Neural Network Initialized with a Decision Tree for Document Categorization

Ethan Fulton, Dr. Girard

Abstract - This paper outlines two methods of using machine learning for document classification (Decision Trees and Neural Networks) and proposes an experiment to test the accuracy of a Neural Network Initialized with a Decision Tree (NNIDT) with the results in "Deep Belief Networks for Document Classification" by Justin Rebok [1]. First, TF-IDF and stemming will be described for the information preprocessing stage, then the C4.5 Decision Tree model will be outlined as well as a description of Neural Networks and how to train them using Back Propagation. Third, methodology will be presented for building a NNIDT. The experiment proposed will test three different samples sizes from the BBC dataset using a 90:10 training to testing ratio. First, the NNIDT will be tested without the use of stemming, then stemming will be introduced to see how reduced input affects the accuracy.

I. Introduction

Technology can be very helpful with the organization of documents into specific categories by using algorithmic learning based on training examples to predict where documents should go with reasonably high precision. When it comes to understanding the use of machine learning to categorize documents and web articles, there are two encapsulating concepts that need addressed. The first concept is information retrieval, which includes the process of extracting important information from a set of data. The method used in this experiment will be Term Frequency – Inverse Document Frequency (TF-IDF). Processes like stemming can also be used in tandem with these methods to condense a data set. The second concept is information processing, which uses the retrieved information and runs it through algorithms to determine the proper category. Two general ways of doing this are Neural Networks trained with back propagation, which include an input, output, and series of hidden layers to properly determine a category, and decision trees, which use a divide and conquer technique based on information gain to build a tree and select which category a document belongs to. Our experiment will use a combination of a Neural Network and Decision Tree.

II. Information Retrieval

Information retrieval can be defined as "Finding material of an unstructured nature that satisfies an information need from within large collections." [2]. In the case of document categorization, the information that needs retrieved is important words from a document. Some implementations consider word order and semantics, whereas others ignore these and use a bag-of-words technique that just pulls relevant words based on frequency. Both methods have their uses, but the categorization methods discussed in this paper will focus on the bag-of-words method.

A. Term Frequency – Inverse Document Frequency

Another method for extracting important words from a document is Term Frequency - Inverse Document Frequency (TF-IDF). This method calculates a weight for each word by multiplying the number of times a word appears in a single document by the logarithm of the inverse of the number of

documents that word appears in a set. TF-IDF is defined mathematically in Figure 2.

This weight is highest when the term t occurs many times in a small number of documents and lowest when t occurs in virtually all documents [2]. Thus, words with a weight within a predetermined limit can be used to determine which words are important to a document. If one wanted to apply this methodology to find important words in a category rather than a document, one could consider all the documents in that category as one, so every category becomes essentially a document, and the set of categories becomes

$$TF - IDF = tf_{t,d} \times \log \frac{N}{df_t}$$

Where N is the total number of documents in the collection, $tf_{t,d}$ is the term frequency and df_t is document frequency [2].

the set of documents. For example, if a word appears five times in a single document, which is in a series of five documents, then the TF-IDF score for that word would be $5 \times log \frac{5}{1} = 3.4948$.

Figure 2 Formaula for calculating TF-IDF.

B. Stemming

Due to the complexity of natural language, specifically English, in an approach that utilizes the bag-of-words method there is a lot of noise when it comes to term frequency. One method used to condense the amount of information pulled from a document is the use of a stemming algorithm. Stemming refers to "reducing inflectional forms and sometimes derivationally related forms of a word to a common base form." [2]. This is a useful tool, especially when it comes to training neural networks, as it simplifies the input. One of the most common algorithms for stemming is Porter's algorithm, which uses a series of five phases to remove suffixes from a word.

In Porter's algorithm, consonants and vowels in a word are grouped represented by a *C* and a *V* respectively. For example, the word *Private* would be represented *CVCVCV*. Once the word is represented in this form, a series of predetermined rules about patterns in the English language are used to determine how the ending of the word should be changed. For example it would reduce *replacement* to *replac* but not *cement* to *c* [2]. Often a stemmer is not as accurate as an algorithm that returns the lemma of a word, but it is easier to implement as it doesn't require a full vocabulary and is usually good enough because creates an equivalence class for the stemmed words.

