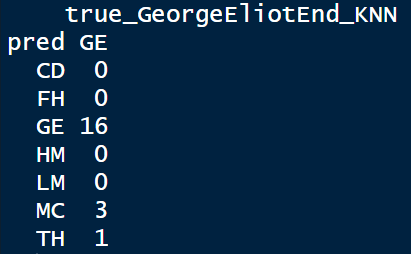
Results from using lines immediately below those lines used to form the training set to test (13870 to 13949):

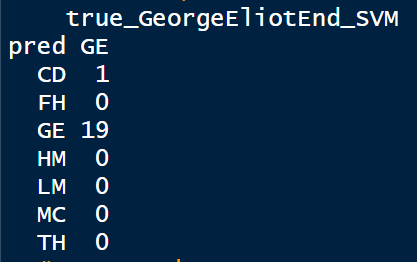
  
KNN correct rate = 19/20 = 95%

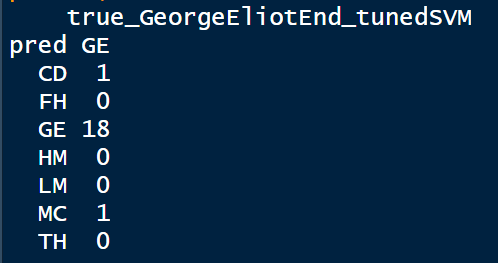
  
SVM (tuned automatically) all correct

  
SVM (tuned manually) correct rate = 19/20 = 95%

Results from using lines at end of George Eliot's portion to test (16286 to 16365):

  
KNN correct rate 4/20 = 80%

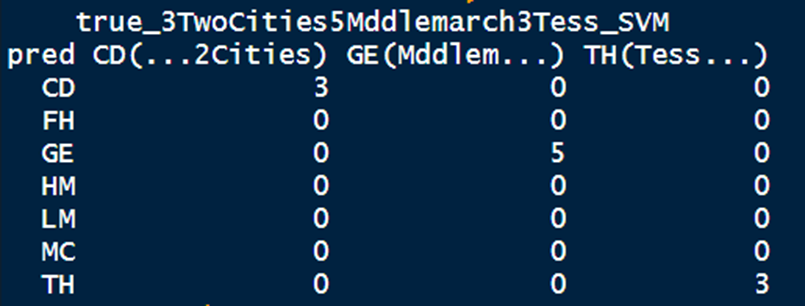
  
SVM (tuned automatically) correct rate = 19/20 = 95%

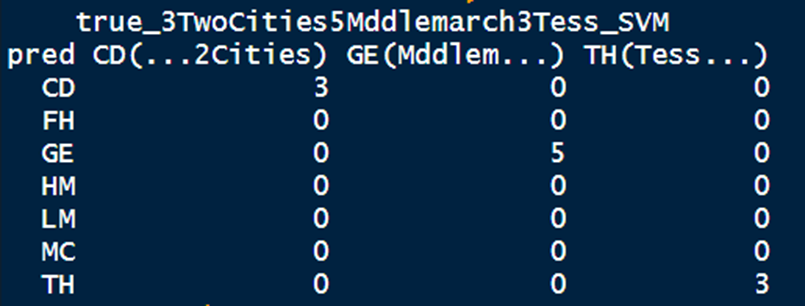
  
SVM (tuned manually) correct rate = 18/20 = 90%

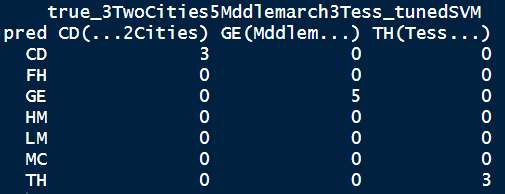
Predictions are less accurate but, except on one occasion (80%), correct rates are still high (equal or higher than 90%).

However, the above results show that the accuracy of automatic tuning is slightly higher than that of manual tuning. I have spent more than ten hours trying to improve the accuracy of manual tuning without success. Further delving into SVM is required.

[iv. Finally, it was found that although George Eliot’s masterpiece *Middlemarch* (300,000+ words) is contained in the data file, it has not been used in the experiments discussed above. Therefore, a copy of this novel was obtained from another source, that is, the famous Gutenberg book corpus (<https://www.gutenberg.org/>) and from which 5 portions of words (4000 words each with all numbers and punctuation and most person and place names deleted and all words were changed to lowercase) were extracted (GutenbergMiddlemarch\_5PortionsEach4000Words.csv). These five portions were then checked with the KNN and SVM models developed above. It was found that both the KNN and SVM models can recognise correctly that all the five portions were written by George Eliot. The results are shown below:

  
KNN all correct

  
SVM (tuned automatically) all correct

  
SVM (tuned manually) all correct

v. KNN is said to be a lazy machine learning algorithm and SVM is not. The following snippets of code extracted from code 4 provide the explanation:

KNN:  
Using KNN to predict which of the seven authors wrote each of the test set texts:

whichOfThe7\_pred <- knn(dfAll7WdFeqDfLabledRandm\_norm\_train, dfAll7WdFeqDfLabledRandm\_norm\_test, dfAll7WdFeqDfLabledRandm[1:280,1], k= 18)

Checking whether KNN can recognise all the 20 test texts were written by George Eliot:

all20GeEnd\_knn\_pred <- knn(dfAll7WdFeqDfLabledRandm\_norm\_train, dfGeEndWdFeqDf\_normNotReal, dfAll7WdFeqDfLabledRandm[1:280,1], k= 18)

SVM:  
Using training set data to produce an SVM model for prediction:

whichOfThe7\_svm\_model <- svm(dfAll7WdFeqDfLabledRandm\_norm\_train, dfAll7WdFeqDfLabledRandm[1:280,1], type = 'C')

The model whichOfThe7\_svm\_model is used to predict which of the seven authors wrote each of the test set texts:

pred <- predict(whichOfThe7\_svm\_model, dfAll7WdFeqDfLabledRandm\_norm\_test)

The model whichOfThe7\_svm\_model is used again to check whether it can recognise that all the 20 test texts were written by George Eliot:

pred\_svm\_GeEnd <- predict(whichOfThe7\_svm\_model, dfGeEndWdFeqDf\_normNotReal)

From observing the code snippets above, it is found that KNN will not use training set data in advance to form a general and reusable model for testing, which is the reason why it is called a lazy machine learning algorithm.

