



Genetic algorithm based neural network approaches for predicting churn in cellular wireless network services [☆]

Parag C. Pendharkar ^{*}

Information Systems, School of Business Administration, Pennsylvania State University at Harrisburg, 777 West Harrisburg Pike, Middletown, PA 17057, United States

ARTICLE INFO

Keywords:

Genetic algorithm
Neural networks
Churn prediction

ABSTRACT

Marketing research suggests that it is more expensive to recruit a new customer than to retain an existing customer. In order to retain existing customers, academics and practitioners have developed churn prediction models to effectively manage customer churn. In this paper, we propose two genetic-algorithm (GA) based neural network (NN) models to predict customer churn in subscription of wireless services. Our first GA based NN model uses a cross entropy based criterion to predict customer churn, and our second GA based NN model attempts to directly maximize the prediction accuracy of customer churn. Using real-world cellular wireless services dataset and three different sizes of NNs, we compare the two GA based NN models with a statistical z-score model using several model evaluation criteria, which include prediction accuracy, top 10% decile lift and area under receiver operating characteristics (ROC) curve. The results of our experiments indicate that both GA based NN models outperform the statistical z-score model on all performance criteria. Further, we observe that medium sized NNs perform best and the cross entropy based criterion may be more resistant to overfitting outliers in training dataset.

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

In the last decade, Customer Relationship Management (CRM) systems have replaced traditional mass marketing strategies by selective marketing practices (Burez & Van den Poel, 2006; Coussement & Van den Poel, 2008). These selective marketing practices involve identifying a sub-set of existing customers that are likely to stop using products or services of the company (churn). There are several data mining models that are proposed to predict potential customers that are most likely to churn. Among the popular models to predict customer churn are: neural networks, support vector machines and logistic regression models (Baesens, Viaene, Van den Poel, Vanthienen, & Dedene, 2002; Coussement & Van den Poel, 2008; Hung, Yen, & Wang, 2006).

Data mining research suggests that for non-parametric datasets, machine learning techniques, such as neural networks, often outperform statistical and structurally restrictive techniques such as linear and quadratic discriminant analysis approaches (Baesens

et al., 2002; Bhattacharyya & Pendharkar, 1998). Additionally, several studies have shown that genetic-algorithm based neural networks outperform traditional local search gradient descent/gradient ascent neural networks that use Rumelhart, Hinton, and Williams (1986) procedure for updating connection weights (Pendharkar, 2007; Pendharkar & Nanda, 2006; Pendharkar & Rodger, 2004). However, to our knowledge, there are no published studies that use genetic-algorithm based neural networks for predicting churn. Noting the paucity of studies in application of genetic-algorithm based neural networks in predicting customer churn, in this paper, we use two genetic-algorithm based neural network models to predict customer churn in cellular wireless network services. Since genetic algorithm based neural networks are sensitive to their network size, we compare different size neural network models to identify the best performing model. We compare our genetic algorithm based neural network models with a simple statistical z-score based prediction model.

The rest of our paper is organized as follows. In Section 2, we describe the two genetic algorithm based neural networks used in our study, and a simple univariate z-score classification model. In Section 3, we describe different neural network design factors and performance metrics used to monitor performance of competing techniques for predicting customer churn. In Section 4, we describe the real-world wireless services dataset, experiments and results. In Section 5, we conclude our paper with a summary and provide a few directions for future work.

[☆] The wireless telecommunication subscription services dataset used in this study was provided by the Teradata Center for Customer Relationship Management at Duke University.

^{*} Tel.: +1 717 948 6028; fax: +1 717 948 6456.

E-mail address: pxp19@psu.edu

URL: <http://www.personal.psu.edu/pxp19/>.

2. An overview of modeling techniques used in our research

The churn prediction model that we are learning is of type $f: X \rightarrow \{0,1\}$, where X represents an instance space of training cases; $f(x) = 1$ represents the decision belonging to class one (the customer will churn), and $f(x) = 0$ represents the decision belonging to class two (the customer will not churn). Assume a data set $S = \{\langle x_1, s_1 \rangle, \dots, \langle x_a, s_a \rangle\}$ of a examples, where x_j is the vector of decision-making attributes and $s_j = \{0, 1\} \forall j \in \{1, \dots, a\}$ is the known observed value of $f(x_j)$. As mentioned in the Introduction section, there are many different ways to learn the classification function $f()$. In this research, we use focus on genetic-algorithm based neural network approaches.