III. Information Processing

Once the proper words have been picked to represent a document they can be fed into an algorithm which can then determine, based on those words, which category the document belongs to. There are two common models used in machine learning to achieve this kind of result; Back Propagated Neural Networks (BPNN) and Decision Trees. The BPNN method takes a longer time to train but performs better in the presence of "noisy data", whereas decision trees run significantly faster, but with a lower success rate [4].

A. Back Propagated Neural Networks

A neural network is a series of nodes that connects some input to some output. The simplest version of this would be a two-layer network that connects the input directly to the output, however this model is basically useless for document categorization. Instead, several hidden layers are added between the input and the output to allow more dimensionality to better solve the problem. Each of the nodes in the hidden layers of the network have an associated activation and

some predetermined bias, and all the connections between nodes have some weight associated to them (typically initialized randomly). The goal of back propagation is to modify the weights and/or activations in order to lower the cost function, or the difference between the output the network gave and the desired output, this is how the network is trained [5]. Back propagation thus consists of two phases:

During the first phase the input is presented and propagated forward through the network to compute the output value o_{pj} for each unit. This output is then compared with the targets, resulting in an error signal δ_{pj} for each output unit. The second phase involves a backward pass through the network (analogous to the initial forward pass) during which the error signal is passed to each unit in the network and the appropriate weight changes are made [6].

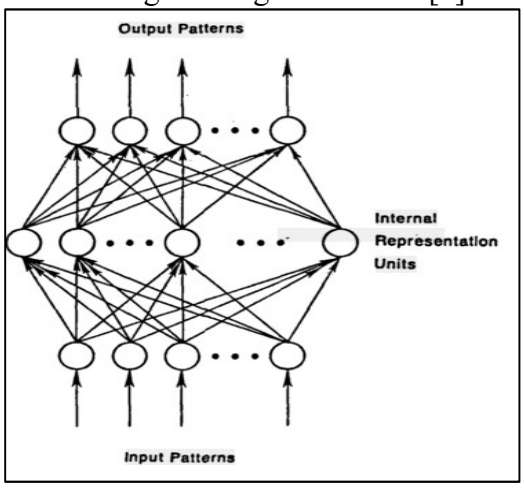

Figure 1 An example of a neural netowrk with a single hidden layer. [7]

To implement the back-propagation algorithm, four equations are needed. First, the weighted input of each hidden node can be defined as

$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l,$

where w_{jk}^l is the weight between the k^{th} node in the $l-1^{th}$ layer and the j^{th} node in the l^{th} layer, a_k^{l-1} is the activation in the $l-1^{th}$ layer and b_j^l is the bias of the j^{th} node in the l^{th} layer. This weighted input can then be used to find the activation,

$a_j^l = \sigma(z_j^l).$

The cost of a single training example x with y being the desired output can then be defined as:

$C = \frac{1}{2}\sum_j (y_j - a_j^L)^2.$

Finally, the error node j in the l^{th} layer with respect to the $l+1^{th}$ layer:

$\delta_j^l = \sum_k w_{kj}^{l+1}\delta_k^{l+1}\sigma'(z_j^l).$

The back-propagation algorithm can then be defined as the following [5]:

1) **Input** x:
   Set the corresponding activation a^1 for the input layer
2) **Feedforward**:
   For each $l = 2, 3, \ldots, L$ compute z^l and a^l
3) **Output Error** δ^L:
   Compute $\delta^L = \frac{\partial C}{\partial a_j^L}\sigma'(z^L)$
4) **Output**:
   The gradient of the cost function $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$
5) **Modify Weights**:
   New weight = old weight – (learning rate)$(\frac{\partial C}{\partial w_{jk}^l})$

