**DALARNA UNIVERSITY**

**DT3020**

*Introduction to Artificial Neural Networks – 03 January 2018*

# Classifying Document Authorship Using Text Mining and Artificial Neural Networks

**Authors:**

| | |
|---|---|
| Amir Golkhari Baghini | h07amigb@du.se |
| Felipe Dutra Calainho | h16feldu@du.se |
| Marina Ferreira Uchoa | h16mferr@du.se |

# 1 Introduction

Authorship attribution problem has been considered and investigated during past decades by applying different methods and techniques[1]. According to [2], "author identification (or attribution) means attributing an unknown text to a writer basing on some feature characteristic or measure". This can be used when author of some written text is unknown or several authors claim for a written text. Stylometry is mainly applied for finding authors of published text, plagiarism detection and classifying spam or non-spam messages.

"Stylometry denotes quantitative analysis of some written text that yields information about the style it is composed with and through that about the author of this text" [2, p:151]. In the stylometric analysis two critical issues should be considered; applied analytical techniques and selection of stylistic features which consist of writing-style markers such as structural, lexical and idiosyncratic style [1].

The textual analysis procedure begins with training in which, the characteristics of selected features are computed by using texts with known authors. Then in the verification stage, unattributed texts with same descriptors are compared with previous results. Then among the set of available authors, the closest one is chosen [2]. Selected stylistic features should constitute a property of a text that belongs to the author. This means that this property should be similar in all texts of that author and meanwhile should be different from texts from different authors.

Usually stylometric problem can be analyzed by applying machine learning or statistic approaches. Statistical approaches are mainly used in Markovian Models, cluster analysis and Cumulative Sum. Machine learning approaches are carried out through applications such as Artificial Neural Networks (ANN), Support Vector Machine (SVM) and Rough Set Theory (RST) [2].

In machine learning approaches, for learning, two different methods such as supervised learning and unsupervised learning can be applied. In the context of stylometric analysis, for supervised learning, there is a need to have author-class labels for classification. Conversely, in unsupervised learning techniques, prior knowledge regarding authors' classes is not required.

In this project, the authorship identification or attribution was considered as a classification task. The aim was to classify written texts at the author level. The selected stylistic features were lexical which were used to describe texts. According to [2], lexical and syntactical features selection as texts' properties provide promising results because of its subconsciously impact on author's writing style and cannot be copied easily.

As it was mentioned, authorship attributions identifies the author with the highest probability among set of candidates. In this regard, according to [1], machine learning approaches provide better results in classification tasks as they are more efficient in dealing with huge data-set. Thus, in this project we tackled the problem by applying Artificial Neural Networks (ANN) as a machine learning approach. Also, the supervised learning technique by using author-class labels were applied.

The required data were collected from Kaggle, which contained 19579 examples extracted from horror books with three different authors. Each example was prepared by tokenizing a large text into a sentence and then labelled to a specific author. The data set contained three different authors, namely Edgar Allan Poe (EAP), Mary Wollstonecraft Shelley (MWS) and HP Lovecraft (HPL). Among the entire data set, 7900 examples were

assigned to EAP as author, 6044 examples to MWS and 5635 examples to HPL.

In order to solve the problem by applying ANN, the texts were transformed into numerical data set and this was done by creating a Term-Document Matrix (TDM). By this approach, the word's frequency in each sentence or documents. In the next Sections, we will discuss the characteristics of the data, the methodology and data processing and the results obtained.

# 2 Data

This project aims to create a neural network to identify three distinct authors of horror books, namely Edgar Allan Poe, Mary Wollstonecraft Shelley, and HP Lovecraft. The data containing labelled observations for the purpose of performing automatic classification is publicly available for non-commercial use from Kaggle (section 7.A of the competition rules).

The data set is constituted of 19579 examples of labelled sentence excerpts from books written by Edgar Allan Poe, Mary Shelley and HP Lovecraft, respectively. This data is unbalanced, Edgar Allan Poe having 7900 examples (40%), Mary Shelley with 6044 excerpts (31%) and HP Lovecraft with 5635 examples (29%). Each example is constituted of a sentence and was created using CoreNLP's MaxEnt sentence tokenizer.

