EXTENDED ESSAY

Research Question:

To what extent are character-level convolutional neural networks viable for classifying texts by their century of creation?

Subject: Computer Science

Report word count: 3998

Number of pages: 33

Done by: Matvey Ryabov

Candidate number: 000556-0010

School: Winston Churchill High School, Lethbridge Alberta

Contents

1 Introduction

Natural language processing (NLP) is a rapidly growing field within data science. This field has led to the development of specialized machine learning models that allow us to summarize, analyze, and classify text. These specialized models are very applicable to a variety of situations. For example, if a company wants to know more about its customers, it can use NLP to summarize and classify reviews - thus, gaining valuable insight [1]. I work in the online shopping department at my local Superstore which happens to employ NLP to discover potential improvements that can be made in our department by analyzing customer surveys. For example, if customers are dissatisfied with our selection of produce, NLP can allow us to detect this sentiment without needing a human to read every single review. Since customer sentiment can be described and communicated in so many ways (there are many ways one can word one's dissatisfaction - such as using sarcasm), it's almost necessary for us to use NLP if we want to get accurate insight - rather than getting humans to read each review or relying on other more traditional algorithms (such as word frequency analysis). I received part of my inspiration for this paper while working in my department.

Another potential application of NLP would be if the local library wants to label each of their books based on its genre - it can use NLP models

to accurately [2] classify and label each book without requiring as much manual labor. Since I am an avid reader and have been looking at reading classics on my kindle, I've come across the Gutenberg project which makes available a lot of classic books in electronic format. The issue I saw was that each book was not labeled with an actual creation date. The date available in the metadata, was the date the book was re-released by the Gutenberg project. Thus, if I wanted to find out the actual creation date of the book I was reading, I had to refer to outside sources. This served as my main inspiration for this paper - I thought that it would be interesting to be able to label books with their creation date using NLP.

One recent development in the field of NLP was the discovery that character-level convolutional neural networks (CNNs) can be applied to the problem of text-classification and yield significant performance increases - when compared with traditional methods. The structure of CNNs allow them to capture patterns in texts such as word appearance and word arrangement. This paper hopes to take advantage of CNNs and their associated performance, and to answer the question of: **"To what extent are character-level CNNs viable for classifying texts by century?"** - when they are created. This research question is unique when compared with prior applications - it is a rather difficult one. Firstly, there is a problem of focus. There are so many different ways the CNN might end up approaching the problem as there isn't one surefire way to discern the century of a text - or that we've at least discovered. A CNN can look at the frequency of occurrence of a certain phrase such as "hereto" in order to discern between the 20th and 18th century. Or, it might notice that the 20th

century texts had more of a focus on the theme of "progress" rather than let's say "liberty". Perhaps it notices that 18th century texts were more often from the perspective of a farmer rather than a factory worker. The second point of hardship is that the amount of data needed to be processed is significantly more than prior applications that have been dealing with the classification of something around the length of an article. An article is short and the meaning - if that's what you are classifying - is somewhere in the body of the text. This application is using a dataset of texts, some having many chapters. My computing resources are not powerful enough to have the CNN read each text fully. Thus, I've had to shorten each text, hoping that the patterns needed to discern between the two centuries, were in the shortened version of the text. Perhaps, the CNN won't be able to identify the prevalence of 'liberty' in the texts of the 18th century all because I had to shorten them. Also, because my CNN (design) will be looking at characters as being the smallest unit of text, there will be many more 'features' or attributes than instances or data points. The length of each text is 3740 characters, while there are only 2101 books. Thus there is a small dimensionality problem (often referred to as the Curse of Dimensionality). For this reason, there might not be enough data to achieve the full performance a CNN potentially has to offer. For these two reasons, my problem is rather unique and challenging. Ideally, this research question will shed more light on the limitations of current-day CNNs.

To answer this research question, I will first provide a theoretical background of CNNs. Then, I will explain in more detail my chosen CNN design (sometimes referred to as architecture). Finally, I will carry out an exper-

iment to actually obtain some experimental data and figure out whether using CNNs for this application would be viable. To ensure the validity of my results, I will be using cross-validation to ensure their accuracy.