For example, consider a simple network with two input nodes, a single hidden layer with one node, and an output layer with two nodes. The activation for the input layer here will be arbitrarily set to 0.3 for one node and 0.4 for the other, the output wanted is 1 on one node and 0 on the other. All weights will be set to 0.5 and the bias will be 1. The weighted input of the hidden node can then be computed as $z^1 = (0.5 \times 0.3) + 1 + (0.5 \times 0.4) + 1 = 2.35$, and the activation can be computed as $a^1 = \sigma(2.35) = 0.9$. In the output layer there will be two nodes with activations calculated the same way,

$z_1^2 = (0.5 \times 0.9) + 1 = 1.45$ $a_1^2 = \sigma(1.45) = 0.8$

$z_2^2 = (0.5 \times 0.9) + 1 = 1.45$ $a_2^2 = \sigma(1.45) = 0.8$

Error for the output layer is computed as such:

$$\delta_1^2 = (0.8 - 1) \times \sigma'(1.45) = -0.2 \times 0.2 = -0.04$$

$$\delta_2^2 = (0.8 - 0) \times \sigma'(1.45) = 0.8 \times 0.2 = 0.16$$

Now the rate of change of the cost function with respect to each weight coming into the output layer can be computed:

$\frac{\partial C}{\partial w_{j1k1}^l} = 0.9 \times -0.04 = -0.036$ and $\frac{\partial C}{\partial w_{j2k1}^l} = 0.9 \times 0.16 = 0.144$.

Using these changes, the weights can be adjusted by subtracting from the original weight the rate of change of the cost function with respect to that weight multiplied by the learning rate. Once the weights have been modified, error can be computed for the previous layer, and the process repeats until the input layer is reached. At this point, all weights have been modified and back propagation for this training example is complete.

B. Decision Trees

A common method used in document classification is the C4.5 model decision tree. A decision tree uses a divide and conquer technique to split a set of training data into a tree that consists of leaves and nodes. Each leaf consists of a homogenous group of examples

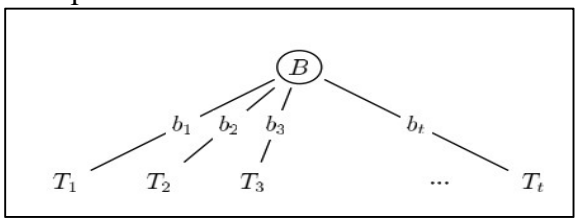

Figure 2 An example of partitioning of a decision tree node. [6]

(i.e. they all contain the same attributes) and each node consists of examples with two or more attributes that can be tested and used to split the examples into more homogenous groups. "If all the cases in S belong to the same class (C_j say) the decision tree is a leaf labelled with C_j. Otherwise, let B be some test with outcomes (b_1, b_2… b_t) that produces a non-trivial partition of S, and denote by S_i the set of cases in S that has outcome b_i of B. The decision tree is [formed], where T_i is the result of growing a decision tree for the cases in S_i." [7]. C4.5 uses two criteria to determine how to split up the examples, information gain and gain ratio. As described in [7]:

> Let $RF(C_j, S)$ denote the relative frequency of cases in S that belong to class C_j. The information content of a message that identifies the class of a case in S is
>
> $$I(S) = -\sum_{j=1}^{x} RF(C_j, S) \log RF(C_j, S)$$
>
> After S is partitioned into subsets $S_1, S_2, \ldots, S_t$ by a test B, the information gained is then
>
> $$G(S, B) = I(S) - \sum_{i=1}^{t} \frac{|S_i|}{|S|} I(S_i)$$
>
> The gain criterion chooses the test B that maximizes $G(S, B)$. The gain ratio criterion sidesteps this problem by also taking in to account the potential information from the partition itself:
>
> $$P(S, B) = -\sum_{i=1}^{t} \frac{|S_i|}{|S|} \log\left(\frac{|S_i|}{|S|}\right)$$
>
> Gain ratio then chooses, from among the tests with at least average gain, the test B that maximizes $G(S, B)/P(S, B)$.