# 3 Methodology and Data Processing

In order to use the neural network toolbox for pattern recognition from Matlab, the textual data had to be processed and transformed into a numeric data set. Since the Text Analytics toolbox for Matlab is not currently provided by Dalarna University, the pre-processing of data had to be performed in a distinct software. For this purpose, R [3] was used, as all three authors are familiar with the software and could equally contribute to the development of the project.

In order to process textual data, Natural Language Processing (NLP) has to be addressed. The term NLP refers to the set of mathematical and computational techniques used for analyzing, representing and/or producing textual content in written or oral form in one or more human languages [4, 5, 6]. The scope of the analysis ranges from the bag-of-words approach to a full NLP incorporating phonetic, morphological, lexical, syntactic, semantic, discourse and pragmatic levels [7, 5].

Bag-of-words is the simplest approach for processing textual input. It performs the analysis only at the word level, disregarding word order and grammatical relations between the elements in sentences, paragraphs and entire texts [8]. For this approach a Term-Document Matrix (TDM) is the representation of the frequency of the words in a set of documents.

Sharda, Delen and Turban emphasize that text mining is performed by following three main steps [8]. The first is to gather data and build a textual corpus. The second is to create the term-document matrix (TDM) with word frequencies across documents. The third step is to actually extract knowledge from the data.

In the current project, data was readily available on the internet and the corpus, as well as the TDM were built using R. This pre-processing phase could have been done in

Matlab, however, the necessary toolbox was not available at the University and the trial version would expire during the development of the work and could not be installed in the computers at the University laboratory, where most of the work was performed.

The pre-processing that was performed in R was composed of the following steps:

1. remove punctuation and other non-alphanumeric, non-space characters;
2. transform all characters into lower case;
3. remove English stop words;
4. stem words using Porter's stemming algorithm; and
5. remove extra spaces resulting from the previous processes.

From all these steps stemming is the only optional process to allow the analysis of textual data. It is a term normalization process. Porter stemming algorithm strips the words of common morphological and inflexional endings, reducing them to a simplified version. Below are the same sentences before textual pre-processing, after pre-processing but without stemming and after the complete pre-processing (after all steps):

> In his left hand was a gold snuff box, from which, as he capered down the hill, cutting all manner of fantastic steps, he took snuff incessantly with an air of the greatest possible self satisfaction.
>
> left hand gold snuff box capered hill cutting manner fantastic steps took snuff incessantly air greatest possible self satisfaction
>
> left hand gold snuff box caper hill cut manner fantast step took snuff incess air greatest possibl self satisfact

As can be seen, stemming is a process that reduces the intelligibility of the text for humans. However, in terms of machine learning, it enhances the perception of the similarity between texts by allowing similar terms in different forms (present versus past tense of the same verb, for example) to be represented as equals. A distinct approach for reducing inflectional forms is lemmatization. Contrary to stemming, lemmatization makes use of vocabulary and morphological analysis of words to reduce them to their base form, their *lemma*. Even though the results from lemmatization are more comprehensible to humans, the authors did not find an automated way to perform it in R and thus opted to perform the stemming.

A second consequence of stemming is the reduction of the number of elements in the TDM and, hence, its dimensionality. Using our data set, the TDM using the full words had 492,216,060 elements (25140 words by 19579 documents) taking up a total memory space of 3.7 Gb, whereas the TDM with stemmed words had 293,939,527 elements (15013 words by 19579 documents), taking up 2.2 Gb. The amount of words to be analyzed and processed was reduced by over 10000, implying a much quicker analysis.

Initially, the process of pre-processing the data in R, importing it to Matlab and running the neural network was done with a reduced subset of 10 examples from each author,

with a total of 30 randomly selected observations. This process went smooth, however, once the TDM with the complete stemmed data set was saved, the authors tried to import the data to Matlab but did not succeed. Hence, to further reduce the dimensionality of the TDM, documents with less than or only 10 words after the pre-processing phase were removed. The authors support that these documents would add noise in the learning process of the neural network, since their contribution to the identification of the authors would be minimal but they have high impact on the sparseness of the TDM.

Of the total 19579 documents, 9091 had 10 or less words, meaning that the final dataset is composed of 10488 examples. These contain 13935 words, hence a reduction of 46% in the number of examples is associated with a reduction in 7% (1078) of the words. The final TDM has 146,150,280 elements (13935 words by 10488 documents). From the total amount of documents, Edgar Allan Poe has 3650 examples (34.8%), Mary Shelley has 3181 excerpts (30.33%) and HP Lovecraft has 3657 examples (34.87%). For importing into Matlab, the TDM was saved in a csv file. However, Matlab once again gave an error message when we tried to import the data.