6. Conclusions and further research

This coursework shows classifiers KNN and SVM and the feature of common words are useful tools for performing authorship attribution analysis. Although KNN is a much simpler classifier when compared to SVM, its performance does not too lag behind that of SVM. However, in authorship attribution analysis, methods and features are not mutually exclusive. They can joint together to provide a more convincing conclusion.

I found that all current popular classifiers used in authorship attribution studies, including KNN and SVM, focus on the frequency of occurrence of a certain feature, such as character n-gram, word n-gram, common words and function words, and that none of them pay attention to the distribution patterns of these features in texts. However, in the first book in this field, the *Inference and Disputed Authorships: The Federalist* (Mosteller and Wallace, 1964), two distributions, the Poisson distribution and the negative binomial distribution, were used to study the disputed texts, and the latter was said to provide more satisfactory results than the other. The difference between these two distributions is that negative binomial distribution requires one more parameter to deal with the overdispersion phenomenon. Perhaps authorship attribution practitioners should pay more attention to the distribution of features, not just focus on their occurrence frequencies.

References

Bacciu, A. et al. (2019) Cross-Domain authorship attribution combining instance-based and profile-Based features: Notebook for PAN at CLEF 2019 Available at: https://www.researchgate.net/publication/334591954\_Cross-Domain\_Authorship\_Attribution\_Combining\_Instance-Based\_and\_Profile-Based\_Features\_Notebook\_for\_PAN\_at\_CLEF\_2019 (Accessed: 16 May 2020).

Gungor A. (2018) Benchmarking authorship attribution techniques using over a thousand books by fifty Victorian Era novelists, Unpublished MSc Thesis, Purdue University, Available at: https://scholarworks.iupui.edu/handle/1805/15938 (Accessed: 16 May 2020).

Hastie, T. et al. (2009) The elements of statistical learning: data mining, inference, and prediction. 2nd edn. New York: Springer.

James, G. et al. (2013) An introduction to statistical Learning: with application in R. New York: Springer.

Kestemont, M. et al. (2019) Overview of the cross-domain authorship attribution task at PAN 2019, Available at: https://www.semanticscholar.org/paper/Overview-of-the-Cross-domain-Authorship-Attribution-Kestemont-Stamatatos/9b78bdd05fdbbedde33db045a3367c08b81339c8 (Accessed: 16 May 2020).

Mosteller, F. and Wallace D. L. (1964) Inference and disputed authorship: The Federalist. Reading, MA, USA: Addison-Welsey.

Appendix

##code 1 (JaneAustenAndJohnMuirTokenizers3Hrs.r)

#this code take 3 hours to run!

#if open this code to RStudio. After opened it,

#select File -> Reopen with Encoding -> UTF-8 (if the default encoding is not UTF-8 (Windows: usually ISO 8859-1)

#Reason: need to recognised the non-ascii symbol â (it should be an a with a caret!)

#set working file

setwd(dirname(file.choose()))

getwd()

#input data

dfVictorianEraAA <- read.table('Gungor\_2018\_VictorianAuthorAttribution\_data-train.csv', header = TRUE, sep = (','))

#produce two dataframes.

#one for Jane Austen's texts (JA) and one for John Muir's texts (JM)

dfJaneAusten26674\_27073 <- dfVictorianEraAA[26674:27073,]

dfJaneAusten26674\_27073$textNo <- rep(1:100, each = 4)

dfJaneAusten26674\_27073 <- dfJaneAusten26674\_27073[c('textNo', 'text')]

dfJohnMuir31420\_31819 <- dfVictorianEraAA[31420:31819,]

dfJohnMuir31420\_31819$textNo <- rep(1:100, each = 4)

dfJohnMuir31420\_31819 <- dfJohnMuir31420\_31819[c('textNo', 'text')]

#Form a unique word list of the words in texts of JA and JM

library(tokenizers)

combineTwoLists <- function (list1, list2) {

n <- c()

for(x in list1){n<-c(n,x)}

for(x in list2){n<-c(n,x)}

return(n)

}

listOfWordsJaneAusten26674\_27073 <- tokenize\_words(paste0(dfJaneAusten26674\_27073[1,2]))

for (i in 2:400) {

listOfWordsJaneAusten26674\_27073 <- combineTwoLists (listOfWordsJaneAusten26674\_27073, tokenize\_words(paste0(dfJaneAusten26674\_27073[i,2])))

listOfWordsJaneAusten26674\_27073 <- unique(listOfWordsJaneAusten26674\_27073)

}

listOfWordsJohnMuir31420\_31819 <- tokenize\_words(paste0(dfJohnMuir31420\_31819[1,2]))

for (i in 2:400) {

listOfWordsJohnMuir31420\_31819 <- combineTwoLists (listOfWordsJohnMuir31420\_31819, tokenize\_words(paste0(dfJohnMuir31420\_31819[i,2])))

listOfWordsJohnMuir31420\_31819 <- unique(listOfWordsJohnMuir31420\_31819)

}

listOfWordsAppearingInBothJAAndJM <- Reduce(intersect, list(listOfWordsJaneAusten26674\_27073,listOfWordsJohnMuir31420\_31819))

#a function for forming a dataframe with each text a row and a list of unique words the columns

A = function(x, y) {

for(k in y)

df <- data.frame('a' = 0)

for(i in x) {

num = 0

for(val in 1: nrow(y)) {

tokenized = tokenize\_words(paste0(y[val,2]))

tokenized = unlist(tokenized, use.names=FALSE)

num = num + length(grep(paste('\\<',i,'\\>', sep =''), tokenized))

}

df[paste(i)] <- c(num)

}

return(df)

}

#apply the A function to form a dataframe for JA's texts

#the below line take more than an hour to run!

JA\_NoOfWdsInJAnJMUniqWdLst = A(listOfWordsAppearingInBothJAAndJM, dfJaneAusten26674\_27073)

#apply the A function to form a dataframe for JA's texts

#the below line take more than an hour to run!