This process is done recursively until each branch of the tree ends in a leaf. As a classic example, consider the following data set for whether someone went golfing or didn't:

Outlook	Temp	Humidity	Windy	Played Golf

Rainy	Cool	High	Yes	No
Sunny	Hot	High	No	Yes
Rainy	Cool	Low	No	No
Sunny	Hot	High	Yes	Yes

The first step is to calculate the information content

$$I(S) = -(RF(Outlook,S)\log RF(Outlook,S) + RF(Temp,S)\log RF(Temp,S) + RF(Humidity,S)\log RF(Humidity,S) + RF(Windy,S)$$
$$= -(0.5\,log\,0.5 + 0.5\,log\,0.5 + 0.75\,log\,0.75 + 0.5\,log\,0.5) = -(-0.15 + -0.15 + -0.09 + -0.15) = 0.54$$

Then, the test B could start by splitting the data based on Outlook, so gain would equal

$$G(S,Outlook) = 0.54 - \left(\frac{1}{4}\times 0.15 + \frac{3}{4}\times 0.39\right) = 0.21$$

and potential information would equal

$$P(S,Outlook) = -\left(\frac{1}{4}\log\frac{1}{4} + \frac{3}{4}\log\frac{3}{4}\right) = 0.24$$

Then, the data could be split up according to the other cases as such, and the test that maximizes $G(S,B)/P(S,B)$ would be chosen to create a partition.

On a training set, this process can be done so as to produce zero error [8]. However, "this so called overfitting is generally thought to lead to a loss of predictive accuracy in most applications." [7]. Pruning, or removing some of the test nodes of the decision tree after it has been produced is an agreed upon method to avoid overfitting. In order to prune a decision tree a predicted error needs to be calculated in order to determine if any changes need made to a node, as described in [8]:

Using the binomial theorem, confidence limits can be calculated for the probability of error for a given confidence level. The confidence level is the certainty factor parameter CF. The upper limit of the probability is found. Given this value the predicted number of errors for each leaf of a test node being considered for pruning can be calculated by multiplying the number of examples at the leaf by the upper limit of the probability confidence limit. The predicted number of errors if a node was a leaf can be calculated from the observed number of errors after its leaves are collapsed. The leaves are pruned if the number of predicted errors after pruning is less than the sum of predicted errors across the leaves. The smaller the CF becomes the more certain we are that the confidence interval contains the true probability of error.

The standard for a C4.5 decision tree is to use a CF value of 0.25, but this value can be altered to change the level to which the tree is pruned [7]. Once a tree has been pruned, it will perform better on examples it has never seen before than if it had been left unpruned.

C. Neural Networks Initialized with Decision Trees

A novel approach to document categorization, as outlined in [8][4], is to initialize the weights of a neural network with a decision tree instead of initializing them randomly. This neural network consists of two hidden layers, the first is called the literal layer and the second is called the conjunctive layer. The first step in creating a neural network initialized with a decision tree is to create the decision tree for a set of training data. Once the tree has been built, every path from the root node to a leaf is made a rule and that set of rules is then converted into the

disjunctive normal form. Then, start the input layer of the neural network with as many nodes as there are distinct attributes (words). The next layer of the neural network, the literal layer, then gets a node for each literal of the disjunctive normal form.

The biases can then be set for the literal layer based on if (attribute > value) $bias = -\alpha \times value$ and weights are set to α and if (attribute $\leq$ value) $bias = \alpha \times value$ and weights are set to $-\alpha$. Then, for every conjunction of literal in the literal layer, create a node in the conjunctive layer and set weights connecting appropriate nodes in literal and conjunctive layer to $bias = \alpha \times (2n - 1)$, where n is the number of literals in the conjunct and set weights to α. Finally, create an output layer with nodes corresponding to each possible output and set weights from corresponding nodes in conjunctive layer to nodes in output layer to α and $biases = -\alpha \times (1/2)$. Set all remaining weights in each layer to $+\beta$ or $-\beta$ with equal probability [8][4]. Here, $\alpha = 5$ and $\beta = 0.25$ and both exist as constants. Once all the layers have been created and the weights have been connected, the network can then train (on the same set of data used to train the decision tree) using the normal back propagation method. This method is capable of in increased accuracy rate over a BPNN or decision tree on its own [8].

