DS7004 Work-based Project Review  
Preliminary progress report  
(file name: u123456\_DS7004\_draft1.docx)  
5 August 2020

Student name: Chung Hoi CHIU  
Student number: u1720146

Note: I regret that, at present, I cannot write a long draft of the coursework, because I needed to spend time acquiring skills in mathematics, statistics, computing and deep learning for understanding to understand the two master thesis projects. I can only briefly list what I have done below.

With regard to:  
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 Unpublished MSc Thesis, Purdue University, Available at: https://scholarworks.iupui.edu/handle/1805/15938 (Accessed: 12 July 2020).

1. The biggest achievement of this project is that it created a large corpus of 1152 19th century books, which were written mainly by American and British novelists (total of 50 authors). The corpus is now downloadable from uci (corpus's name: 'Victorian Era Authorship Attribution Data Set').
2. However, the corpus has many significant shortcomings. The most salient ones are:  
     
   a. The quality is poor, because the quality of the texts of the source, Google's GDELT database, is poor.  
      
   b. Only the most frequent 10,000 words of the texts were retained. However, deleting rare words is not necessary in authorship attribution studies.  
     
   c. Labels of the training set are incorrect, which require that I spent about ten days to find the correct labels (only part of which I have completed). Labels of the test set are missing, which renders the test set useless.
3. The project mentioned the character n-gram method, but it did not use it to perform authorship attribution studies. However, it inspired me to use character 4-gram as the classifier to analyse the same texts I used in DS7003. By using 4-gram as the classifier, the results are as good as the results obtained from using common words as the classifier, which I used in DS7003 (Please see pictures in the Appendices). Character n-gram is a strange classifier which perhaps is used only for authorship attribution studies. Therefore, I can find only one R library that has a function to perform character n-gram dissection ('stylo').
4. The project used the traditional machine learning method, SVM, to perform authorship attribution studies, but the results were very poor (the accuracy rate is only about 60%), because it made a colossal mistake. It deleted from the texts the common words before feeding the texts to the learning machine. However, frequency of common words is one of the two most successful classifiers in authorship attribution studies (the other one is character n-gram).
5. The project mentioned deep learning and one of the word embedding techniques (word2vec). It inspired me to start delving into deep learning and word embedding techniques such as word2vec and GloVe. The project did not yield good results when using word2vec, which I suspect is because the project applied only some already available code to the texts, and the code was not for conducting authorship attribution studies. Therefore, I decided to learn deep learning and word embedding skills first before revisiting the project.
6. There are four most commonly used deep learning APIs which are also widely used to perform NLP studies: Tensorflow, Keras, Pytorch and MXNet. Tensorflow was invented and is backed by Google. Keras and Pytorch were built on top of Tensorflow to make Tensorflow easier to use. MXnet is backed by Amazon.
7. R has packages to link Tensorflow, Keras and Pytorch which can work only in Python/Conda environments. I spent two whole days linking RStudio to Keras and Tensorflow, after which I performed a simple deep learning study on the common word dataframe that I produced in DS7003 (1 to 3 neural node layers with reLu activation function + a softmax layer). The result is as good as using SVM (accuracy rate > 95%, but the task is easy. Please see the resulting pictures and code in the Appendices).

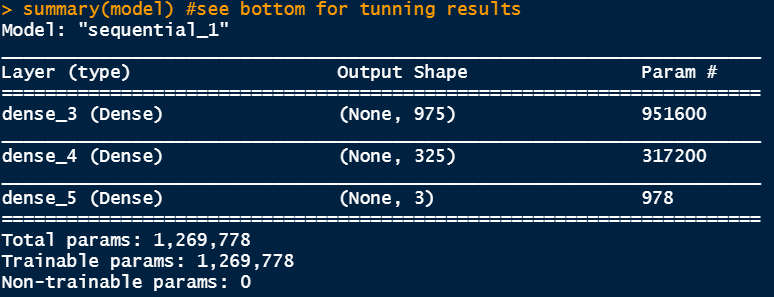
MXNet can be directly installed in RStudio but the procedure is quite complicated, unlike the procedure for installing in Python.