JM\_NoOfWdsInJAnJMUniqWdLst = A(listOfWordsAppearingInBothJAAndJM, dfJohnMuir31420\_31819)

#combine two dataframes and

#retain those columns occurrence of the word equal or larger than 400 times

JAnJMJoin\_NoOfWdsInJAnJMUniqWdLst = rbind(JA\_NoOfWdsInJAnJMUniqWdLst, JM\_NoOfWdsInJAnJMUniqWdLst)

JaJmTtl400OrMore = JAnJMJoin\_NoOfWdsInJAnJMUniqWdLst[, colSums(JAnJMJoin\_NoOfWdsInJAnJMUniqWdLst) >=400]

JaJmTtl400OrMoreByAlpha = JaJmTtl400OrMore[,order(names(JaJmTtl400OrMore))]

#?JaJmTtl400OrMoreByAlpha = JaJmTtl400OrMoreByAlpha[-c(2)] # deleting a head

JaJmTtl400OrMoreByAlphaHeader = colnames(JaJmTtl400OrMoreByAlpha) # a list of 231 words

#formed JA and JM word dataframe (200 x 220)

JaJmTtl400OrMoreByAlphaHeader = colnames(JaJmTtl400OrMoreByAlpha)

remove <- c('â', 'e', 'f', 'h', 'j', 'l', 'n', 'o', 'r', 'u', 'v')

JaJmTtl400OrMoreByAlphaHeader = setdiff(JaJmTtl400OrMoreByAlphaHeader, remove)

dfEach4RWdNoOfOccu = function (x, y, z) {

for (i in 5: nrow(y)) {

if (i%%4 == 1) {

df <- y[i,]

}

else if (i%%4 != 0) {

df <- rbind(df, y[i,])

}

else {

df <- rbind(df, y[i,])

z = rbind(z, A(x, df))

}}

return(z)

}

dfJaWdFeqDf = A(JaJmTtl400OrMoreByAlphaHeader, dfJaneAusten26674\_27073[1:4,])

dfJaWdFeqDf = dfEach4RWdNoOfOccu(JaJmTtl400OrMoreByAlphaHeader, dfJaneAusten26674\_27073, dfJaWdFeqDf)

dfJmWdFeqDf = A(JaJmTtl400OrMoreByAlphaHeader, dfJohnMuir31420\_31819[1:4,])

dfJmWdFeqDf = dfEach4RWdNoOfOccu(JaJmTtl400OrMoreByAlphaHeader, dfJohnMuir31420\_31819, dfJmWdFeqDf)

dfJaAndJmWdFeqDf <- rbind(dfJaWdFeqDf, dfJmWdFeqDf)

#Add a column of labels and move it to the front

dfJaAndJmWdFeqDf$JAOrJM = c(rep('JA', 100), rep('JM', 100))

dfJaAndJmWdFeqDfLabled = dfJaAndJmWdFeqDf[,c(221,1:220)]

#shuffling rows:

set.seed(12345)

rrowNos <- sample(nrow(dfJaAndJmWdFeqDfLabled))

dfJaAndJmWdFeqDfLabledRandm <- dfJaAndJmWdFeqDfLabled[rrowNos,]

#normalisation of columns

data\_norm <- function(x) {(x- min(x))/ (max(x)- min(x))}

dfJaAndJmWdFeqDfLabledRandm\_norm <- as.data.frame(lapply(dfJaAndJmWdFeqDfLabledRandm[,-1], data\_norm))

#view normalisation summaryy of the first four columns

summary(dfJaAndJmWdFeqDfLabledRandm\_norm[,1:4])

#KNN!

library(class)

dfJaAndJmWdFeqDfLabledRandm\_norm\_train <- dfJaAndJmWdFeqDfLabledRandm\_norm[1:160,]

dfJaAndJmWdFeqDfLabledRandm\_norm\_test <- dfJaAndJmWdFeqDfLabledRandm\_norm[161:200,]

JaOrJm\_pred <- knn(dfJaAndJmWdFeqDfLabledRandm\_norm\_train, dfJaAndJmWdFeqDfLabledRandm\_norm\_test, dfJaAndJmWdFeqDfLabledRandm[1:160,1], k= 13)

table(pred\_3Hrs = JaOrJm\_pred, true\_JaneAusten\_JohnMuir\_KNN = dfJaAndJmWdFeqDfLabledRandm[161:200,1])

######################################################################

##code 2 (JaneAustenAndJohnMuirTmDtm.r)

#uel 19/20 DS7003

#WARNING! because there is a non-ascii character â in the code, after importing this code to RStudio, one should select File -> Reopen with encoding... then select UTF-8 (Windows)

#Jane Austen vs John Muir. Each 400 lines x 1000 words divided into 100 documents.

#so each document 4000 words and total 200 documents

#use dtm of package tm to form the document-word table

#set working directory and load package tm

setwd(dirname(file.choose()))

getwd()

library(tm)

#input data and form two dataframes

dfVictorianEraAA <- read.table('Gungor\_2018\_VictorianAuthorAttribution\_data-train.csv', header = TRUE, sep = (','))

dfJA26674\_27073 <- dfVictorianEraAA[26674:27073,]

dfJM31420\_31819 <- dfVictorianEraAA[31420:31819,]

#form corpa with dataframes. Texts already cleaned

#package tm is required

JA26674\_27073\_corpus <- VCorpus(VectorSource(dfJA26674\_27073$text))

JA26674\_27073\_corpus <- tm\_map(JA26674\_27073\_corpus, stripWhitespace)

JM31420\_31819\_corpus <- VCorpus(VectorSource(dfJM31420\_31819$text))

JM31420\_31819\_corpus <- tm\_map(JM31420\_31819\_corpus, stripWhitespace)

#form dtm. Each line(1000 words) a document

#change minimum word length to 1 from 3

JA26674\_27073\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(JA26674\_27073\_corpus, control=list(wordLengths = c(1, Inf)))))

JM31420\_31819\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(JM31420\_31819\_corpus, control=list(wordLengths = c(1, Inf)))))

#retain columns of words which can found both from JA and JM's texts

common\_cols <- intersect(colnames(JA26674\_27073\_dtDf), colnames(JM31420\_31819\_dtDf))

JAAndJMdtDf <- rbind(JA26674\_27073\_dtDf[common\_cols], JM31420\_31819\_dtDf[common\_cols])

#further retain columns of words each of which are at least appeared 400 times 0.05%

#(i.e. 0.05% of the total number of words: 800000)

JAAndJMdtDf400OrMore <- JAAndJMdtDf[, colSums(JAAndJMdtDf) >=400]#231 columns

#delete single character columns (but not m: reason: i'm/ t: can't/ etc.)