2 Theoretical Background

2.1 Neural Networks

Neural networks (NNs) are a type of machine learning model that learns from data to either create or classify data points. NNs are made up of several layers of interconnected nodes (node is practically synonymous with neuron and perception) that propagate data from an input layer to an output layer. The whole model is represented by a series of matrix operations (1) where the data at each layer is first summed, then weighed, then 'biased', and then passed through an activation function. This process is called the feed-forward process and allows NNs to carry out complicated logical 'decisions' - for example, two parallel nodes can act as a XOR or an AND logical operation. Because the behavior of nodes can be adjusted (learned), NNs

can find the patterns that contribute to a certain classification.

$$\overbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \end{bmatrix}}^{\text{output}} = \varphi \left(\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_j \end{bmatrix}}_{\text{input}} * \underbrace{\begin{bmatrix} w_{11} & \cdots & w_{1j} \\ \vdots & \ddots & \vdots \\ w_{i1} & \cdots & w_{ij} \end{bmatrix}}_{\text{weights}} + \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \end{bmatrix}}_{\text{biases}} \right) \tag{1}$$

Equation 1: The matrix operations between layers, where phi is the activation function.

Each connection between nodes is assigned a weight value. This weight value corresponds to how much 'weight' is placed on the information going between the connected nodes. If the weight is high, the node is more affected by the firing - which represents the information - of the connected node. Starting from the input layer, the inputs are multiplied by the weights and are summed. This is done through a dot product, so we don't have to deal with each node individually. Each node's information is stored as elements inside of matrices - instead of having separate vectors of weights for each node, we put them together into a matrix. Each node also has a bias. Since nodes are supposed to represent neurons, the bias is relative to how much easier it is for a neuron to fire; by adding a bias to the weighed sum, we get a higher probability of the activation function 'firing'. The activation function is a mathematical representation of the action potential of a neuron. If the weighed sum + bias is large enough, the activation function 'fires' - activates.

Because the output layer is also made up of nodes, each node can be assigned to represent a different class. When the network is inputted a data vector, the data propagates through the network and eventually arrives at the output layer where the node representing a correct class fires. If several nodes fire or if the correct node doesn't fire at all, the NN uses the negative gradient of the error (chosen as part of the architecture) function to then adjust the inner variables (this is done during the training period) - the weights and biases described above. My chosen error function works by finding the difference between the optimal output and the real output (and then summing and squaring the difference). This is why a NN is a machine **learning** model.

2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) don't differ significantly from regular neural networks (NNs). The main difference is that prior to being fed into a regular NN (referred to as "fully-connected layers"), the data is 'filtered' and simplified by a set of specialized layers.

If you wanted to work with an image, you could technically have each pixel be a separate node in the input layer. However, as you add more and more input nodes in the input layer, the computational power needed, quickly increases to non-manageable levels [3]; as you add more nodes you also add more weights and biases which make the matrix math quite computationally intensive. As a solution, it was discovered that applying convolutions and sub-sampling - via these specialized layers - would allow NNs to work with multidimensional data, such as images, without sacrificing much performance. Applying convolutions and sub-sampling, simplifies the data -

while retaining the most important features - and allows us to comfortably work with these forms of 'high-density' information. The convolutional and sub-sampling layers work by using a set of scanning kernels (often reffered to as filters and represented as a matrix) - that move from the top left corner of the image to bottom right - which apply the specific operation (convolution or sub-sampling) to each pixel in the image (the size of the kernel at a time). For this reason, CNNs are able to capture spatial and temporal dependencies in the data [4], which are ultimately the features most important for the classification of a picture.

Because CNNs work by using kernels that capture spatial and temporal dependencies, they are naturally very popular for image classification. However, because text also has spatial and temporal dependencies, CNNs can also be used for text classification with very minor modification to the data and the model architecture. Previous studies have found that CNNs can be successfully applied to traditional text classification problems and with very performative results. In this case, I used a CNN because I wanted to take advantage of the aforementioned ability to discern spatial and temporal patterns, and the flexibility that they provide - because of their computational efficiency, I am less limited by the type of feature that I am allowing the CNN to discern. Using traditional models, I've always had to decide on which features I would allow the model to learn by configuring it to take in a certain input - such as a word frequency dictionary. With a character-level CNN, I can just feed in the text with minor amounts of pre-processing. A CNN configured to classify text would also use kernels to scan the text, however, the kernels would be only one dimensional - in an image, the vertical

placement of pixels matters more, than the vertical placement of characters in text.