Taking into consideration that dimension reduction would imply networks with fewer nodes and, thus, a less complex error space and also facilitate importing the data into Matlab, the authors opted for performing a Principal Component Analysis (PCA). The PCA permits to transform the number of features (words) into Principal Components (PCs) ordered in terms of their significance, which can then be selected according to a threshold of how well the data is to be represented by the PCs. The PCA was also performed in R. Its outputs were saved into a csv file, which was then successfully imported into Matlab.

Once in Matlab, the data was divided into testing and training data sets. The proportion for training was of 70% of the data, which was extracted in equal proportions from the observations regarding each author. This is, 70% of the examples for Edgar Allan Poe, for Mary Wollstonecraft Shelley and for HP Lovecraft were randomly selected to compose the training data set. The remaining examples (30%) were assigned to the test data set. The final data sets have 7340 documents for training and 3148 for testing. To ensure that the examples would be presented to the neural network in a random order, the function `randperm` was used to perform a random permutation of the elements in both data sets.

In order to tune the results of the neural network, the transfer function as well as the number of hidden layers were varied. Matlab has 15 built-in transfer functions which can be visualized by calling the command `help nntransfer`. These functions are displayed in Table 1.

Since not all of these functions have been studied in the course and performing an extensive analysis with all transfer functions was not viable due to time constraints, the authors opted for performing the analysis with four transfer functions, namely: Symmetric sigmoid (tansig), Elliot sigmoid (elliotsig), Logarithmic sigmoid(logsig) and Radial basis (radbas). To the best of our knowledge, the other basic characteristics of the network which are tunable are the training function and the number of epochs.

In a more complex level, it is possible to transform open into closed feedback loop and the other way around, add or remove a delay to the neural network response, to allow the network to adapt, to change the learning and distance functions, for example. A detailed list of all these options can be found by inputting `help nnet` into the Matlab command window. Due to time constraints, the authors used only the Scaled Conjugate Gradient

Table 1: Matlab Built-in Transfer Functions

| Code | Function |
|---|---|
| compet | Competitive transfer function |
| elliotsig | Elliot sigmoid transfer function |
| hardlim | Positive hard limit transfer function |
| hardlims | Symmetric hard limit transfer function |
| logsig | Logarithmic sigmoid transfer function |
| netinv | Inverse transfer function |
| poslin | Positive linear transfer function |
| purelin | Linear transfer function |
| radbas | Radial basis transfer function |
| radbasn | Radial basis normalized transfer function |
| satlin | Positive saturating linear transfer function |
| satlins | Symmetric saturating linear transfer function |
| softmax | Soft max transfer function |
| tansig | Symmetric sigmoid transfer function |
| tribas | Triangular basis transfer function |

Backpropagation as the training function along with a cross-entropy performance measure.

# 4 Data analysis and results

In the TDM the words are the features for each pattern, or document. Having 13935 features as inputs for the Neural Network (NN) is unsuitable. This increases the complexity of the network and might have a negative impact on the network's performance. This happens because adding one input feature, thus, one passive node in the input layer increases the amount of weights and, therefore, the error space complexity by at least the total number of neurons in the hidden layer. In the present case, since backpropagation training was used, the increase is twice the referred amount.

As an example, a network with 10 neurons in the hidden layer, 3 output neurons and 7 input features has 113 weights, while by increasing the number of input features to 1000 forces an increase to 10043 weights and an increase to 2850 features, to 28543 weights. Also, the more complex neural network demands a high processing rate from the CPU and a high RAM memory storage, incurring in long processing time and potential memory availability errors. Based on the above mentioned conditions a PCA was performed for a feature reduction of the TDM. The analysis and its results will be discussed in the following subsection.

## 4.1 Principal Component Analysis

The PCA was performed in a computer with a Intel i5-7600 processor with 3.5GHz, 16 GB of RAM and a 64 bits operating system. The PCA for the 1.1 GB TDM took this computer 4604.55 seconds to run, that is approximately 77 minutes. After performing the PCA in the TDM, 10488 PCs were generated. This is because the number of documents is smaller than the number of words in the TDM, hence the dimensionality of the PCs is equal to the number of documents (10488). In order to decide which PCs should be taken further for model building, Figure 1 helps identify which PCs possess the most variability from our data.