JAAndJMdtDf400OrMore[ ,c('â', 'e', 'f', 'h', 'j', 'l', 'n', 'o', 'r', 'u', 'v')] <- list(NULL)#231 to 220

#aggreate and sum every four lines

JAAndJMdtDf400OrMore$textNo <- rep(1:200, each = 4)

dfJaAndJmWdFeqDf <- aggregate(. ~ textNo, JAAndJMdtDf400OrMore, sum)

dfJaAndJmWdFeqDf$textNo <- NULL

#add labels JA and JM and put the label column to the front

dfJaAndJmWdFeqDf$JAOrJM <- c(rep('JA', 100), rep('JM', 100))

dfJaAndJmWdFeqDfLabled <- dfJaAndJmWdFeqDf[,c(221,1:220)]

#shuffling rows:

set.seed(12345)

rrowNos <- sample(nrow(dfJaAndJmWdFeqDfLabled))

dfJaAndJmWdFeqDfLabledRandm <- dfJaAndJmWdFeqDfLabled[rrowNos,]#ok

#normalising columns

data\_norm <- function(x) {(x- min(x))/ (max(x)- min(x))}

dfJaAndJmWdFeqDfLabledRandm\_norm <- as.data.frame(lapply(dfJaAndJmWdFeqDfLabledRandm[,-1], data\_norm))#ok

summary(dfJaAndJmWdFeqDfLabledRandm\_norm[,1:4])

#KNN!

library(class)

dfJaAndJmWdFeqDfLabledRandm\_norm\_train <- dfJaAndJmWdFeqDfLabledRandm\_norm[1:160,]

dfJaAndJmWdFeqDfLabledRandm\_norm\_test <- dfJaAndJmWdFeqDfLabledRandm\_norm[161:200,]

JaOrJm\_pred <- knn(dfJaAndJmWdFeqDfLabledRandm\_norm\_train, dfJaAndJmWdFeqDfLabledRandm\_norm\_test, dfJaAndJmWdFeqDfLabledRandm[1:160,1], k= 13)

#show cross table

table(pred = JaOrJm\_pred, true\_JaneAusten\_JohnMuir\_KNN = dfJaAndJmWdFeqDfLabledRandm[161:200,1]) #all correct

######################################################################

##code 3 (ThreeVictorianEraFemlNovlstsTmDtm.r)

#Three popular female novelists all born in the 1850s: 17 Helen Mathers 1853-1920 (18010- 18669 in the kaggle csv file), 32 Lucas Malet 1852-1931 (33861-34563), 33 Marie Corelli 1855-1924 (34564-36305)

#200 lines each

#there is a â in the code. If this code is loaded to RStudio, the encoding of it should be changed to UTF-8!

#set working directory and load package tm

setwd(dirname(file.choose()))

getwd()

library(tm)

#input data and form three dataframes

#be careful of the correctness of the filenames

#for example, hyphen or underscore?

dfVictorianEraAA <- read.table('Gungor\_2018\_VictorianAuthorAttribution\_data-train.csv', header = TRUE, sep = (','))

dfHelen\_Mathers18009\_18208 <- dfVictorianEraAA[18009:18208,]

dfLucas\_Malet33860\_34059 <- dfVictorianEraAA[33860:34059,]

dfMarie\_Corelli34563\_34762 <- dfVictorianEraAA[34563:34762,]

#form corpa from dataframes.

#texts are already all in lower case and no punctuation

#package tm is required

dfHelen\_Mathers18009\_18208\_corpus <- VCorpus(VectorSource(dfHelen\_Mathers18009\_18208$text))

dfHelen\_Mathers18009\_18208\_corpus <- tm\_map(dfHelen\_Mathers18009\_18208\_corpus, stripWhitespace)

dfLucas\_Malet33860\_34059\_corpus <- VCorpus(VectorSource(dfLucas\_Malet33860\_34059$text))

dfLucas\_Malet33860\_34059\_corpus <- tm\_map(dfLucas\_Malet33860\_34059\_corpus, stripWhitespace)

dfMarie\_Corelli34563\_34762\_corpus <- VCorpus(VectorSource(dfMarie\_Corelli34563\_34762$text))

dfMarie\_Corelli34563\_34762\_corpus <- tm\_map(dfMarie\_Corelli34563\_34762\_corpus, stripWhitespace)

#form dtm. Each line a document (1000 words)

#change minimum word length to 1 from 3

dfHelen\_Mathers18009\_18208\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfHelen\_Mathers18009\_18208\_corpus, control=list(wordLengths = c(1, Inf)))))

dfLucas\_Malet33860\_34059\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfLucas\_Malet33860\_34059\_corpus, control=list(wordLengths = c(1, Inf)))))

dfMarie\_Corelli34563\_34762\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfMarie\_Corelli34563\_34762\_corpus, control=list(wordLengths = c(1, Inf)))))

#retain only columns of words which can found both in HM, LM and MC's texts

common\_cols <- intersect(intersect(colnames(dfHelen\_Mathers18009\_18208\_dtDf), colnames(dfLucas\_Malet33860\_34059\_dtDf)), colnames(dfMarie\_Corelli34563\_34762\_dtDf))

HmLmMcDtDf <- rbind(dfHelen\_Mathers18009\_18208\_dtDf[common\_cols], dfLucas\_Malet33860\_34059\_dtDf[common\_cols], dfMarie\_Corelli34563\_34762\_dtDf[common\_cols])#5228 cols

#further retain only columns of words each of which are at least appeared

#300 times 0.05%

HmLmMcTtl300OrMore <- HmLmMcDtDf[, colSums(HmLmMcDtDf) >=300] #237

#texts quite untidy. number of â in HM 2077, LM 1743 and MC 6280

#delete the column â

HmLmMcTtl300OrMore$â <- NULL #236

#aggreate and sum every four lines (reduced to 150 lines)

#add and delete column textNO

HmLmMcTtl300OrMore$textNo <- rep(1:150, each = 4)

dfHmLmMcWdFeqDf <- aggregate(. ~ textNo, HmLmMcTtl300OrMore, sum)

dfHmLmMcWdFeqDf$textNo <- NULL

#add labels HM, LM and MC and put the column to the front

dfHmLmMcWdFeqDf$HmOrLmOrMc <- c(rep('HM', 50), rep('LM', 50), rep('MC', 50))

dfHmLmMcWdFeqDfLabled = dfHmLmMcWdFeqDf[,c(237,1:236)] #236+1

#shuffling rows:

set.seed(12345)

rrowNos <- sample(nrow(dfHmLmMcWdFeqDfLabled))

dfHmLmMcWdFeqDfLabledRandm <- dfHmLmMcWdFeqDfLabled[rrowNos,]