2.3 Convolution and Sub-sampling Operations

The convolution operation (see Figure 1) works by multiplying (element-wise matrix multiplication) elements of the kernel and the numerical value of the pixels in an image or the characters in a text (usually the intensity of color of a pixel or the alphabetically assigned numerical ID of the character) at that particular place in the image or the text. The elements of a kernel (represented as a matrix) are equivalent to the weights in a normal NN and after the 'weighing' has occurred, the matrix elements are summed - just like in a regular NN. Like the weights in a regular MN, a kernel's elements are adjusted during the training period. Since the CNN uses several kernels at each layer and since they are adjusted (trained), each kernel ends up focusing on a feature - such as one kernel might look for a circle in the image while another might look for an X. The output of each kernel is referred to as a feature map and is what is transferred to the next layer. In the final layer the feature maps are collapsed into one dimension and are then fed into the fully connected layers (the regular NN part), which would look at the presence of features when making its decision. In text, the feature maps might be representing a word, phrase, or even punctuation. Using these feature maps, a CNN would ideally be able to learn how to classify the century of a text.

Figure 1: The convolution operation.[5]

The subsampling operation is quite simpler than the convolution operation. All it does is simplify the image or the text using one of several subsampling methods that are chosen by the model's architecture. In my case, the subsampling method chosen was max-pooling, which works by also using scanning kernels but which take the largest numerical value present in the 'field of view' of the kernel and assemble that value into a thus downsampled output matrix. Ideally, using the right number of max-pooling layers, the CNN would be left with simple enough data to collapse the dimensions of, and feed into the fully-connected layers. If the final feature maps are still quite large (after all the max-pooling and convolutions), one could use a global pooling layer which sub-samples the whole feature map (using MAX in my case) and outputs a single value. Using convolutions and sub-sampling operations, a CNN can simplify and work with quite information-dense data. Refer to Figure 2 for a visual representation of a typical CNN structure.

Figure 2: A typical CNN.[6]

2.4 Epochs and Cross-Validation

Because I don't have infinite data to train my model on, I had to train the model on the same data for several 'epochs' - iterations. For this investigation I found that 20 worked best.

Cross-validation (see Figure 3) is a method of validation used to make sure the model isn't overfitting (too specialized and fitted to the training dataset). It works by splitting the dataset 90% training, 10% testing. After

all the testing has occurred on the 10% split, the two combine and the process is repeated on another different 90% 10% split (there is no overlap in the testing 10% between repetitions). This way we are able to test/train on 100% of the data. This allows us to see if the model is doing what it is supposed to - learning how to classify the texts rather than just memorizing the dataset.

Figure 3: k-cross validation where k is in my case 10. At each iteration different portions of the dataset serve as training and testing.

3 Experimental Methodology & Materials

3.1 Experimental Procedure

1. Pre-process the data and convert it into a usable file format for Weka (See Figure 4).

2. Set up the model using the Weka command line interface (commands used are included in appendix).

3. Set up the instance iterator to read each book from left to right and recognize each column as a character.

4. Train the model on the training data (a 90% split from the initial dataset).

5. Run the trained model on the testing data (a 10% split from the initial dataset).

6. Record the evaluation metrics and repeat 10 times (10 folds).

Figure 4: The program I used, Weka, is used to rapidly prototype and test different models and algorithms. I used the command line interface.

3.2 The Dataset Used

The dataset was self-compiled by downloading the freely available texts released by the Gutenberg project. Since I don't have the computational resources to be able to process whole books, each text was shortened to 3740 words - the length of the shortest text in the dataset. For the data to be properly used, some necessary pre-processing measures have been taken. In total, there were 2101 usable texts - some weren't properly labeled with a date which I could train the model with - 1051 texts from the 18th century and 1050 texts from the 20th.