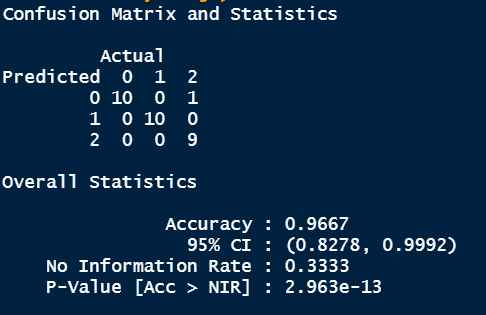
1. I found R is very good in text data cleaning and processing. However, since all important deep learning APIs that are suitable to perform NLP studies are Python biased, and, more importantly, it is much easier to obtain help in the field of deep learning if I code with Python, I must also learn Python.
2. It will be very difficult to have any breakthroughs in authorship studies if we are still confined to using traditional machine learning methods, such as SVM, multiclass logistic regression and random forest. Therefore, as inspired by this thesis, I will delve into deep learning and word embedding techniques. Furthermore, as indicated by this project, common word embedding techniques are suitable only to conduct semantic based NLP studies, such as sentiment analysis. I need to find or to invent new word embedding methods to conduct authorship attribution.

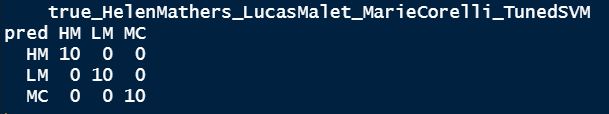
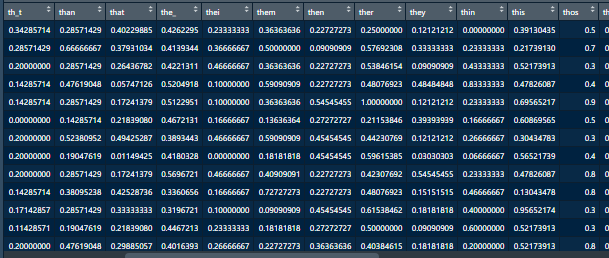
With regard to:  
Segarra S. (2014) Word adjacency networks for authorship attribution: solving Shakespearean controversies. Unpublished MSc Thesis, University of Pennsylvania, Available at: https://cpb-us-e1.wpmucdn.com/blogs.rice.edu/dist/a/9284/files/2018/07/MSc\_Thesis\_Segarra-zwwuyf.pdf (Accessed: 12 July 2020).

1. The novel method invented by this project, work adjacency networks, is attractive, since, in addition to the frequency of occurrence, it also pays attention to the distribution of common words. However, I will wait to see whether the inventors of this method will, ultimately, after repeated urging, subject the method to be tested by independent practitioners.
2. However, this project used 19th century texts to develop the method. Only after the method was developed and tested, was it applied to analysing early modern English texts, which was the aim of this project. This is a good approach. At the developmental stage of a machine learning technique, easier and simpler test samples, instead of complicated ones, should be used.

Appendix I:  
Resulting pictures:

Ddeployment of the neural network:  
(Two neural node layers with reLu activation function and a softmax layer)  


The confusion matrix for the results obtained from using character 4-gram as the classifier and the above neural network:   


The confusion matrix for the results obtained from using character 4-gram as the classifier and SVM:   
  
Part of the normalised character 4-gram table:  
  
  
Appendix II:  
The code for applying KNN, SVM and neural network techniques on character 4-gram classifier to perform authorship attribution studies:

#DS7004 Gungor's dataset/ character 4-gram

#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()

if (!require('tm')) install.packages('tm'); library('tm')

#input data and form three dataframes

if(!file.exists('Gungor\_2018\_VictorianAuthorAttribution\_data-train.csv')){

download.file('http://archive.ics.uci.edu/ml/machine-learning-databases/00454/dataset.zip', 'dataset.zip')

unzip('dataset.zip')

file.copy('./dataset/Gungor\_2018\_VictorianAuthorAttribution\_data-train.csv', '.')