#normalisation

data\_norm <- function(x) {(x- min(x))/ (max(x)- min(x))}

dfHmLmMcWdFeqDfLabledRandm\_norm <- as.data.frame(lapply(dfHmLmMcWdFeqDfLabledRandm[,-1], data\_norm))

summary(dfHmLmMcWdFeqDfLabledRandm\_norm[,1:4]) #see whether normalised

#KNN!

library(class)

dfHmLmMcWdFeqDfLabledRandm\_norm\_train <- dfHmLmMcWdFeqDfLabledRandm\_norm[1:120,]

dfHmLmMcWdFeqDfLabledRandm\_norm\_test <- dfHmLmMcWdFeqDfLabledRandm\_norm[121:150,]

HmOrLmOrMc\_pred <- knn(dfHmLmMcWdFeqDfLabledRandm\_norm\_train, dfHmLmMcWdFeqDfLabledRandm\_norm\_test, dfHmLmMcWdFeqDfLabledRandm[1:120,1], k= 11)

table(pred = HmOrLmOrMc\_pred, true\_HelenMathers\_LucasMalet\_MarieCorelli\_KNN = dfHmLmMcWdFeqDfLabledRandm[121:150,1]) #mistake rate 1/30

#sqrt(120) = 10.954 . Therefore use k =11.

#k = 11 perform the best, only one error: 1 MC was misjudged as LM

#SVM! tune automatically

library("e1071")

HmOrLmOrMc\_svm\_model <- svm(dfHmLmMcWdFeqDfLabledRandm\_norm\_train, dfHmLmMcWdFeqDfLabledRandm[1:120,1], type = 'C')

pred <- predict(HmOrLmOrMc\_svm\_model, dfHmLmMcWdFeqDfLabledRandm\_norm\_test)

table(pred, true\_HelenMathers\_LucasMalet\_MarieCorelli\_SVM = dfHmLmMcWdFeqDfLabledRandm[121:150,1])

#all correct

#tune manually

dfHmLmMcWdFeqDfLabledRandm1To120AsFactors = as.factor(dfHmLmMcWdFeqDfLabledRandm[1:120,1])

set.seed(12345)

svm\_tune <- tune(svm, train.x = dfHmLmMcWdFeqDfLabledRandm\_norm\_train,

train.y = dfHmLmMcWdFeqDfLabledRandm1To120AsFactors,

kernel = 'linear',

type = 'C',

ranges = list(cost = c(.001,.01,.1,1,5,10,100)))

svm\_tune

svm\_tune$best.model

#besides best cost, also best number of support vectors, etc.

pred\_svm\_after\_tune <- predict(svm\_tune$best.model, dfHmLmMcWdFeqDfLabledRandm\_norm\_test)

table(pred = pred\_svm\_after\_tune, true\_HelenMathers\_LucasMalet\_MarieCorelli\_TunedSVM = dfHmLmMcWdFeqDfLabledRandm[121:150,1])

######################################################################

##code 4 (DickensEliotHardyAndFourOthersTmDtm200LinesEachAndEliotImmBlr80End80GutenbergMiddlemarch5Portions.r)

#uel 19/20 DS7003

#WARNING! because there is a non-ascii character â in the code, after importing this code to RStudio, one should select File -> Reopen with encoding... then select UTF-8

#Besides the three popular female English novelists I used in the last attempt (17 Helen Mathers 1853-1920 67 (18010- 18669 in csv file), 32 Lucas Malet 1852-1931 79 (33861-34563), 33 Marie Corelli 1855-1924 69 (34564-36305)), I added four novelists, a propular male novelist born in 1850s': 12 Fergue Hume 1859- 1932 73 (12599- 13185), 8 Charles Dickens 1812- 1870 58 (3399- 10312), 14 George Eliot, Real name Mary Ann Evans, author of Middlemarch 1819- 1880 71 (13671- 16366), 45 Thomas Hardy 1840- 1928 88 (48024- 50335)

#Number of words extracted from each of the seven authors' texts: 50 texts of 4000 words each (32 for training and 8 for testing).

#set working directory and load package tm

setwd(dirname(file.choose()))

getwd()

library(tm)

#input data and form seven dataframes

#please pay attention to file names. e.g. hyphen or underscore?

dfVictorianEraAA <- read.table('Gungor\_2018\_VictorianAuthorAttribution\_data-train.csv', header = TRUE, sep = (','))

dfCharles\_Dickens3398\_3597 <- dfVictorianEraAA[3398:3597,]

dfFergue\_Hume12558\_12757 <- dfVictorianEraAA[12558:12757,]

dfGeorge\_Eliot13670\_13869 <- dfVictorianEraAA[13670:13869,]

dfHelen\_Mathers18009\_18208 <- dfVictorianEraAA[18009:18208,]

dfLucas\_Malet33860\_34059 <- dfVictorianEraAA[33860:34059,]

dfMarie\_Corelli34563\_34762 <- dfVictorianEraAA[34563:34762,]

dfThomas\_Hardy48023\_48222 <- dfVictorianEraAA[48023:48222,]