3.3 Pre-processing

The data was pre-processed in R using a custom script (see appendix). As I mentioned earlier, each of the texts was shortened to 3740 characters. Before this, however, each text had 4000 characters of the beginning removed in order to remove the Project Gutenberg header. Then, all extra whitespace and non-alphanumerical characters were removed. The text was lowercased and each of the characters was translated into a number from 1-37 - where 37 is the space character. After pre-processing, the data was outputted as a .csv where each comma separated value represented a character. The last

column in the file stored the label of the data, in this case, the century - either 18th or 20th.

3.4 Model Architecture

The model architecture I chose consists of 2 convolution layers separated by sub-sampling layers, a global-pooling layer, and 1 fully-connected layer. The activation function used for the output is sigmoid, and the one used for each of the layers is ReLU (Figure 5).

Figure 5: Graph comparing Sigmoid and ReLU activation functions. Sigmoid spans from y=0 to y=1. ReLU spans from y=0 and has no upper limit. An output close to 1 for sigmoid would be considered "activated" while ReLU is without an real firing point and the larger the value, the stronger the activation. [7]

The gradient optimizer is ADAM which was chosen for its performance See Figure 6. I chose this architecture because it performed slightly better than the other architectures I was working with (e.g. 3 cov. layers, 3 max-pool. layers, 1 glob. pool. layer, and 2 dense layers).

Figure 6: Graph showing that the ADAM optimizer is most efficient. [8]

4 Experimental Results & Reflection

4.1 Results Tables

4.1.1 20th Epoch Performance Metrics

This table was created by taking the 20th epoch target metrics for each cross-validation fold in terms of Twentieth Century.

Loss:	Accuracy:	Precision:	Recall:
0.82	0.77	0.71	0.92
0.33	0.77	0.70	0.92
1.12	0.75	0.71	0.86
0.47	0.77	0.70	0.94
0.67	0.77	0.72	0.89
0.00	0.78	0.72	0.92
0.55	0.76	0.68	0.96
0.05	0.75	0.67	0.98
0.01	0.77	0.71	0.92
0.09	0.76	0.68	0.98

4.1.2 Confusion Matrix

a	b	<- classified as
617	433	a (18th Century)
82	969	b (20th Century)

4.1.3 Stratified Cross-Validation Results

Correctly Classified Instances: 1586 - 75.5%

Incorrectly Classified Instances: 515 - 24%

Total Number of Instances: 2101

4.1.4 Detailed Accuracy By Class

Precision	Recall	MCC	Class
0.883	0.588	0.541	18th
0.691	0.922	0.541	20th
0.787	0.755	0.541	Weighted Avg.

4.2 Results Graphs

Figure 7: Experimental results graphed vs. Cross-validation folds. Reported at the end of the training period for each fold and in terms of the 20th century.

5 Analysis & Conclusion

5.1 Evaluation Metric Explanations

The evaluation metrics reported for this investigation were chosen by the type of problem - binary classification. Thus F1 scores were not reported as they are not as accurate as the MCC in terms of evaluating the model's accuracy [9]. True positives are instances of the positive class (in this case, 18th century) that were correctly classified with the right label - 18th as 18th. False positives are instances of the positive class that were classified with the wrong label - so 18th as 20th. True and false negatives are just using the

negative class - 20th century. Precision is described as the ratio between the # of correctly identified instance for a class and the total number of instances that the model guesses to be in that class (so true positive + false positive or true negative and false negative). Precision is a measure of how often the model is correct in its prediction of an instance's class (18th or 20th). Precision is not very useful by itself, and here's why: Imagine you have a NN which is capable of identifying terrorists. Aside from the ethical problems of such a NN existing, technically, if the NN always guesses that a person is not a terrorist, they would be correct 99.9% of the time. However, this doesn't mean that the model is actually doing what it is supposed to - it doesn't correctly identify terrorist 99.9% of the time. This is where recall comes in. Recall is the ratio between the number of true classification for a class and the sum of the number of true and false identifications for that class. Recall thus provides us with a measure of how many instances of a class the model can classify correctly. How many terrorists can the model identify? 99.9%? Loss is the output of the error function. The last evaluation metric I chose to report is the Matthews Correlation Constant or MCC (Figure 8), which is regarded as a balanced measure of a binary classification model's performance. The MCC takes into account all four squares of the confusion matrix and ranges between -1 and 1. A value of 0 would mean performance no better than random, a value of 1 would mean perfect performance, and -1 would mean a model totally wrong in its predictions. With all four metrics, we can move on to the analysis.

