Spam ? Not Any More ! Detecting Spam emails using neural networks

ECE / CS / ME 539
Project

Submitted by
Sivanadyan, Thiagarajan
Last Name **First Name**

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Importance of the topic

'Spam' is the word for unsolicited or unwanted emails including advertisements for products/web sites, make money fast schemes, chain letters, pornography... In short, it is several varieties of email and newsgroup abuse, loosely categorized by using the Internet to deliver inappropriate messages to unwilling recipients. The amount of it delivered today is appalling. An average user on the internet gets about 10-50 spam emails a day and about 13 billion pieces of unsolicited commercial e-mail are sent each day, which represents about half of all e-mail sent[1].

The harm in spam, excluding the time spent in getting rid of them, is the fact that it uses up resources such as disk space and bandwidth, extremely precious quantities in the modern context. Also, a lot of disreputable and illegal companies and individuals utilize the opportunity to perpetrate various scams, illegal products and other inappropriate materials. Estimates of lost productivity are as high as $10 billion year. Spam does not cost the sender anything - most of the expenses are paid for by the carriers or the recipient rather than by the sender. Due to these reasons and for the outright infringement on personal space, some method of preventing the normal delivery of spam is desired. The seriousness of the problem of spam can be inferred from the fact that, 'The U.S. House of Representatives has approved an amended version of a bill that will allow penalties of up to $6 million and five years in jail for sending some e-mail spam as part of the CAN-SPAM Act' [1].

However, the goals of a spam blocking mechanism vary depending on the user. In most cases, it is important to avoid false hits (where non-spam gets classified as spam), at the risk of allowing some spam through. In few cases, such as net Moms (software which prevent inappropriate messages reaching younger viewers), some degree of laxity is allowed, especially since the parent can view it later and sort it out. Also, the nature of spam emails received differs among users, and spam content can also vary with time. Therefore high adaptability is one of the prime concerns of any anti-spam software.

1.2 Current Technology

Software to combat spam e-mail, until recently, was based on simple keyword filters; if a given term was present in a message's headers and/or body, the mail was labeled as spam. This rapidly became unscalable as each potential spam term had to be manually added to the system, resulting in little time saved for users. There are several spam filters available today. Most of them fall into one of the following categories: [2]

User defined filters: Available on most email servers. With these filters you can forward email to different mailboxes depending on headers or contents.

Header filters: These are more sophisticated. They look at the email headers to see if they are forged.

Language filters: They simply filter out any email that is not in your native tongue. It only filters out foreign language spam, which is not a major problem today, unless the foreign language under question is English.

Permission filters: They block all email that does not come from an authorized source. Typically the first time you send an email to a person using a permission filter you will receive an auto-response inviting you to visit a web page and enter some information.

Content filters: They scan the text of an email and either neural networks or fuzzy logic to give a weighted opinion as to whether the email is Spam. They can be highly effective, but can also occasionally filter out newsletters and other bulk email that may appear to be Spam.

1.3 Project statement and outline

This project attempts to apply a neural network to the problem of spam recognition. It is based on content based filtering and is a method that is getting popular these days. It is can be a very accurate and efficient method for text classification. It is an extension of text classification technology, which searches the textual content of an email and employs algorithms to classify an email as spam or not-spam (1 or 0). The algorithms are able to classify the occurrence of certain words and phrases in terms of how and where they appear in the email message, not by their existence alone. Thus the spam filtering problem can be broken down into a simple classification problem and most of the time-tested networks and algorithms such as MLP with Back propagation can be used.

The project does a full implementation of spam classification and comparison of the performance of different MLP network configurations. Also the programs have been optimized with respect to the momentum and learning parameters. The learning algorithm used was the back propagation algorithm. A number of configurations were tried out for both the complete input data set and the modified input data set with fewer attributes. A two way cross-validation was also performed on the networks.

An attempt was made to apply the K-nearest neighbor classifier to perform clustering on the data. However, due to extremely large requirement for K, it is impractical for an implementation of the K-nearest-neighbor algorithm. I did obtain partial results for the algorithm, but due to its very poor performance, I ignored its inclusion in the report. However, it should be mentioned that the algorithm performs very badly compared to the base performance of an MLP.