#form corpa with dataframes. Texts already cleaned

#package tm is required here

dfCharles\_Dickens3398\_3597\_corpus <- VCorpus(VectorSource(dfCharles\_Dickens3398\_3597$text))

dfCharles\_Dickens3398\_3597\_corpus <- tm\_map(dfCharles\_Dickens3398\_3597\_corpus, stripWhitespace)

dfFergue\_Hume12558\_12757\_corpus <- VCorpus(VectorSource(dfFergue\_Hume12558\_12757$text))

dfFergue\_Hume12558\_12757\_corpus <- tm\_map(dfFergue\_Hume12558\_12757\_corpus, stripWhitespace)

dfGeorge\_Eliot13670\_13869\_corpus <- VCorpus(VectorSource(dfGeorge\_Eliot13670\_13869$text))

dfGeorge\_Eliot13670\_13869\_corpus <- tm\_map(dfGeorge\_Eliot13670\_13869\_corpus, stripWhitespace)

dfHelen\_Mathers18009\_18208\_corpus <- VCorpus(VectorSource(dfHelen\_Mathers18009\_18208$text))

dfHelen\_Mathers18009\_18208\_corpus <- tm\_map(dfHelen\_Mathers18009\_18208\_corpus, stripWhitespace)

dfLucas\_Malet33860\_34059\_corpus <- VCorpus(VectorSource(dfLucas\_Malet33860\_34059$text))

dfLucas\_Malet33860\_34059\_corpus <- tm\_map(dfLucas\_Malet33860\_34059\_corpus, stripWhitespace)

dfMarie\_Corelli34563\_34762\_corpus <- VCorpus(VectorSource(dfMarie\_Corelli34563\_34762$text))

dfMarie\_Corelli34563\_34762\_corpus <- tm\_map(dfMarie\_Corelli34563\_34762\_corpus, stripWhitespace)

dfThomas\_Hardy48023\_48222\_corpus <- VCorpus(VectorSource(dfThomas\_Hardy48023\_48222$text))

dfThomas\_Hardy48023\_48222\_corpus <- tm\_map(dfThomas\_Hardy48023\_48222\_corpus, stripWhitespace)

#form dtm. Each line(1000 words) a document

#change minimum word length to 1 from 3

dfCharles\_Dickens3398\_3597\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfCharles\_Dickens3398\_3597\_corpus, control=list(wordLengths = c(1, Inf)))))

dfFergue\_Hume12558\_12757\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfFergue\_Hume12558\_12757\_corpus, control=list(wordLengths = c(1, Inf)))))

dfGeorge\_Eliot13670\_13869\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfGeorge\_Eliot13670\_13869\_corpus, control=list(wordLengths = c(1, Inf)))))

dfHelen\_Mathers18009\_18208\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfHelen\_Mathers18009\_18208\_corpus, control=list(wordLengths = c(1, Inf)))))

dfLucas\_Malet33860\_34059\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfLucas\_Malet33860\_34059\_corpus, control=list(wordLengths = c(1, Inf)))))

dfMarie\_Corelli34563\_34762\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfMarie\_Corelli34563\_34762\_corpus, control=list(wordLengths = c(1, Inf)))))

dfThomas\_Hardy48023\_48222\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfThomas\_Hardy48023\_48222\_corpus, control=list(wordLengths = c(1, Inf)))))

#retain columns of words which can found in every of the seven authors' texts

common\_cols <- Reduce(intersect, list(colnames(dfCharles\_Dickens3398\_3597\_dtDf), colnames(dfFergue\_Hume12558\_12757\_dtDf), colnames(dfGeorge\_Eliot13670\_13869\_dtDf), colnames(dfHelen\_Mathers18009\_18208\_dtDf), colnames(dfLucas\_Malet33860\_34059\_dtDf), colnames(dfMarie\_Corelli34563\_34762\_dtDf), colnames(dfThomas\_Hardy48023\_48222\_dtDf)))

BindAll7NoOfWdsInAll7UniqWdLst <- rbind(dfCharles\_Dickens3398\_3597\_dtDf[common\_cols], dfFergue\_Hume12558\_12757\_dtDf[common\_cols], dfGeorge\_Eliot13670\_13869\_dtDf[common\_cols], dfHelen\_Mathers18009\_18208\_dtDf[common\_cols], dfLucas\_Malet33860\_34059\_dtDf[common\_cols], dfMarie\_Corelli34563\_34762\_dtDf[common\_cols],

dfThomas\_Hardy48023\_48222\_dtDf[common\_cols])

#further retain columns of words each of which appearing in the texts at least

#700 times(50 x 4000 x 7 x 0.05% = 700)

SevenAuthsTtl700OrMore <- BindAll7NoOfWdsInAll7UniqWdLst[, colSums(BindAll7NoOfWdsInAll7UniqWdLst) >=700] #230

#after inspection of the dataframe,

#remove columns â, o (but not t(can't), m(i'm), etc.)

#be careful! â is not an ascii character

SevenAuthsTtl700OrMore[,c('â', 'o')] <- list(NULL) #228

#add textNo column

#aggreate and sum every four lines (reduce to 350 lines from 1400 lines)

#delete column textNO

SevenAuthsTtl700OrMore$textNo <- rep(1:350, each = 4)

dfAll7WdFeqDf <- aggregate(. ~ textNo, SevenAuthsTtl700OrMore, sum)

dfAll7WdFeqDf$textNo <- NULL

#add labels and move the label column to the first column

dfAll7WdFeqDf$Label = c(rep('CD', 50), rep('FH', 50), rep('GE', 50), rep('HM', 50), rep('LM', 50), rep('MC', 50), rep('TH', 50)) # ?228+1 = 229

dfAll7WdFeqDfLabled = dfAll7WdFeqDf[,c(229,1:228)]

# shuffling rows:

set.seed(12345)

rrowNos <- sample(nrow(dfAll7WdFeqDfLabled))

dfAll7WdFeqDfLabledRandm <- dfAll7WdFeqDfLabled[rrowNos,]

#normalisation:

data\_norm <- function(x) {(x- min(x))/ (max(x)- min(x))}

dfAll7WdFeqDfLabledRandm\_norm <- as.data.frame(lapply(dfAll7WdFeqDfLabledRandm[,-1], data\_norm))

summary(dfAll7WdFeqDfLabledRandm\_norm[,1:4]) #see whether normalised

#KNN!