Based on Figure 1, it seems that the first 6 PCs have a higher variability than the others, suggesting that the other PCs could be dismissed. However, based on the variance value for each PC, this is a shallow analysis. Since the value of the variance is very low (bellow 1), those first 6 PCs might not sufficiently represent the original dataset (TDM). Therefore, the cumulative importance of the PCs was checked. PCs 1 to 7 had a cumulative importance of 0.04, meaning that the first 7 PCs only explained 4% of the TDM. PCs 1 to 1000 had a cumulative importance of 67% and PCs 1 to 2850 had a cumulative importance of 90%.

Motivated by the above mentioned results, it was decided to test if there was any significant difference in performance between the use of 7, 1000 or 2850 PCs for multiple classification. This decision was made because the increase in inputs implies an increase in the complexity of the NN. Therefore, likely compromising the NN performance for this task even though more information with regard to the actual data would be passed into the network.

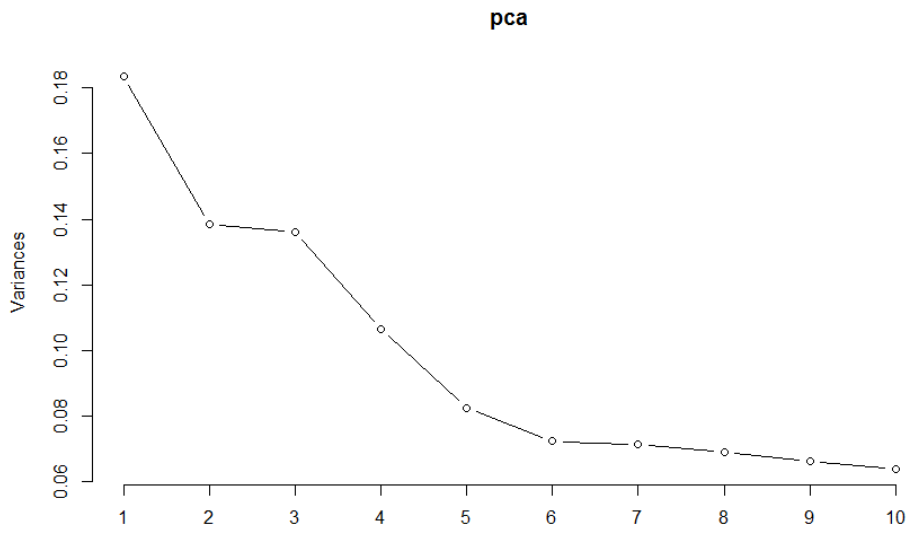



Figure 1: PCs Variances

## 4.2 Neural Network

The NN was feeded with three different datasets, each one regarding to the number of PCs. Hence, a Overall Performance Table for each dataset. In order to optimize the NN several Hidden Layer Sizes (HLS) and Transfer Functions (TFs) were tested, as mentioned in the Methodology. Running the optimization code that tested the different HLSs and TFs for the 7 PC data set (1.4 MB) took less than 1 hour to run, on the above mentioned computer. The 1000 PC data set (197 MB) took almost 3 hours, and the 2850 PC data set (567 MB) took 9 hours. It is important to note that, loading all datasets into Matlab took about 1 hour and 40 minutes to load. Therefore, the analysis was very time consuming and restrained our tuning possibilities.

Tables 2 to 6 show the Percentage Error obtained for each dataset with the different HLSs and TFs. The Percentage Error is a proportional average between the Training Error and the Testing error, in this case 0.7 for training and 0.3 for testing. This is not the ideal standard to measure the NNs performance, but the authors had no better option to use, being limited by their proficiency in Matlab. For all combinations of HLS and TF tested the confusion matrix plot for both Training and Testing were saved. These display the Training and Testing accuracy and errors, and follow along with this report. In the corpus of this report, we will only discuss the confusion matrix for the best performing HLS and TF from each dataset.