Other methods such as Support Vector Machine and Self Organizing Map were impractical due to the extremely large number of inputs (57) and large input data (around 4000 feature vectors).

2. PREPROCESSING THE DATA

2.1 Analysis of the original data

The data used for the project was obtained from the University of California – Irvine Machine Learning Repository [3]. The data was obtained from an analysis done by Hewlett-Packard [See Acknowledgements].

Output Information:

The last column of 'spambase.data' denotes whether the e-mail was considered spam (1) or not (0) i.e. unsolicited commercial e-mail.

Input attributes:

Most of the attributes indicate whether a particular word or character was frequently occurring in the e-mail. The run-length attributes (55-57) measure the length of sequences of consecutive capital letters. Here are the definitions of the attributes:

- 48 continuous real [0,100] attributes of type word_freq_WORD = percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail. A "word" in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.
- 6 continuous real [0,100] attributes of type char_freq_CHAR = percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurences) / total characters in e-mail
- 1 continuous real [1,...] attribute of type capital_run_length_average = average length of uninterrupted sequences of capital letters
- 1 continuous integer [1,...] attribute of type capital_run_length_longest = length of longest uninterrupted sequence of capital letters
- 1 continuous integer [1,...] attribute of type capital_run_length_total = sum of length of uninterrupted sequences of capital letters = total number of capital letters in the e-mail

Class Distribution:

Spam	- 1813 (39.4%)
Non-Spam	- 2788 (60.6%)

Examples of the words used in the input are: ‘free’, ‘credit’, ‘business’ (commonly found in spam emails) and ‘project’, ‘meeting’ , ‘George’, ‘hp’ (found in personal mails)

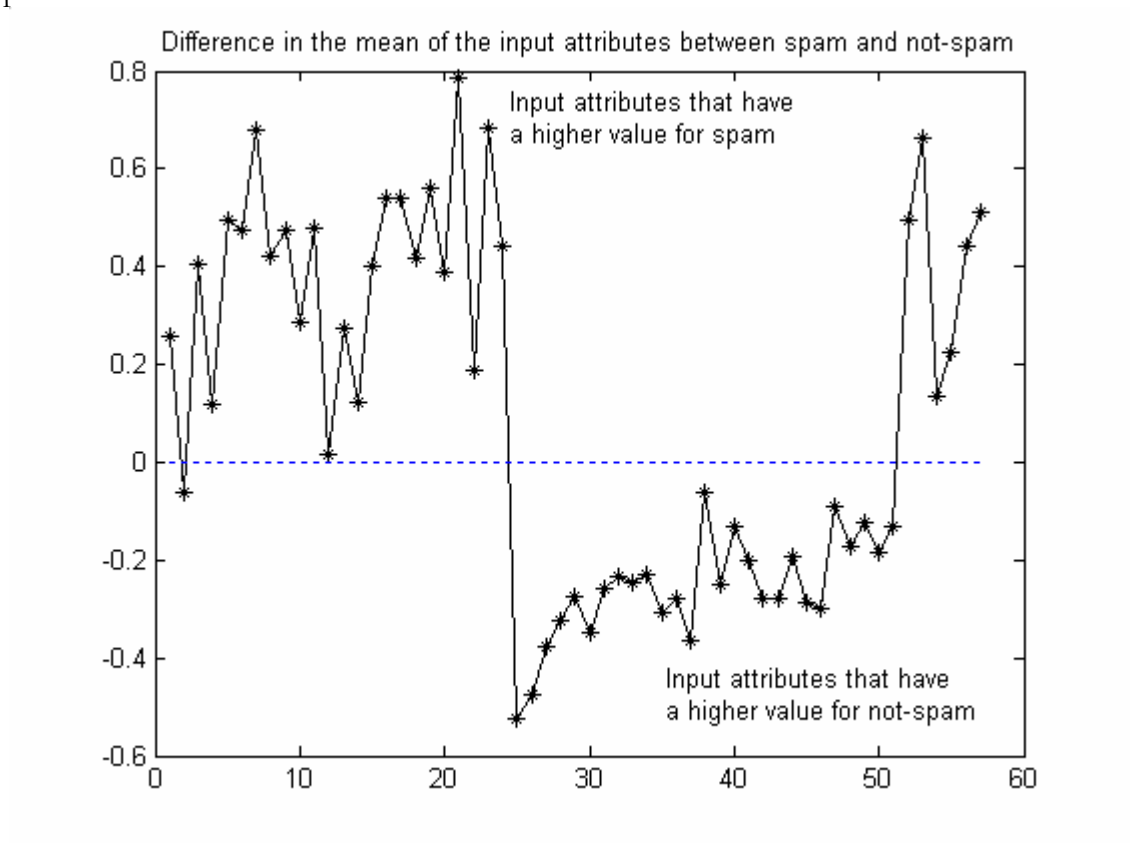
The mean, variance, min and max values were found for the all the input attributes [File No 1]. From analysis of the results one can deduce that certain input attributes tend be redundant and do not really have high variance (values closer together implying they cannot easily be distinguished from each other). To find out the inputs that are really important to our classification, the following procedure was carried out.