#number of k: usually start from sqrt of data points of

#the training set. So sqrt(280): 18

#then trial and error

library(class)

dfAll7WdFeqDfLabledRandm\_norm\_train <- dfAll7WdFeqDfLabledRandm\_norm[1:280,]

dfAll7WdFeqDfLabledRandm\_norm\_test <- dfAll7WdFeqDfLabledRandm\_norm[281:350,]

whichOfThe7\_pred <- knn(dfAll7WdFeqDfLabledRandm\_norm\_train, dfAll7WdFeqDfLabledRandm\_norm\_test, dfAll7WdFeqDfLabledRandm[1:280,1], k= 18)

table(pred = whichOfThe7\_pred, true\_7Authors\_KNN = dfAll7WdFeqDfLabledRandm[281:350,1])#mistake rate 1/70

#SVM!

library("e1071")

# simple: no tunning

whichOfThe7\_svm\_model <- svm(dfAll7WdFeqDfLabledRandm\_norm\_train, dfAll7WdFeqDfLabledRandm[1:280,1], type = 'C')

pred <- predict(whichOfThe7\_svm\_model, dfAll7WdFeqDfLabledRandm\_norm\_test)

table(pred, true\_7Authors\_SVM = dfAll7WdFeqDfLabledRandm[281:350,1]) #all correct

#use tunning to find costs

dfAll7WdFeqDfLabledRandmLabel1To280AsFactors = as.factor(dfAll7WdFeqDfLabledRandm[1:280,1])

set.seed(12345)

svm\_tune <- tune(svm, train.x = dfAll7WdFeqDfLabledRandm\_norm\_train,

train.y = dfAll7WdFeqDfLabledRandmLabel1To280AsFactors,

kernel = 'linear',

#type = 'C',

ranges = list(cost = c(.001,.01,.1,1,5,10,100)))

print(svm\_tune)

svm\_tune$best.model

#besides best cost, also best number of support vectors, etc.

pred\_svm\_after\_tune <- predict(svm\_tune$best.model, dfAll7WdFeqDfLabledRandm\_norm\_test)

table(pred = pred\_svm\_after\_tune, true\_7Authors\_tunedSVM = dfAll7WdFeqDfLabledRandm[281:350,1]) #all correct

#---------------------------------------------------------------

#Check whether the knn and svm models developed above (using the first 200 lines of each of the seven authors) can recognise the 20 documents (each x 4000 words) complied with George Eliot's lines extracted from end of her lines were written by her.

#need to use package tm

dfGeorge\_Eliot16286\_16365End <- dfVictorianEraAA[16286:16365,]

dfGeorge\_Eliot16286\_16365End\_corpus <- VCorpus(VectorSource(dfGeorge\_Eliot16286\_16365End$text))

dfGeorge\_Eliot16286\_16365End\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfGeorge\_Eliot16286\_16365End\_corpus, control=list(wordLengths = c(1, Inf)))))

dfGeorge\_Eliot16286\_16365End\_dtDf$textNo <- rep(1:20, each = 4)

dfGeEndWdFeqAll4000WrdsDf <- aggregate(. ~ textNo, dfGeorge\_Eliot16286\_16365End\_dtDf, sum)

dfGeEndWdFeqAll4000WrdsDf$textNo <- NULL

dfGeEndWdFeqAll4000WrdsDf[setdiff(colnames(dfAll7WdFeqDfLabled), colnames(dfGeEndWdFeqAll4000WrdsDf))] <- 0

dfGeEndWdFeqDf <- dfGeEndWdFeqAll4000WrdsDf[colnames(dfAll7WdFeqDfLabled)]

dfGeEndWdFeqDf$Label <- NULL #delete Label col

#'normalisation': use max and min values of the data for building the models

dfGeEndWdFeqDf\_addMaxMin = rbind(dfGeEndWdFeqDf, apply(dfAll7WdFeqDfLabledRandm[,-1], 2, max), apply(dfAll7WdFeqDfLabledRandm[,-1], 2, min))

normGeEtc = function(x, y) {

for (i in 2: (nrow(y)-2)) {

x = rbind(x, (y[i,] - y[nrow(y),]) / (y[(nrow(y)-1),] - y[nrow(y),]))

}

return(x)

}

#'normalise' the first row

dfGeEndWdFeqDf\_normNotReal = (dfGeEndWdFeqDf\_addMaxMin[1,] - dfGeEndWdFeqDf\_addMaxMin[22,]) / (dfGeEndWdFeqDf\_addMaxMin[21,] - dfGeEndWdFeqDf\_addMaxMin[22,])

#'normalise' the rest

dfGeEndWdFeqDf\_normNotReal = normGeEtc(dfGeEndWdFeqDf\_normNotReal, dfGeEndWdFeqDf\_addMaxMin)

#KNN!

set.seed(12345)

all20GeEnd\_knn\_pred <- knn(dfAll7WdFeqDfLabledRandm\_norm\_train, dfGeEndWdFeqDf\_normNotReal, dfAll7WdFeqDfLabledRandm[1:280,1], k= 18)

table(pred = all20GeEnd\_knn\_pred, true\_GeorgeEliotEnd\_KNN = rep('GE', 20)) #mistake rate 4/20

#svm\_no\_tune

pred\_svm\_GeEnd <- predict(whichOfThe7\_svm\_model, dfGeEndWdFeqDf\_normNotReal)

table(pred = pred\_svm\_GeEnd, true\_GeorgeEliotEnd\_SVM = rep('GE', 20)) #mistake rate 1/20

#svm\_tuned

pred\_svm\_after\_tune\_GeEnd <- predict(svm\_tune$best.model, dfGeEndWdFeqDf\_normNotReal)

table(pred = pred\_svm\_after\_tune\_GeEnd, true\_GeorgeEliotEnd\_tunedSVM = rep('GE', 20)) #mistake rate 2/20

#---------------------------------------------------------------

#Check whether the knn and svm models developed above (using the first 200 lines of each of the seven authors) can recognise that the 20 documents (each x 4000 words) complied with George Eliot's lines extracted from immediately below the first 200 lines were written by her.

#need to use package tm

#(no comment lines below. For explanations, see the above part)

dfGeorge\_Eliot13870\_13949ImmBlw <- dfVictorianEraAA[13870:13949,]

dfGeorge\_Eliot13870\_13949ImmBlw\_corpus <- VCorpus(VectorSource(dfGeorge\_Eliot13870\_13949ImmBlw$text))

dfGeorge\_Eliot13870\_13949ImmBlw\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfGeorge\_Eliot13870\_13949ImmBlw\_corpus, control=list(wordLengths = c(1, Inf)))))

dfGeorge\_Eliot13870\_13949ImmBlw\_dtDf$textNo <- rep(1:20, each = 4)

dfGeImmBlwWdFeqAll4000WrdsDf <- aggregate(. ~ textNo, dfGeorge\_Eliot13870\_13949ImmBlw\_dtDf, sum)

dfGeImmBlwWdFeqAll4000WrdsDf$textNo <- NULL

dfGeImmBlwWdFeqAll4000WrdsDf[setdiff(colnames(dfAll7WdFeqDfLabled), colnames(dfGeImmBlwWdFeqAll4000WrdsDf))] <- 0

dfGeImmBlwWdFeqDf <- dfGeImmBlwWdFeqAll4000WrdsDf[colnames(dfAll7WdFeqDfLabled)]

dfGeImmBlwWdFeqDf$Label <- NULL #delete Label col

dfGeImmBlwWdFeqDf\_addMaxMin = rbind(dfGeImmBlwWdFeqDf, apply(dfAll7WdFeqDfLabledRandm[,-1], 2, max), apply(dfAll7WdFeqDfLabledRandm[,-1], 2, min))

normGeEtc = function(x, y) {

for (i in 2: (nrow(y)-2)) {

x = rbind(x, (y[i,] - y[nrow(y),]) / (y[(nrow(y)-1),] - y[nrow(y),]))