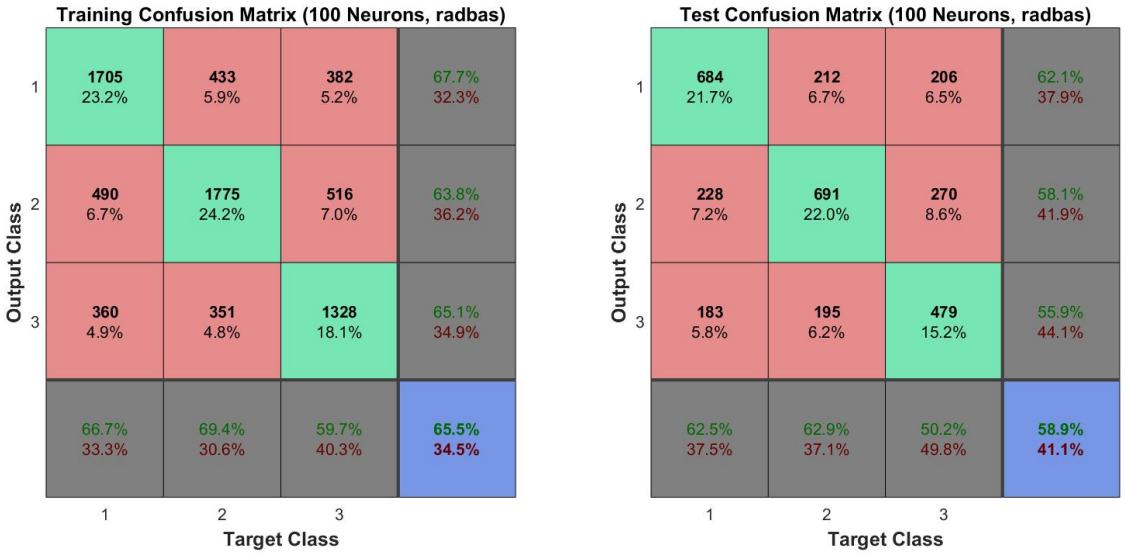
In the following cofusion matrix plots, 1 represents Edgar Alan Poe, 2 represents HP Lovecraft and 3 Mary Shelley.

### 4.2.1 7 PCs

For the data set containing 7 PCs, the best result was obtain by using 100 neurons and the radial base function as the Transfer Function in the Hidden Layer. The Training error

was 34.5% (Figure 2a), the Test error was 41.1% (Figure 2b), therefore the percentage error was 0.3648 as can be seen in Table 3. The average error for all trials in 7 PCs dataset was 38.69%. For this data set no overfitting was observed as the NN did not converge in any of Training trials, hence low accuracy results for all trials using this 7PCs dataset. Overall this was the worst result from all PCs datasets, as will be seen further. This was expected giving in consideration 7 PCs only explained 4% of the original dataset.

Figure 2: Best Result for 7 PCs with 100 neurons in HL and Radial Basis TF




(a) Confusion Matrix for Training

(b) Confusion Matrix for Test

### 4.2.2 1000 PCs

For the dataset containing 1000 PCs, the best result was obtained using 100 neurons and the Hyperbolic tangent sigmoid transfer function in the Hidden Layer. The Training error was 0% (Figure 3a), the Test error was 22.2% (Figure 3b) and the percentage error was 6.66% as can be seen in Table 4. The average error for all trials in 1000 PCs dataset was 8.11%. All training sets converged thus, obtaining 100% of accuracy. This might incur in overfitting in the test set. Additionally, this implies that all variations in percentage errors in Table 4 regard the differences in testing performance.

There is a difference in performance between training and testing. It is expected that the performance in training be higher than the performance in testing. Depending on this difference it might indicate a overfitting problem. In the present case the error decreases as the number of neurons in the hidden layer increases. This suggests, that there is no presence of overfitting in the NN.

The 1000 PCs explained 67% of the original dataset and so was expected that the NN using this dataset was going to outperform the 7 PCs dataset. The 1000 PCs dataset provided the best performing NN as it will be presented below.