2.2 Difference in mean procedure

To find out the inputs that were different for spam and not-spam emails, we can do a difference of mean procedure as follows: [File No. 2]

1) Separate the feature vectors for spam and not-spam emails into spam sets and not-spam sets.
2) Calculate the mean for each feature vector in the spam set.
3) Calculate the mean for each feature vector in the not-spam set.
4) Find the difference of the means for each feature vector.

The plot of the difference of means is shown below



From the plot, one can infer the following

- Certain input features have high values for spam emails (Eg. More occurrences of the word credit) and certain others for not-spam emails (Eg. conference)
- Among these input features some show larger differences than others.
- If one were to use a threshold (only inputs that differ a lot for spam and not spam emails), one can use a reduced set of input without much loss in performance (in some cases actually improve performance)

2.3 Reduction of the data set

Based on the above findings, a thresholding procedure was applied and two reduced data sets were obtained (threshold values – 0.4 and 0.5) **[File No 3].** Nearly all the results shown in this project were obtained with the reduced data sets. The original data set was too extensive and redundant and one of the results actually shows that using an inefficient data set reduces the performance considerably.

2.4 Preprocessing & splitting the data

Preprocessing: All the inputs were then made zero mean and unit variance. Since it is important that we detect all non-spam emails, even at the risk of allowing a few spam mails sneak in, the class attribute for spam is made 1 and class attribute for non-spam is made -2. It is the equivalent of providing greater rewards for detecting non-spam emails than spam emails. This ensures that the non-spam emails will have a higher detection rate.

Splitting the data: The data was then split into testing and training sets. To eliminate any data specific behavior, two random sets of testing data were extracted from the preliminary data set, and the other set in the two cases were used for training. From this point on, we will have a training set 1 and a corresponding testing set 1, and a training set 2 and a corresponding testing set 2. [File No. 4]

Data Sets used for training and testing the networks:

- Reduced data set 1:

Number of input attributes: 21
Number of output classes: 2 (spam and non-spam)

Training data sets: 2 (4025 Feature Vectors each)
Testing data sets: 2 (576 Feature Vectors each)

- Reduced data set 2:

Number of input attributes: 9
Number of output classes: 2 (spam and non-spam)

Training data sets: 2 (4025 Feature Vectors each)
Testing data sets: 2 (576 Feature Vectors each)

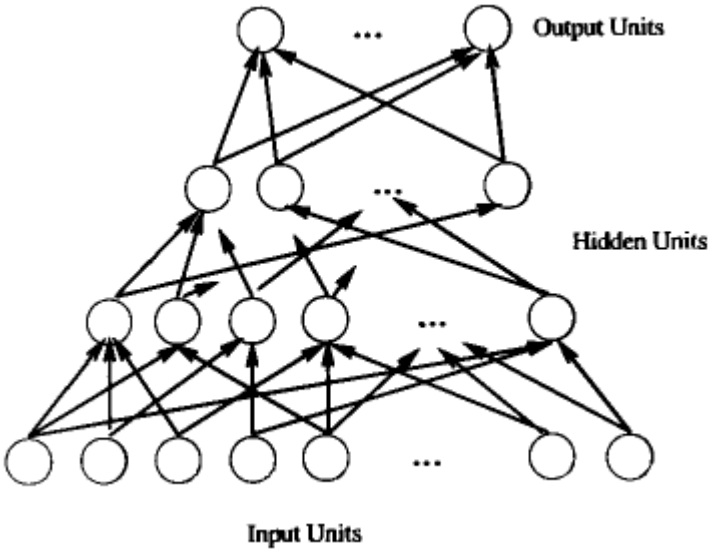
3. MULTI-LAYER PERCEPTRON IMPLEMENTATION