}

return(x)

}

dfGeImmBlwWdFeqDf\_normNotReal = (dfGeImmBlwWdFeqDf\_addMaxMin[1,] - dfGeImmBlwWdFeqDf\_addMaxMin[22,]) / (dfGeImmBlwWdFeqDf\_addMaxMin[21,] - dfGeImmBlwWdFeqDf\_addMaxMin[22,])

dfGeImmBlwWdFeqDf\_normNotReal = normGeEtc(dfGeImmBlwWdFeqDf\_normNotReal, dfGeImmBlwWdFeqDf\_addMaxMin)

all20GeImmBlw\_knn\_pred <- knn(dfAll7WdFeqDfLabledRandm\_norm\_train, dfGeImmBlwWdFeqDf\_normNotReal, dfAll7WdFeqDfLabledRandm[1:280,1], k= 18)

table(pred = all20GeImmBlw\_knn\_pred, true\_GeorgeEliotImmBlw\_KNN = rep('GE', 20)) #mistake rate 1/20

#svm\_no\_tune

pred\_svm\_GeImmBlw <- predict(whichOfThe7\_svm\_model, dfGeImmBlwWdFeqDf\_normNotReal)

table(pred = pred\_svm\_GeImmBlw, true\_GeorgeEliotImmBlw\_SVM = rep('GE', 20)) #all correct

#svm\_tuned

pred\_svm\_after\_tune\_GeImmBlw <- predict(svm\_tune$best.model, dfGeImmBlwWdFeqDf\_normNotReal)

table(pred = pred\_svm\_after\_tune\_GeImmBlw, true\_GeorgeEliotImmBlw\_tunedSVM = rep('GE', 20)) #mistake rate: 1/20

#---------------------------------------------------------------

#The data file contains George Eliot's masterpiece Middlemarch (300000+ words). However, it is in the region around line 15000, not in the lines already used above (lines 13670 - 13949 and 16286 - 16368). Therefore, a copy of this novel was obtained from the famous Gutenberg book corpus and from which 5 portions of words were extracted for doing the below experiment (Each portion contains 4000 words. All words in lowercase. All numbers and punctuation and most person and place names were deleted)

#data file name: GutenbergMiddlemarch\_5PortionsEach4000Words.csv

#The KNN and SVM models produced above can correctly recognise that all the 5 portions were written by George Eliot.

dfGutenbergMiddlemarch <- read.table('GutenbergMiddlemarch\_5PortionsEach4000Words.csv', header = TRUE, sep = (','))

dfGutenbergMiddlemarch\_corpus <- VCorpus(VectorSource(dfGutenbergMiddlemarch$text))

dfGutenbergMiddlemarch\_dtDf <- as.data.frame(as.matrix(DocumentTermMatrix(dfGutenbergMiddlemarch\_corpus, control=list(wordLengths = c(1, Inf)))))

dfGutenbergMiddlemarch\_dtDf$textNo <- NULL

dfGutenbergMiddlemarch\_dtDf[setdiff(colnames(dfAll7WdFeqDfLabled), colnames(dfGutenbergMiddlemarch\_dtDf))] <- 0

dfGutenbergMiddlemarchWdFeqDf <- dfGutenbergMiddlemarch\_dtDf[colnames(dfAll7WdFeqDfLabled)]

dfGutenbergMiddlemarchWdFeqDf$Label <- NULL #delete Label col

dfGutenbergMiddlemarchWdFeqDf\_addMaxMin = rbind(dfGutenbergMiddlemarchWdFeqDf, apply(dfAll7WdFeqDfLabledRandm[,-1], 2, max), apply(dfAll7WdFeqDfLabledRandm[,-1], 2, min))

dfGutenbergMiddlemarchWdFeqDf\_normNotReal = (dfGutenbergMiddlemarchWdFeqDf\_addMaxMin[1,] - dfGutenbergMiddlemarchWdFeqDf\_addMaxMin[7,]) / (dfGutenbergMiddlemarchWdFeqDf\_addMaxMin[6,] - dfGutenbergMiddlemarchWdFeqDf\_addMaxMin[7,])

dfGutenbergMiddlemarchWdFeqDf\_normNotReal = normGeEtc(dfGutenbergMiddlemarchWdFeqDf\_normNotReal, dfGutenbergMiddlemarchWdFeqDf\_addMaxMin)

#KNN!

set.seed(12345)

all5GEMiddlemarch\_knn\_pred <- knn(dfAll7WdFeqDfLabledRandm\_norm\_train, dfGutenbergMiddlemarchWdFeqDf\_normNotReal, dfAll7WdFeqDfLabledRandm[1:280,1], k= 18)

table(pred = all5GEMiddlemarch\_knn\_pred, true\_GeorgeEliotMiddlemarch\_KNN = rep('GE(Middlemarch5Portions)', 5)) #all correct

#svm\_no\_tune

pred\_svm\_all5GEMiddlemarch <- predict(whichOfThe7\_svm\_model, dfGutenbergMiddlemarchWdFeqDf\_normNotReal)

table(pred = pred\_svm\_all5GEMiddlemarch, true\_GeorgeEliotMiddlemarch\_SVM = rep('GE(Middlemarch5Portions)', 5)) #all correct

#svm\_tuned

pred\_svm\_after\_tune\_all5GEMiddlemarch <- predict(svm\_tune$best.model, dfGutenbergMiddlemarchWdFeqDf\_normNotReal)

table(pred = pred\_svm\_after\_tune\_all5GEMiddlemarch, true\_GeorgeEliotMiddlemarch\_tunedSVM = rep('GE(Middlemarch5Portions)', 5)) #all correct