Table 2: Overall Performance - 7 PCs

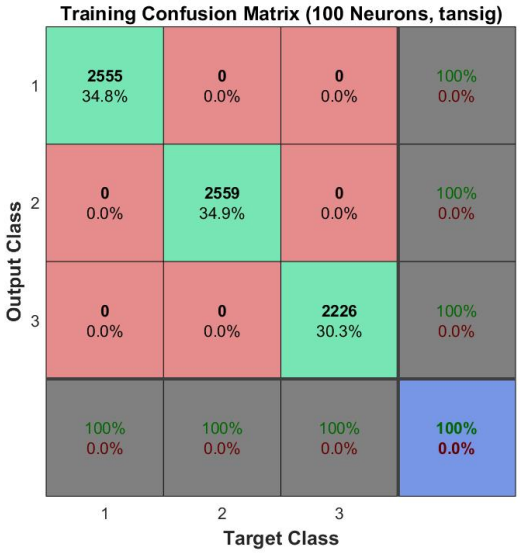
| Transfer Function | Hidden Layer Size | Percentage Error |
|---|---|---|
| elliotsig | 3 | 0.40542 |
| | 6 | 0.39865 |
| | 9 | 0.38787 |
| | 13 | 0.39445 |
| logsig | 3 | 0.41342 |
| | 6 | 0.40036 |
| | 9 | 0.38902 |
| | 13 | 0.38768 |
| radbas | 3 | 0.41743 |
| | 6 | 0.4077 |
| | 9 | 0.39092 |
| | 13 | 0.38368 |
| tansig | 3 | 0.4078 |
| | 6 | 0.39693 |
| | 9 | 0.3872 |
| | 13 | 0.3873 |

#### 4.2.3 2850 PCs

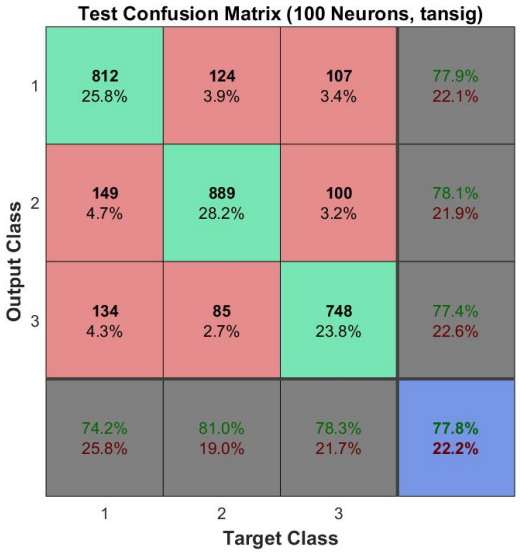
For the dataset containing 2850 PCs, the best result was obtained using 2000 neurons and the Radial Basis transfer function in the Hidden Layer. The Training error was 0% (Figure 4a), the Test error was 33.4% (Figure 4b) and the percentage error was 10.03% as can be seen in Table 5. The average error for all trials in 2850 PCs dataset was 11.04%. Only one of the trials in training set did not converge, namely the one with 40 neurons and Radial Base TF in the hidden layer. This training trial had a 99.5% of accuracy.

The 2850 PCs explained 90% of the original dataset and so was expected that the NN using this dataset was going to outperform all the others datasets. However, it turned out that this data set had a worst performance compared to the dataset using 1000 PCs. This decrease in performance might be explained partially by the fact that the NN using 2850 PCs has more weights, thus being more complex than the other two. To take in consideration, the best performing NN for this dataset had 5,708,003 weights versus 2,008,003 weigths for the 1000 PCs with 2000 neurons in the HL (numWeightsElements from `disp(net)`).

Figure 3: Best Result for 1000 PCs with 100 neurons in HL and Hyperbolic Tangent Sigmoid TF