3.1 What is it?

" The Multi-layer perceptron is one of the most widely used types of neural networks. It is simple and based on solid mathematical grounds. Input quantities are processed through successive layers of "neurons". There is always an input layer, with a number of neurons equal to the number of variables of the problem, and an output layer, where the perceptron response is made available, with a number of neurons equal to the desired number of quantities computed from the inputs (very often only one). The layers in between are called "hidden" layers. With no hidden layer, the perceptron can only perform linear tasks (for example a linear discriminant analysis, which is already useful). All problems which can be solved by a perceptron can be solved with only one hidden layer, but it is sometimes more efficient to use 2 hidden layers. Each neuron of a layer other than the input layer computes first a linear combination of the outputs of the neurons of the previous layer, plus a bias. The coefficients of the linear combinations plus the biases are called the weights. They are usually determined from examples to minimize, on the set of examples, the (Euclidian) norm of the desired output - net output vector. Neurons in the hidden layer then compute a non-linear function of their input. Usually, the non-linear function is the sigmoid function $y(x) = 1/(1+\exp(-x))$. The output neuron(s) has its output equal to the linear combination. Thus, a Multi-Layer Perceptron with 1 hidden layer basically performs a linear combination of sigmoid function of the inputs. A linear combination of sigmoids is useful because of 2 theorems:

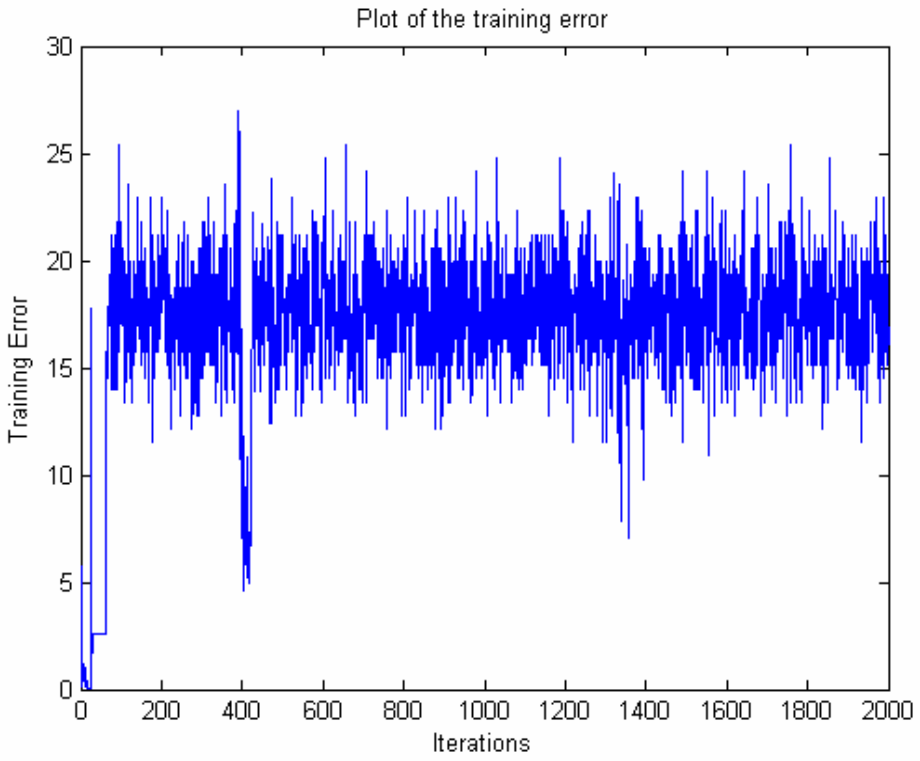
- a linear function of sigmoids can approximate any continuous function of 1 or more variable(s). This is useful to obtain a continuous function fitting a finite set of points when no underlying model is available.
- trained with a desired answer = 1 for signal and 0 for background, the approximated function is the probability of signal knowing the input values. This second theorem is the basic ground for all classification applications. " [11]

A generic picture of an MLP is shown below.