Figure 8: The equation for the Mathews Correlation Coefficient.

5.2 Results Analysis

The first major thing I saw in my data was that 18th century had high precision but low recall, while 20th century had high recall but low precision. This means that the model more often correctly identified 20th century texts but was only correct 69% of the time when it actually ended up guessing 20th century. One might interpret this result as suggesting that since the model more often guessed 20th century, it was able to correctly identify more of the 20th century texts. This also suggests that the model was less hasty to guess 18th century rather than 20th century but was correct 88% of the time. This might be a result of the model, perhaps finding a linguistic feature common to most but not all 18th century text. Using this feature the model was almost certain that a text would be 18th century, but because this feature wasn't in all texts of the 18th century, it more often guessed 20th. Perhaps in the end, the CNN focused on the presence of a theme common to the 18th century but largely absent from the 20th. Since the MCC score is above 0 and is slightly above halfway to 1, I would conclude that this binary classifier is rather O.K - it has a strong correlation. But is this model viable?

5.3 Experimental Analysis & Limitations

In my testing, I only really found 2 distinctive configurations that consistently worked. I wasn't sure exactly why, but for a lot of configurations, the model didn't classify 20th century at all. I believe this is something to do with the CNN not finding useful patterns in the text however it might have had something to do with the configuration file of the model - because I was

using the non-GUI version of Weka, I didn't have all the tooltips and might have missed something which prevented the other configurations from working. My reported data was for the better of the two configurations which worked.

A major limitation for this project was the size of my dataset. Like I mentioned in my intro, Project Gutenberg doesn't have their books labeled by creation date. For this reason, I had to rely on an incomplete third party resource, which only ended up having the information for 2101 texts. 2101 texts still seems like a lot, but it's likely the reason the model couldn't perform better. If I had more texts, I would likely have better performance. Also, since my computer isn't powerful enough to process each of the texts in its full length, I've had to shorten texts, and this might have removed some valuable features the model would have otherwise used. Like I mentioned in my Results Analysis, the model likely found a pattern or theme that was absent from the 20th century but not from the 18th which allowed the model to make accurate predictions in terms of 18th century. If I had more data - and longer text - it would be easier for the model to find these features and the model would have likely had a greater performance See (Figure 9). With this experiment I had too much data for my computer to handle, but not enough to allow the model to perform at its best.

Figure 9: This graph shows that large NNs such as CNNs benefit most from more data.[10]

Because I used cross-validation, I am not worried about the possibility that my model overfit to the data and that my results are inflated. If you

look at the metrics at the 20th epoch and the metrics after the testing has occured (stratified), we see that there is a difference (especially in accuracy). This is likely due to the model overfitting (to a degree) for each epoch, but when stratified, we have a more accurate representation of its performance.

Even though the final accuracy of the model is only 75%, this model can still be useful and has real-life implications. Firstly, a model like this - perhaps trained with more data and able to predict the decade - could be applied by Project Gutenberg to reduce the amount of manual labor the volunteer organization needs to carry out. The model could assign each text a date of creation and then whoever accesses the text next time can check and make sure this date is correct or at least reasonable (if the actual date is unknown). The important thing is that this investigation proves that it is possible and worth pursuing (viable).

5.4 Conclusion

This investigation aimed at assessing the viablity of using character level CNNs for classifying text by century of creation. After carrying out an experiment and assessing the results, I am led to conclude that CNNs are indeed viable for this application, especially if they get enough data.

References

[1] Loblaw uses microsoft a.i. to break down silos and unlock new insights in the digital age. *Microsoft News Center Canada*, Jun 2019. `https://news.microsoft.com/en-ca/2019/06/11/`

loblaw-uses-microsoft-ai-to-break-down-silos-and-unlock-new-insights-in-the-digital-age/.