IV. Primary Objective

To compare the classification method presented in [8] for categorizing articles into one of the following categories: business, entertainment, politics, sports and technology, with the method used in [1]. (1.5 person weeks over 1 semester)

V. Solution Description

This experiment will be implemented in C using a Linux environment. All training and test documents will come from the BBC dataset, which consists of a total of 2,225 documents from the BBC news website split into 5 categories (business, entertainment, politics, sports and technology) from the years 2004-2005. The first step will be to build a 50-Most Important Word tool, which will extract from each of the 5 categories the top 50 words within the chosen range of TF-IDF values. TF-IDF here will be calculated by acting under the assumption that all documents under the same category make up a single document. In this way each category can be considered a single document, and categories can be compared against each other. Next will be to implement the pre-built Porter's Stemmer by Martin Porter [9] to create a stemmed version of each category. Once each category has a stemmed and non-stemmed version, two sets of 50-Most Important words will be calculated respectively (this will only need to be calculated once). Next, training and test documents can be chosen from the corpus at random. The NNIDT can be built by first using Ross Quinlan's prebuilt C4.5 Decision tree [10] to build a decision tree for the given training set and thus create a set of rules that can be converted into the DNF. Once the DNF is computed, the Neural Network portion of the NNIDT can be built by first creating an input layer with 250 nodes (50 words times 5 categories). Once the input layer has nodes, the literal layer can be created with weights and biases, then the conjunctive layers and output layers. Once all weights and biases are initialized, the network can begin training using back propagation on each of the training documents in the sample. Upon testing the test documents from the sample, expected input should be a count of how many times each important word from each category appeared and the expected output should be the correct category the test document belongs to.

VI. Goals and Hypotheses

The first hypothesis for this experiment is that, without the use of stemming, the NNIDT will perform better than a Deep Belief Networks with three RBMs. The second hypothesis is that the introduction of stemming will increase the accuracy of the NNIDT. Figure 5 shows the Goal Tree for this experiment.

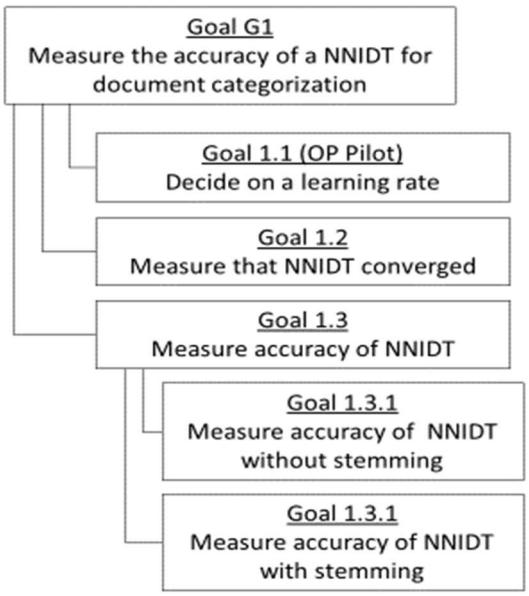

Figure 3 Goal Tree

VII. Experiment Design

This experiment will test three different sample sizes from the corpus. One size will be 1,500 documents total, 750 documents total and 375 documents total. Each sample size will use a 90:10 training to testing aspect ratio, and each sample size will be tested one hundred times with different, randomly chosen articles. Additionally, each sample size will be tested using both un-stemmed and stemmed 50-Most Important words. Figure 6 shows the block design for this experiment and Figure 7 shows the time estimates.