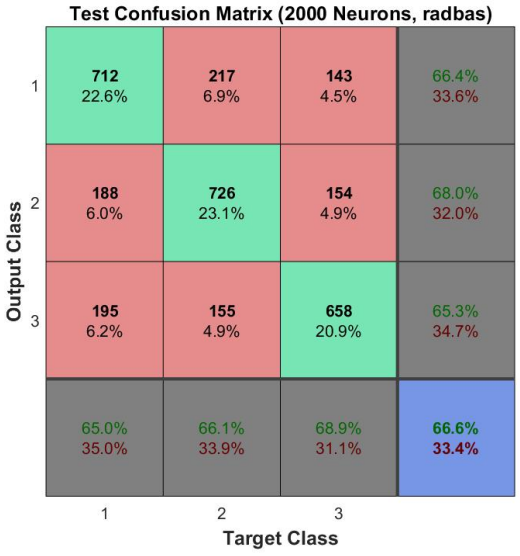

(a) Confusion Matrix for Training




(b) Confusion Matrix for Test

Figure 4: Best Result for 2850 PCs with 2000 neurons in HL and Radial Basis TF


(a) Confusion Matrix for Training




(b) Confusion Matrix for Test

Table 3: Overall Performance 2 - 7 PCs

| Transfer Function | Hidden Layer Size | Percentage Error |
|---|---|---|
| elliotsig | 10 | 0.3954 |
| | 20 | 0.38501 |
| | 30 | 0.38139 |
| | 40 | 0.38339 |
| | 50 | 0.38539 |
| | 60 | 0.3751 |
| | 70 | 0.36861 |
| | 80 | 0.37433 |
| | 90 | 0.37557 |
| | 100 | 0.37166 |
| logsig | 10 | 0.38959 |
| | 20 | 0.38234 |
| | 30 | 0.3853 |
| | 40 | 0.37757 |
| | 50 | 0.37548 |
| | 60 | 0.37662 |
| | 70 | 0.37433 |
| | 80 | 0.37281 |
| | 90 | 0.36985 |
| | 100 | 0.371 |
| radbas | 10 | 0.39159 |
| | 20 | 0.38196 |
| | 30 | 0.37576 |
| | 40 | 0.37815 |
| | 50 | 0.37185 |
| | 60 | 0.3688 |
| | 70 | 0.36737 |
| | 80 | 0.36728 |
| | 90 | 0.36613 |
| | 100 | 0.3648 |
| tansig | 10 | 0.38349 |
| | 20 | 0.38186 |
| | 30 | 0.37977 |
| | 40 | 0.37796 |
| | 50 | 0.37548 |
| | 60 | 0.375 |
| | 70 | 0.37157 |
| | 80 | 0.37233 |
| | 90 | 0.3688 |
| | 100 | 0.37176 |

Table 4: Overall Perfomance - 1000 PCS

| Transfer Function | HiddenLayer Size | Percentage Error |
|---|---|---|
| elliotsig | 10 | 0.089626 |
| | 20 | 0.081998 |
| | 30 | 0.08915 |
| | 40 | 0.082189 |
| | 50 | 0.083429 |
| | 60 | 0.078471 |
| | 70 | 0.076468 |
| | 80 | 0.081522 |
| | 90 | 0.082285 |
| | 100 | 0.078852 |
| logsig | 10 | 0.090961 |
| | 20 | 0.085336 |
| | 30 | 0.083905 |
| | 40 | 0.077803 |
| | 50 | 0.073989 |
| | 60 | 0.079043 |
| | 70 | 0.078757 |
| | 80 | 0.07151 |
| | 90 | 0.079805 |
| | 100 | 0.075706 |
| radbas | 10 | 0.10059 |
| | 20 | 0.090103 |
| | 30 | 0.086766 |
| | 40 | 0.084668 |
| | 50 | 0.080187 |
| | 60 | 0.080854 |
| | 70 | 0.08238 |
| | 80 | 0.083524 |
| | 90 | 0.084287 |
| | 100 | 0.077899 |
| tansig | 10 | 0.082094 |
| | 20 | 0.081998 |
| | 30 | 0.081712 |
| | 40 | 0.077613 |
| | 50 | 0.084382 |
| | 60 | 0.071892 |
| | 70 | 0.082761 |
| | 80 | 0.073513 |
| | 90 | 0.070271 |
| | 100 | 0.066648 |