[2] Nikos Fakotakis Efstathios Stamatatos and George Kokkinakis. Automatic text categorization in terms of genre and author. *Computational Linguistics*, pages 471–495, Mar 2006.

[3] Neural networks for image recognition: Methods, best practices, applications. *MissingLink.ai*. https://missinglink.ai/guides/computer-vision/neural-networks-image-recognition-methods-best-practices-applications/.

[4] Sumit Saha. A comprehensive guide to convolutional neural networks. *Medium Journal*, Dec 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[5] Chm. Convolution operation - comprehensive guide. *Mc.ai*, Sep 2019.

[6] Yaron Meirovitch, Alexander Matveev, Hayk Saribekyan, David Budden, David Rolnick, Gergely Odor, Seymour Knowles-Barley, Thouis Jones, Hanspeter Pfister, Jeff Lichtman, and Nir Shavit. A multi-pass approach to large-scale connectomics. Dec 2016.

[7] Deep maxout neural networks for speec recognition - scientific figure on researchgate. *Research Gate*. avaliable at: https://www.researchgate.net/figure/Illustration-of-sigmoid-and-ReLU-nonlinearity_fig1_261309983 [accessed 2 Mar, 2020].

[8] Jason Brownlee. Gentle introduction to the adam optimization algorithm for deep learning. *Machine Learning Mastery*, Nov 2019. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[9] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21(1):6, 2020.

[10] Jesse Moore. Deep misconceptions about deep learning. *Medium Journal*, Jan 2018. https://towardsdatascience.com/deep-misconceptions-about-deep-learning-f26c41faceec.