Block Design		With Stemming	
		Yes	No
Training and Test Size	1500 total – 1350 training, 150 test	X	X
	750 total – 675 training, 75 test	X	X
	375 total – 338 training, 37 test	X	X

Figure 4 Experiment Block Design

Training Time per Experiment:

1350 articles * 1000 iterations * 10 runs * 0.0001 seconds * 2 = 55 minutes

675 articles * 1000 iterations * 10 runs * 0.0001 seconds * 2 = 23 minutes

338 articles * 1000 iterations * 10 runs * 0.0001 seconds * 2 = 12 minutes

Total Training Time: **2 hours**

Testing Time per Experiment:

150 articles * 10 runs * 0.0001 seconds * 2 = 0.3 seconds

75 articles * 10 runs * 0.0001 seconds * 2 = 0.15 seconds

37 articles * 10 runs * 0.0001 seconds * 2 = 0.074 seconds

Total Testing Time: **> 1 minute**

Figure 7 Time estimates for experiment

VIII. Results

Average accuracy for a DBN with 3 compared to that of an NNIDT using an $\alpha = 1$, $\beta = 0.025$ and a learning rate of 0.01 (Figure 8) produced t-values of 42.89 on 1500 samples, 64.16 on 750 samples and 87.29 on 375 samples. Comparing the NNIDTs accuracy using un-stemmed input

	Average Accuracy	
Sample Size	DBN with 3 RBM	Un-stemmed NNIDT α = 1 β = 0.025
1500	70%	88.29%
750	61%	90.89%
375	56%	93.99%

Figure 8 Average accuracy of DBN with 3 RBMs and NNIDT

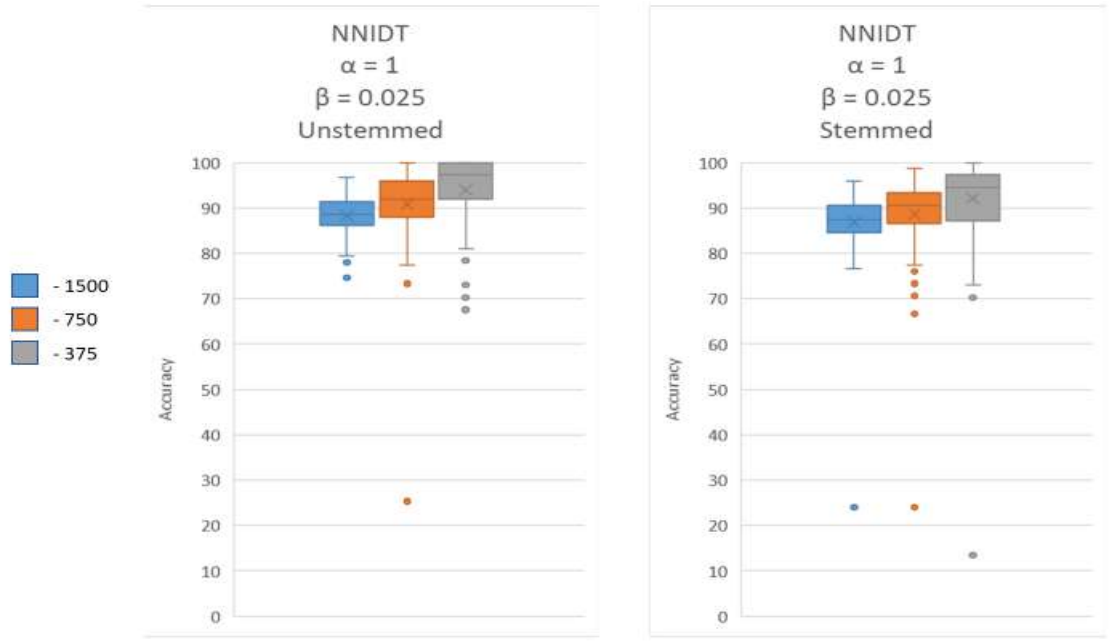

Figure 9 Average accuracy of NNIDT stemmed and un-stemmed

vs. stemmed input (Figure 9) produced t-values of 1.59 for 1000 samples, 1.76 for 750 samples and 1.41 for 375 samples. It should be noted that because samples were shuffled and selected randomly there is potential for two treatments to be tested on the exact same sample, which would nullify the use of a two-sample t-test for the statistical analysis of these results.