3.2 MLP Simulation using the complete data set

An MLP network with 5 hidden layers (20 – 10 – 10 – 10 – 5) was simulated for the complete training data set (4025 Feature vectors with 57 input attributes). However the network was extremely unstable as far as the error convergence was concerned. In some cases, it classified all the not- spam as spam. However, on increasing the momentum, there was a small improvement in performance of the network. One can see from the plot of the training error that the network failed to classify the input vectors.



The following table lists the performance of various MLP networks on the complete data set

MLP Architecture	Learning Rate	Momentum	Average Classification Rate
20 -10 - 10 -10 - 5	0.1	0.8	62.4 %
20 -10 - 10 -10 - 5	0.1	0.95	63.2 %
20 - 10 - 10 - 5	0.1	0.85	60.61 %
15 - 15 - 15 - 5	0.1	0.85	60.59%

The network will 5 layers and with higher momentum gave the best performance. However, since the percentage of non-spam emails is around 39.31%, this means the last two networks classified all emails as not-spam irrespective of the input conditions. However, the reduced data set had much better performance levels.

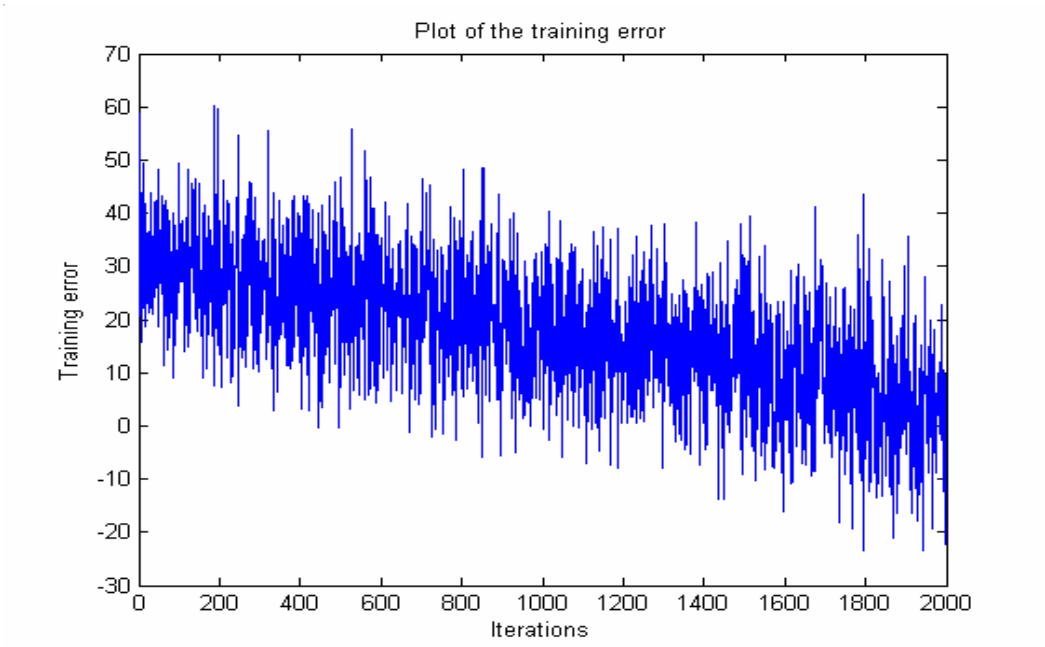
3.3 MLP Simulation using the reduced data set (Inputs - 21)

Numerous MLP configurations were tested on the reduced data set 1, and also the momentum and learning parameter was varied. The results were averaged over 20 iterations of the algorithm. The best classification rates have been tabulated below.