6 Appendix

6.1 Command Line Arguments for Weka:

```
!/bin/bash
java -Xmx25G -cp ~/weka/weka.jar weka.Run .Dl4jMlpClassifier -t ~/CharLevelNominal
.arff \
        -S 1 -cache-mode MEMORY -early-stopping "weka.dl4j.earlystopping.EarlyStop
ping -maxEpochsNoImprovement 4 -valPercentage 0.0" -normalization "No normalizatio
n/standardization" -iterator "weka.dl4j.iterators.instance.ConvolutionInstanceIter
ator -height 1 -numChannels 1 -width 3740 -bs 1" -iteration-listener "weka.dl4j.li
stener.EpochListener -eval true -n 1" \
        -layer "weka.dl4j.layers.ConvolutionLayer -nFilters 25 -mode Truncate -cud
```

```
nnAlgoMode PREFER_FASTEST -rows 1 -columns 7 -paddingColumns 4 -paddingRows 0 -str
ideColumns 1 -strideRows 1 -nOut 10 -activation weka.dl4j.activations.ActivationRe
LU" \
        -layer "weka.dl4j.layers.SubsamplingLayer -mode Truncate -eps 1.0E-8 -rows
 2 -columns 2 -paddingColumns 0 -paddingRows 0 -pnorm 1 -poolingType MAX -strideCo
lumns 1 -strideRows 1" \
        -layer "weka.dl4j.layers.ConvolutionLayer -nFilters 25 -mode Truncate -cud
nnAlgoMode PREFER_FASTEST -rows 7 -columns 1 -paddingColumns 4 -paddingRows 0 -str
ideColumns 1 -strideRows 1 -nOut 10 -activation weka.dl4j.activations.ActivationRe
LU" \
                -layer "weka.dl4j.layers.SubsamplingLayer -mode Truncate -eps 1.0E
-8 -rows 2 -columns 2 -paddingColumns 0 -paddingRows 0 -pnorm 1 -poolingType MAX -
strideColumns 1 -strideRows 1" \
                -layer "weka.dl4j.layers.ConvolutionLayer -nFilters 25 -mode Trunc
ate -cudnnAlgoMode PREFER_FASTEST -rows 7 -columns 1 -paddingColumns 4 -paddingRow
s 0 -strideColumns 1 -strideRows 1 -nOut 10 -activation weka.dl4j.activations.Acti
vationReLU" \
        -layer "weka.dl4j.layers.ConvolutionLayer -nFilters 25 -mode Truncate -cud
nnAlgoMode PREFER_FASTEST -rows 7 -columns 1 -paddingColumns 4 -paddingRows 0 -str
ideColumns 1 -strideRows 1 -nOut 10 -activation weka.dl4j.activations.ActivationRe
LU" \
        -layer "weka.dl4j.layers.SubsamplingLayer -mode Truncate -eps 1.0E-8 -rows
 2 -columns 2 -paddingColumns 0 -paddingRows 0 -pnorm 1 -poolingType MAX -strideCo
lumns 1 -strideRows 1" \
                -layer "weka.dl4j.layers.GlobalPoolingLayer -collapseDimensions fa
```

```
lse -pnorm 2 -poolingType MAX" \
        -layer "weka.dl4j.layers.DenseLayer -nOut 10 -activation weka.dl4j.activat
ions.ActivationReLU" \
        -layer "weka.dl4j.layers.OutputLayer -lossFn \"weka.dl4j.lossfunctions.Los
sBinaryXENT \" -nOut 2 -activation \"weka.dl4j.activations.ActivationSigmoid \" -n
ame \"Output layer\"" \
        -logConfig "weka.core.LogConfiguration -append true " \
  -config "weka.dl4j.NeuralNetConfiguration -biasInit 0.0 -biasUpdater \"wek
a.dl4j.updater.Sgd -lr 0.001 -lrSchedule \\\"weka.dl4j.schedules.ConstantSchedule
-scheduleType EPOCH\\\"\" -dist \"weka.dl4j.distribution.Disabled \" -dropout \"we
ka.dl4j.dropout.Disabled \" -gradientNormalization None -gradNormThreshold 1.0 -l1
 NaN -l2 NaN -minimize -algorithm STOCHASTIC_GRADIENT_DESCENT -updater \"weka.dl4j
.updater.Adam -beta1MeanDecay 0.9 -beta2VarDecay 0.999 -epsilon 1.0E-8 -lr 0.001 -
lrSchedule \\\"weka.dl4j.schedules.ConstantSchedule -scheduleType EPOCH\\\"\" -wei
ghtInit XAVIER -weightNoise \"weka.dl4j.weightnoise.Disabled \"" -numEpochs 20 -nu
mGPUs 1 -averagingFrequency 10 -prefetchSize 24 -queueSize 0 -zooModel "weka.dl4j.
zoo.CustomNet "
```