IX. Discussion

Compared to a DBN with 3 RBMs, the NNIDT had a significantly higher average accuracy with all three sample sizes. An interesting observation is that as the sample size decreased with the DBN, accuracy decreased, however the opposite was true with the NNIDT. This is probably due to a large amount of known overlap between categories in this dataset, meaning fewer samples will produce less words that appear in more than one category and thus more unique words per category. With the t-values computed, the null hypothesis can be rejected and the alternative hypothesis that the NNIDT was more accurate than a DBN with 3 RBMs is supported. However, there seems to be no significant difference between the NNIDTs accuracy with un-stemmed and with stemmed input, therefore the alternative hypothesis that stemming would increase the NNIDTs accuracy is not supported. Likely, stemming did not increase accuracy because the amount of overlap between categories meant that any stemming of words was proportional in all categories and therefore in no way modified the information associated with the words relative to the documents (i.e. all categories used different forms of words equally, so stemming reduced the quantity of unique words equally in all categories).

X. Conclusions and Future Work

Document categorization is a topic that has grown in relevance as more and more documents are created online. Discussed in this paper are the two basic steps needed to use machine learning as a method of categorizing text documents; information retrieval and information processing as well as an experiment designed to test the accuracy of an NNIDT compared to the results in [1]. The first step is to extract relevant information from a document, and

both TF-IDF and Information Gain are techniques used to decide which words in a document are important based on a bag-of-words model. Once these words have been extracted, stemming is a technique that can be used to condense the data into a smaller set by shortening a word to its root.

Both BPNNs and decision trees are common methods of processing these words to determine what category the document belongs to. BPNNs use a series of hidden layers of interconnected nodes, each with an activation and weight that are initialized randomly and can be adjusted using a cost function back propagated through the network to get closer to the correct result. Decision trees use a divide and conquer technique to split a training set into groups based on like attributes, eventually ending in leaves that contain a homogeneous group of examples. Also, as a novel method, BPNNs and decision trees can be combined to create a method of processing the words and choosing a category for the document that has increased accuracy over a BPNN or a decision tree on their own. Further, it was found that a NNIDT had a higher average accuracy than that of a DBN with 3 RMBs, however the introduction of stemming the NNIDTs input did not have a significant effect. Future work to be done on this design is modifying α and β values as well as the learning rate of the NNIDT to lower the number of iterations needed to reach a maximum accuracy. Additionally, testing the NNIDT on a different data set with less overlap and a larger population would be useful in verifying why fewer samples made the NNIDT perform better than it did with more samples.

XI. Bibliography

[1] J. Rebok, "Deep Belief Network to Classify Documents." .

[2] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. 2009.

[3] T. M. Mitchell, "Machine learning in ecosystem informatics and sustainability," in *IJCAI International Joint Conference on Artificial Intelligence*, 2009, pp. 8–13.

[4] A. Banerjee, *Initializing Neural Networks using Decision Trees*. 1989.

[5] M. A. Mielsen, "Neural Networks and Deep Learning," in *Determination Press*, Determination Press, 2014, pp. 875–936.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations By Error Propogation," *ICS Rep. 8506*, no. V, 1985.

[7] R. Kohavi and R. Quinlan, "Decision Tree Discovery," *Handb. Data Min. Knowl. Discov.*, vol. 3, no. Hunt 1962, pp. 267--276, 1999.

[8] N. Remeikis, I. Skucas, and V. Melninkaite, "Text Categorization Using Neural Networks Initialized with Decision Trees," *Informatica*, vol. 15, no. 4, pp. 551–564, 2004.

[9] M. Porter, "Porter Stemming Algorithm." [Online]. Available: https://tartarus.org/martin/PorterStemmer/. [Accessed: 24-Apr-2018].

[10] R. Quinlan, "Ross Quinlan's personal homepage." [Online]. Available: http://www.rulequest.com/Personal/. [Accessed: 24-Apr-2018].

[11] Y. Yang and J. O. Pedersen, "A Comparative Study on Feature Selection in Text Categorization," *Bioinformatics*, vol. 20, no. 15, pp. 2429–2437, 2004.