Table 5: Overall Performance - 2850 PCS

| Transfer Function | Hidden Layer Size | Percentage Error |
|---|---|---|
| elliotsig | 10 | 0.10555 |
| | 20 | 0.1086 |
| | 30 | 0.11175 |
| | 40 | 0.11318 |
| | 50 | 0.1127 |
| | 60 | 0.1086 |
| | 70 | 0.10917 |
| | 80 | 0.10898 |
| | 90 | 0.11127 |
| | 100 | 0.10898 |
| logsig | 10 | 0.10174 |
| | 20 | 0.10469 |
| | 30 | 0.10126 |
| | 40 | 0.10707 |
| | 50 | 0.11098 |
| | 60 | 0.11404 |
| | 70 | 0.10793 |
| | 80 | 0.1066 |
| | 90 | 0.10603 |
| | 100 | 0.10669 |
| radbas | 10 | 0.12471 |
| | 20 | 0.10412 |
| | 30 | 0.10908 |
| | 40 | 0.16924 |
| | 50 | 0.11041 |
| | 60 | 0.11318 |
| | 70 | 0.10555 |
| | 80 | 0.10669 |
| | 90 | 0.10574 |
| | 100 | 0.10908 |
| tansig | 10 | 0.10431 |
| | 20 | 0.1087 |
| | 30 | 0.10498 |
| | 40 | 0.10755 |
| | 50 | 0.10402 |
| | 60 | 0.10841 |
| | 70 | 0.10698 |
| | 80 | 0.10688 |
| | 90 | 0.10946 |
| | 100 | 0.1045 |

Table 6: Overall Performance 2 - 2850 PCS

| Transfer Function | Hidden Layer Size | Percentage Error |
|---|---|---|
| elliotsig | 250 | 0.11508 |
| | 500 | 0.11737 |
| | 800 | 0.11384 |
| | 1100 | 0.10898 |
| | 1400 | 0.10965 |
| | 1700 | 0.11404 |
| | 2000 | 0.10946 |
| logsig | 250 | 0.11089 |
| | 500 | 0.11527 |
| | 800 | 0.11079 |
| | 1100 | 0.10774 |
| | 1400 | 0.10917 |
| | 1700 | 0.10278 |
| | 2000 | 0.10612 |
| radbas | 250 | 0.10698 |
| | 500 | 0.11899 |
| | 800 | 0.10812 |
| | 1100 | 0.11137 |
| | 1400 | 0.10755 |
| | 1700 | 0.11127 |
| | 2000 | 0.10031 |
| tansig | 250 | 0.11527 |
| | 500 | 0.10564 |
| | 800 | 0.10765 |
| | 1100 | 0.11003 |
| | 1400 | 0.10593 |
| | 1700 | 0.10564 |
| | 2000 | 0.10631 |

# 5 Final considerations

While performing this study the major difficulties faced by the authors were the dimensionality of the data together with the memory and processing capabilities of the computers used. This culminated in a very time consuming pre-processing of the data as well as for running the NN algorithm.

There were very long computations, often having errors occured after more than one hour running the algorithm. Taking into consideration this is a three party project, conciliating the tight time schedules with the demand for this project was a big challenge. Summing up the processing time taken, not accounting the time wasted with errors after processing begun and the time employed to write the code, the total duration was over 16h. However, we greatly deepened our knowledge in Text Mining, PCA and NN. Also, we further developed our Matlab programming skills and gained more familiarity with big data hurdles.

For a further work the authors would like to perform a cross-validation on the training sets to better access model's performance and robustness. Also, there are many NN parameters to be optimized, such as trying an adaptive NN, changing the training and distance functions, for example. Furthermore, the models discussed in this work are the best models regarding percentage errors, they do not take in consideration the time spent to run them. However, throughout this project the authors noted that some transfer functions were faster to train than others. Thus, an analysis encompassing the tradeoff between performance and processing time for different transfer functions (specially for the 2850 PCs dataset) could provide a better insight.

# References


[1] LZ Wang. News authorship identification with deep learning, 2017.

[2] Urszula Stańczyk and Krzysztof A Cyran. Machine learning approach to authorship attribution of literary texts. *International Journal of Applied Mathematics and Informatics*, 1(4):151–158, 2007.

[3] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.

[4] James F. Allen. Natural language processing. In *Encyclopedia of Computer Science*, pages 1218–1222. John Wiley and Sons Ltd., Chichester, UK, 2003.

[5] Elizabeth D. Liddy. Natural language processing. In *Encyclopedia of Library and Information Science*. Marcel Decker Inc, NY, 2nd edition, 2001.

[6] Aravind K. Joshi. Natural language processing. *Science*, 253(5025):1242–1249, 1991.

[7] Gobinda G. Chowdhury. Natural language processing. *Annual Review of Information Science and Technology*, 37(1):51–89, 2003.

[8] Ramesh Sharda, Dursun Delen, and Efraim Turban. *Business Intelligence and Analytics: Systems for Decision Support*. Pearson, 10th edition, 2014.