MLP Architecture	Learning Rate	Momentum	Average Classification Rate
20 -10 - 10 -10 - 5	0.1	0.8	93.5 %
20 -10 - 10 -10 - 5	0.1	0.95	90.3 %
20 -10 - 10 -10 - 7	0.1	0.8	**93.8% (Best Performance)**
15 -15 - 10 -10 - 5	0.1	0.8	91.6 %
15 -15 - 10 -10 - 7	0.1	0.85	91.2 %
20 - 10 - 10 - 5	0.1	0.85	87.6 %
15 - 15 - 15 - 5	0.1	0.85	87.8 %

Parameters for the simulations:

Input Scaling	[-5 5]
Output Scaling	[0.2 0.8]
Number of Epochs	2000
Epochs before convergence test.	50



The error plot for the best network (20 -10 - 10 -10 - 7) has been shown in the above figure. Although the error variance is high, the average error seems to be steadily going down. This shows that the MLP is getting better at classifying the data as the training increases.

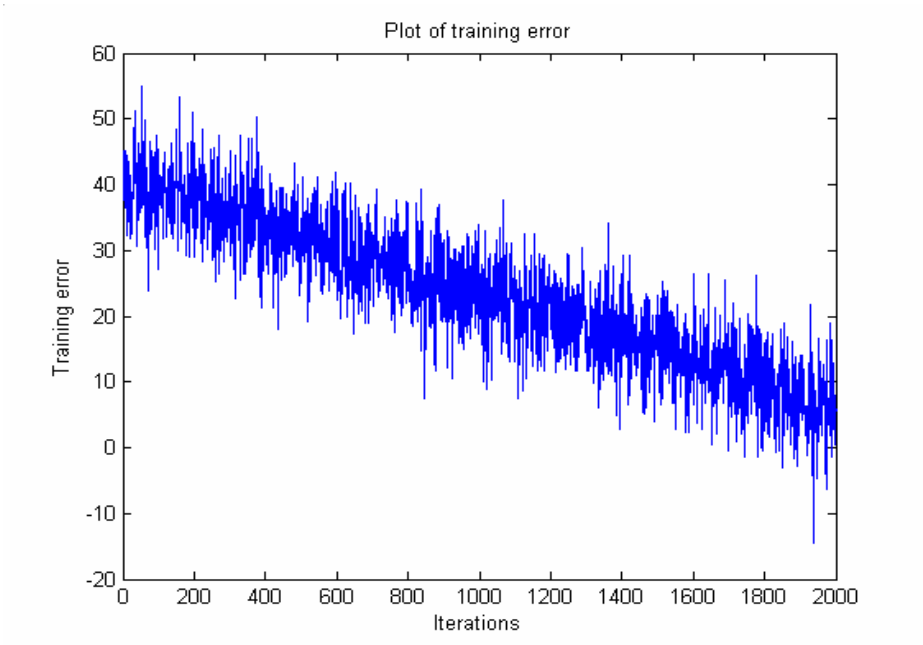
3.4 MLP Simulation using the reduced data set (Inputs - 9)

Numerous MLP configurations were tested on the reduced data set 2, and also the momentum and learning parameter was varied. The results were averaged over 20 iterations of the algorithm. The best classification rates have been tabulated below.

MLP Architecture	Learning Rate	Momentum	Average Classification Rate
20 - 10 -10 - 8	0.1	0.8	**92.1% (Best Performance)**
20 -10 - 10 -5	0.1	0.8	91.8 %
20 - 10 -10 - 5	0.1	0.95	91.7 %
15 - 10 -10 - 5	0.1	0.8	90.1 %
15 - 10 -10 - 8	0.1	0.85	90.1 %
10 - 10 - 5	0.1	0.85	86.1 %
15 - 15 - 5	0.1	0.85	87.3 %

Parameters for the simulations:

Input Scaling	[-5 5]
Output Scaling	[0.2 0.8]
Number of Epochs	2000
Epochs before convergence test.	50



One can notice that the error variance is reduced from the training error plot shown above. This error plot is for the best MLP configuration (20 - 10 -10 - 8). But increasing the number of levels brought degradation in the performance due to over-modeling the data.