6.2 Preprocessing Script Written in R:

```
library("qdap")
library("readr")
library(dplyr)
library(stringr)
library(tm)
library(MASS)
```

```
library(rlist)
list.save(df_3, "mylist.rds")
mylist <- list.load("mylist.rds")

#Get the file names of the texts
fileslist <- list.files(getwd(), full.names = T)
documents <- lapply(fileslist, function(x)readLines(x))

#Input the files into a dataframe
df <- lapply(documents, FUN = toString)
df <- data.frame(sapply(df,c), stringsAsFactors = FALSE)
df <- mutate(df, Target = fileslist)
colnames(df) <- c("Text", "Target")

#Remove whitespace, convert to alphanum, and remove whitespace again
df$Text <- str_replace(gsub("\\s+", " ", str_trim(df$Text)), "B", "b") %>% tolower()
df$Text <- str_replace_all(df$Text, "[^[:alnum:]]", " ")
df$Text <- str_replace(gsub("\\s+", " ", str_trim(df$Text)), "B", "b")

#Replace each character with a number
df_2 <- df_1
df_2 <- gsub("1", "_1_", df_2)
df_2 <- gsub("2", "_2_", df_2)
df_2 <- gsub("3", "_3_", df_2)
df_2 <- gsub("4", "_4_", df_2)
```

```
df_2 <- gsub("5", "_5_", df_2)
df_2 <- gsub("6", "_6_", df_2)
df_2 <- gsub("7", "_7_", df_2)
df_2 <- gsub("8", "_8_", df_2)
df_2 <- gsub("9", "_9_", df_2)
df_2 <- gsub("0", "_10_", df_2)

df_2 <- gsub("a", "_11_", df_2)
df_2 <- gsub("b", "_12_", df_2)
df_2 <- gsub("c", "_13_", df_2)
df_2 <- gsub("d", "_14_", df_2)
df_2 <- gsub("e", "_15_", df_2)
df_2 <- gsub("f", "_16_", df_2)
df_2 <- gsub("g", "_17_", df_2)
df_2 <- gsub("h", "_18_", df_2)
df_2 <- gsub("i", "_19_", df_2)
df_2 <- gsub("i", "_19_", df_2)
df_2 <- gsub("j", "_20_", df_2)
df_2 <- gsub("k", "_21_", df_2)
df_2 <- gsub("l", "_22_", df_2)
df_2 <- gsub("m", "_23_", df_2)
df_2 <- gsub("n", "_24_", df_2)
df_2 <- gsub("o", "_25_", df_2)
df_2 <- gsub("p", "_26_", df_2)
df_2 <- gsub("q", "_27_", df_2)
```

```
df_2 <- gsub("r", "_28_", df_2)

df_2 <- gsub("s", "_29_", df_2)

df_2 <- gsub("t", "_30_", df_2)

df_2 <- gsub("u", "_31_", df_2)

df_2 <- gsub("v", "_32_", df_2)

df_2 <- gsub("w", "_33_", df_2)

df_2 <- gsub("x", "_34_", df_2)

df_2 <- gsub("y", "_35_", df_2)

df_2 <- gsub("z", "_36_", df_2)

df_2 <- gsub(" ", "_37_", df_2)



#Fix double delimeter appearing twice

df_2 <- gsub("__", "_", df_2)


#Data frame formatting

#df$SourceFile <- gsub(".txt", " ", df$SourceFile)

df$Target <- gsub("wo", "", df$Target)

fileslist <- substr(fileslist, 1,nchar(fileslist)-1)


#To remove last element

df_3 <- df_2[-2102]


#Remove all delimeters

for (i in length(df_2)) {
```

```
  z <- max(sapply(df_3, function(x) length(x)))
  x <- z - length(df_3[[i]])
  y <- ""
  for (j in seq(x)) {
    y <- paste(y, "_", sep = "")
  }
  df_2[[i]] <- paste(df_2[[i]], y)
}

df_3 <- strsplit(df_2, split = "_")[]

for (m in seq(length(df_3))) {
  df_4 <- df_3[[m]][-1]
  df_4 <- df_4[-length(df_4)]
  #df_3 <- df_3[[m]][[length()]]

  write_lines(df_4, path = (paste(toString(fileslist[m]), ".txt")), append = T)
}

#Used for prior testing
document_text <- df$Text
#R needs to interpret each element of this vector to be a seperate document
document_text_source <- VectorSource(document_text)
document_text_corpus <- Corpus(document_text_source)
```

```
frequent_terms <- freq_terms(document_text, 3000)
plot(frequent_terms)

clean_corpus <- function(corpus){
  corpus <- tm_map(corpus, stripWhitespace)
  corpus <- tm_map(corpus, removePunctuation)
  corpus <- tm_map(corpus, content_transformer(tolower))
  corpus <- tm_map(corpus, removeWords, stopwords("en"))
  corpus <- tm_map(corpus, replace_ordinal)
  corpus <- tm_map(corpus, replace_number)
  corpus <- tm_map(corpus, replace_symbol)
  return(corpus)
}

clean_corp <- clean_corpus(document_text_corpus)
clean_corp <- tm_map(clean_corp, PlainTextDocument)

clean_corp[[127]][1]

text_dtm <- DocumentTermMatrix(tm_map(clean_corpus(Corpus(VectorSource(document_text))))
text_dtm_test <- removeSparseTerms(text_dtm, 0.95)
text_m<- mutate(data.frame(text_m), target = fileslist)

text_m <- as.matrix(text_dtm_test)
dim(text_m)
```

```
write.matrix(text_m, file = "95-505att.csv", sep = ",")
write.csv(df, file = "18-19.csv", sep = ",")

write.list(df_3, file = "output.txt", t.name = NULL )
```