#if the working directory does not have the csv file, this if statement

#needs several minutes to run

}

dfVictorianEraAA <- read.table('Gungor\_2018\_VictorianAuthorAttribution\_data-train.csv', header = TRUE, sep = (','), encoding = 'utf-8')

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

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

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

# Function for forming character 4-Grams

if (!require('stylo')) install.packages('stylo'); library('stylo')

library(stylo)

charNGramDf <- function(columnCell) {

my.text = gsub('\\s+', "\_", columnCell, perl = T)

my.vector.of.chars = txt.to.features(my.text, features = "c")

x = make.ngrams(my.vector.of.chars, ngram.size = 4)

xx = lapply(x,function(x) gsub('(?<=[\\S]) (?=[\\S])', '',x, perl = T))

return(paste(xx, collapse = ' '))

}

# Converting to 4-grams texts

dfHelen\_Mathers18009\_18208 <- as.data.frame(cbind(lapply(dfHelen\_Mathers18009\_18208[,1], charNGramDf), dfHelen\_Mathers18009\_18208[,2]))

colnames(dfHelen\_Mathers18009\_18208) <- c('text', 'author')

dfLucas\_Malet33860\_34059 <- as.data.frame(cbind(lapply(dfLucas\_Malet33860\_34059[,1], charNGramDf), dfLucas\_Malet33860\_34059[,2]))

colnames(dfLucas\_Malet33860\_34059) <- c('text', 'author')

dfMarie\_Corelli34563\_34762 <- as.data.frame(cbind(lapply(dfMarie\_Corelli34563\_34762[,1], charNGramDf), dfMarie\_Corelli34563\_34762[,2]))

colnames(dfMarie\_Corelli34563\_34762) <- c('text', 'author')

#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])#15220 cols

#delete columns with their names contain â #14924

HmLmMcDtDf <- HmLmMcDtDf[, -grep(pattern = '.\*â.\*â\*.\*', colnames(HmLmMcDtDf))]

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

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

#600 times

HmLmMcTtl600OrMore <- HmLmMcDtDf[, colSums(HmLmMcDtDf) >=600] #975

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

#add and delete column textNO

HmLmMcTtl600OrMore$textNo <- rep(1:150, each = 4)

dfHmLmMcWdFeqDf <- aggregate(. ~ textNo, HmLmMcTtl600OrMore, 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(976,1:975)] #975+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

View(dfHmLmMcWdFeqDfLabledRandm\_norm)

#KNN!

if (!require('class')) install.packages('class'); 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

if (!require('e1071')) install.packages('e1071'); library('e1071')

HmOrLmOrMc\_svm\_model <- svm(dfHmLmMcWdFeqDfLabledRandm\_norm\_train, as.factor(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])

# Deep learning using package keras

# import keras

# note: use\_condaenv("r\_reticulate") is only work for my Asus PC + Windows 10

# see README.md for matters related to installation of Python

if (!require('keras')) install.packages('keras'); library('keras')

use\_condaenv("r\_reticulate")

# Convert to matrix

training <- as.matrix(dfHmLmMcWdFeqDfLabledRandm\_norm[1:120,])

dimnames(training) <- NULL

test <- as.matrix(dfHmLmMcWdFeqDfLabledRandm\_norm[121:150,])

dimnames(test) <- NULL

# Convert labels to numerics and one hot encoding form

trainLabels <- to\_categorical(as.numeric(as.factor(dfHmLmMcWdFeqDfLabledRandm[1:120,1])) - 1)

testtarget <- as.numeric(as.factor(dfHmLmMcWdFeqDfLabledRandm[121:150,1])) - 1

testLabels <- to\_categorical(testtarget)

# Create sequential model (975 input columns, 3 categories)

model <- keras\_model\_sequential()

model %>% #one hidden layer, units = 975 (975 input columns, 3 categories)

layer\_dense(units=975, activation = 'relu', input\_shape = c(975)) %>%

layer\_dense(units=325, activation = 'relu', input\_shape = c(325)) %>%

layer\_dense(units = 3, activation = 'softmax')

summary(model) #see bottom for tunning results

# Compile

model %>%

compile(loss = 'categorical\_crossentropy',

optimizer = 'adam',

metrics = 'accuracy')

# Fit model

history <- model %>%

fit(training,

trainLabels,

epoch = 200,

batch\_size = 64,

validation\_split = 0.2)

# Prediction & confusion matrix - test data and labels

pred <- model %>%

predict\_classes(test)

library(caret)

confusionMatrix(table <- table(Predicted = pred, Actual = testtarget), mode = "everything")

#prob, pred, testtarget:

prob <- model %>%

predict\_proba(test)

cbind(prob, pred, testtarget)

#epoch 200 batch\_size 32 validation\_split 0.2 325 3

#f1: 0.9524 1 0.9474

#changed to 975 3 the same