3.5 MLP Simulation using cross-validation

The method to cross validate is to mix and match training and testing data sets (Training data set 1 used with testing data set 2 and vice versa). Again the results have been tabulated for both the reduced data sets with 21 and 9 inputs.

For data set with 21 inputs

MLP Architecture	Learning Rate	Momentum	Average Classification Rate
20 -10 - 10 -10 - 5	0.1	0.8	93.5 %
20 -10 - 10 -10 - 5	0.1	0.95	90.3 %
20 -10 - 10 -10 - 7	0.1	0.8	**93.8 % (Best performance)**

For data set with 9 inputs

MLP Architecture	Learning Rate	Momentum	Average Classification Rate
20 - 10 -10 - 8	0.1	0.8	**92.4% (Best Performance)**
20 -10 - 10 -5	0.1	0.8	91.7 %
20 - 10 -10 - 5	0.1	0.95	91.8%

Parameters for the simulations:

Input Scaling	[-5 5]
Output Scaling	[0.2 0.8]
Number of Epochs	2000
Epochs before convergence test.	50

In this case, the results indicate that cross-validation does not change the results significantly.

4. DISCUSSION AND INFERENCES

4.1 Discussion of the results

In the first case, when the whole data set was used for classification, one would expect a higher classification rate, just on the basis that more information has been provided to the neural network. However, for an MLP, it seems more important that it has input vectors that it can distinguish rather than larger number of input vectors. The percentage of not-spam in the total data was 60.31% percent. This means that even if the network classified all the emails as not-spam, it would have still achieved a classification rate of 60.31%. This shows that a classification rate of 63.4% in the first case, is near worst performance and hence it has to be rejected outright.

However, a big jump is noticed in the classification rate on working with the reduced input data set. The error curves plotted for both the reduced data sets (Inputs 9 & 21) show a good convergence property. However, one does notice the extent of noisiness in the error curves. This is due to the fact that email content changes a lot within spam emails and also with non-spam emails. Thus the weights shift around a lot during the training with each epoch. Also, there is a noticeable reduction in noise in the convergence curve of the data set with lower number of inputs (9). Again this is a direct offshoot of the fact that the content changes.

The overall performance numbers in the case of the MLP with the reduced data set show that as the number of inputs decrease, we encounter a reduction in the optimum number of layers (also number of neurons). This is accompanied with a slight decrease in the classification rates. However, this does provide us with the valuable degree of freedom as far as complexity is concerned. The number of layers, the input space dimension and the total number of neurons are a direct indication of the complexity of the network. So the results agree with our intuitive idea that beyond a point, reduction in number of inputs, starts decreasing the classification rates, but also reduces network complexity.

4.2 Inferences

- With larger number of inputs, the network complexity increases, but so does the performance.
- Utilizing a large input data set, without identifying the most important inputs, does not necessarily improve performance. We have shown in our case, that the network fails (classifies all email as spam) when we use the complete original data set.
- So rather than pick a lot of words from an email, we'd do better to pick fewer words but which occur in very different amounts in spam and non-spam emails.
- In this case, some of the inputs which varied a lot between spam and non-spam emails were words like 'meeting', 'hp' etc. In a university context, one could use the occurrences of words like 'university', 'research' etc, which occur very rarely in spam emails.
- By applying varying thresholds while restricting the number of inputs we can have a performance trade-off between the complexity of the network (number of inputs) and the accuracy of classification.
- There is no optimum MLP configuration for all inputs. We need to have an adaptive (software approach) to the MLP design problem.

- For a higher classification rate, it is important to utilize a combination of spam filtering methods, rather than just the neural network based spam blocking. Commercial email software such as Eudora utilize more adaptive means to change even the inputs given to the neural network, thereby making the spam blocking highly personalized and optimized on a person-to-person basis.

5. CONCLUSION

The application of neural networks to detecting spam is definitely something that can and is being pursued as a viable option. However, to obtain optimum performance, we do have to do sufficient amount of data analysis. Also, this data analysis has to be general so as to block a wider variety of spam.

'The basic principal used in any spam filtering technique, whether heuristic or keyword-based, is identical: spam messages generally look different than good messages and detecting these differences is a good way to identify and stop spam. The difference between these technologies really comes down to the problem of distinguishing between these two classes of email. The neural networks approach is more refined, more mathematical and potentially far more accurate and reliable in accomplishing this task.

Although no single technology can achieve one hundred percent spam detection with zero false positives (despite vendor claims), machine-learned heuristics in general and neural networks in particular have proven extremely effective and reliable at accurately identifying spam and minimizing errors to an acceptable minimum'. [5]

Further work:

- It would definitely be interesting to conduct cross-validation between data sets used from different sources and one could develop a heuristic model to pick inputs to be used for the network.
- Fuzzy logic is another important content-based method to distinguish spam. A fuzzy logic approach to the same problem can bring some new insights into the problem.
- A combinational approach can be used to achieve higher classification rates (using header filters, content based filters and user specific information).

6. REFERENCES AND ACKNOWLEDGEMENTS

[1] Grant Gross, 'Spam bill heads to the president', IDG News Service, http://www.nwfusion.com/news/2003/1209spambill.html

[2] 'White Papers - Spam Tutorial' – Vicomsoft http://www.spambolt.com/anti_spam_faq/email_spam_filter.html

[3] U-Cal, Irvine Machine Learning repository http://www.ics.uci.edu/~mlearn/MLSummary.html

[4] I. Androutsopoulos et al., ' An evaluation of Naïve Bayesian Anti-Spam Filtering', Proceedings of Machine Learning in the New Information Age, Spain, 2000

[5] Chris Miller, 'Neural Network-based Antispam Heuristics', Symantec Enterprise Security

[6] Levent Ozgur et al., ' Adaptive Turkish anti-spam filtering based on artificial neural networks', Turkish Symposium on AI and Neural Networks (TAINN), 2003

[7] Bart Massey et al., 'Learning Spam: Simple Techniques for freely-available software', to appear in proceedings of 2003 Usenix Annual Technical Conference.

[8] Simon Haykin, 'Neural Networks: A comprehensive foundation', Prentice Hall, Second Edition, 1999

[9] Richard S. Sutton, A.G. Barto, 'Reinforcement Learning: An introduction', MIT Press, 1998.

[10] Yu Hen Hu, 'ECE 539 lecture presentations' http://www.cae.wisc.edu/~ece539/fall03/notes/index.html

[11] MLP in Physics Analysis Workstation (PAW at Cern – the European laboratory for particle physics) http://paw.web.cern.ch/paw/mlpfit/pawmlp.html

- The data used for the project was obtained from the University of California – Irvine Machine Learning Repository.
- Creators: Mark Hopkins, Erik Reeber, George Forman, Jaap Suermondt , Hewlett-Packard Labs, 1501 Page Mill Rd., Palo Alto, CA 94304
- Donor: George Forman (gforman at nospam hpl.hp.com)
- Generated: June-July 1999

7. MATLAB FILES USED

Files for preprocessing the data:

1. analyzer.m - raw data analysis
2. trimmer.m - reducing the number of input attributes
3. preprocess.m - preprocessing the data
4. splitter.m - splitting the data into training and testing vectors

Files for the MLP simulations

5. bp_small_set1.m - Training an MLP using reduced data set1 (training and testing set 1)
6. bp_small_set2.m - Training an MLP using reduced data set1 (training and testing set 2)
7. bp_smaller_set1.m - Training an MLP using reduced data set2 (training and testing set 1)
8. bp_smaller_set2.m - Training an MLP using reduced data set2 (training and testing set 2)
9. bpconfig1.m - MLP configuration file
10. bpconfig2.m - MLP configuration file
11. bpconfig3.m - MLP configuration file
12. bpconfig4.m - MLP configuration file

- All other files used were taken from the course homepage (a line or two might change). Cross validation was performed by changing the testing vectors and taking the average performance.
- Simulations to show the effect of the learning parameters and momentum were carried out by individual modifications instead of in one set of iterations.
- There are a number of data sets provided. The original data set has been provided in spambase.data. All other data sets were got from this main data set.