One of the challenges of learning classification function is to learn how to deal with inconsistent examples. For example, among the examples exhibiting same decision-making attributes, a few may belong to class one and the others to class two. This inconsistency in the data set may be due to lack of data on all attributes that predict the *positive* class one outcome. The maximum likelihood neural network (MLNN) attempts to resolve this inconsistency by learning the probability that an example belongs to class one (Mitchell, 1997). The MLNN learns a probabilistic function of type $g: X \rightarrow [0,1]$ such that $g(x)$ is the probability that $f(x) = 1$. Learning in MLNN requires learning of maximum likelihood hypothesis. The maximum likelihood procedure can be described as follows:

Assume that x_j and s_j are random variables, we can write the conditional probability of data S given a hypothesis h , $P(S|h)$ as follows:

$$P(S|h) = \prod_{j=1}^a P(x_j, s_j|h) = \prod_{j=1}^a P(s_j|h, x_j)P(x_j).$$

Let $h(x_j) = P(s_j = 1|h, x_j)$, we can write $P(s_j|h, x_j)$ as follows:

$$P(s_j|h, x_j) = h(x_j)^{s_j} (1 - h(x_j))^{1-s_j}.$$

Thus, $P(S|h)$ can be written as follows:

$$P(S|h) = \prod_{j=1}^a h(x_j)^{s_j} (1 - h(x_j))^{1-s_j} P(x_j).$$

Let H be set of all possible hypotheses, the maximum likelihood hypothesis $h_{ML} \in H$ can be obtained by dropping the constant probabilities $P(x_j)$'s as follows:

$$h_{ML} = \arg \max_{h \in H} \prod_{j=1}^a h(x_j)^{s_j} (1 - h(x_j))^{1-s_j}.$$

The above can be simplified by taking the natural logarithm of maximum likelihood hypothesis – the log maximum likelihood hypothesis.

$$\ln(h_{ML}) = \arg \max_{h \in H} \sum_{j=1}^a s_j \ln(h(x_j)) + (1 - s_j) \ln(1 - h(x_j)).$$

The negation of the expression $s_j \ln(h(x_j)) + (1 - s_j) \ln(1 - h(x_j))$ is called cross entropy (Mitchell, 1997).

When a neural network is used to maximize the log maximum likelihood hypothesis (minimizing cross entropy), a logistic function $g(x)$ is often used (Baesens et al., 2002). This logistic function takes the following form:

$$g(x) = \frac{1}{1 + e^{-x}}.$$

Using the logistic function and using a local search procedure similar to one described in Rumelhart et al. (1986), a local search gradient ascent optimization procedure can be developed to learn

connection weights of a neural network so that log maximum likelihood hypothesis can be maximized. The details for such an optimization procedure can be found in Pendharkar (2007) study.

The local search procedures for learning connection weights for neural networks are widely criticized by researchers (Pendharkar & Rodger, 2004; Sexton, Allidae, Dorsey, & Johnson, 1998; Sexton, Dorsey, & Johnson, 1999). Among the criticisms of local search procedures are tendency to overfit the training data examples, and convergence to sub-optimal solutions. There are several studies in the literature that have shown that global search procedures to learn connection weights of neural networks consistently outperform the local search optimization procedures on holdout data samples (Pendharkar, 2007; Pendharkar & Rodger, 2004; Sexton et al., 1998, 1999).

There are a few differences between local search optimization and global search procedures. Local search procedures have strong theoretical foundations in applied mathematics and assume continuity and differentiability of optimization function to guide the search for an optimal solution. Global search procedures, which include genetic algorithms, tabu search and simulated annealing; do not require differentiability of optimization function, and search for an optimal solution using heuristic strategies. The relaxation of the constraint that an optimization function has to be differentiable makes global search procedures very flexible to use in real-world decision-making situations. For example, complex single/multiple criteria classification functions for minimizing misclassification cost or minimize misclassification and information acquisition cost remain a challenge for calculus based methods, but these functions can be easily used with heuristic search strategies.

In our research, we use heuristic global search genetic algorithms to learning connection weights of a neural network. A genetic algorithm (GA) uses the *survival of the fittest* heuristic to learn connection weights of a neural network. Fig. 1 illustrates how connection weights of a neural network are represented as genes in a GA population member. The connection weights represented by horizontal arrows represent the bias connection weights.

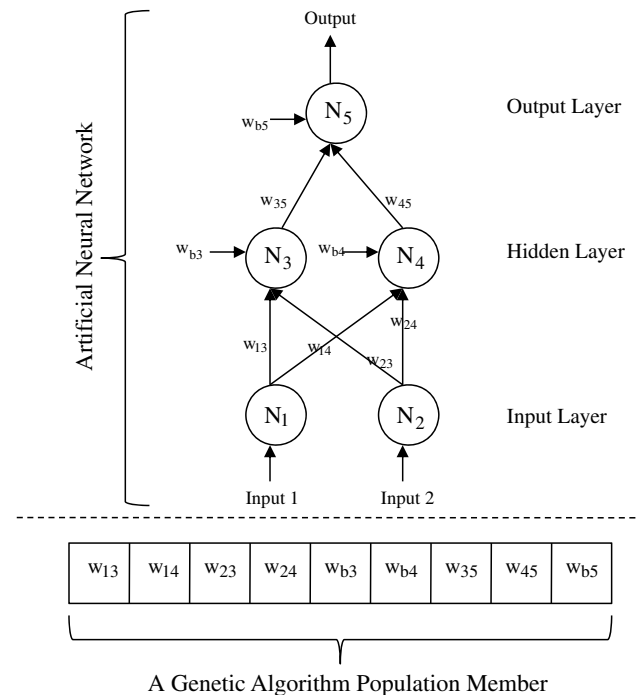


Fig. 1. A genetic algorithm representation of an artificial neural network.

A GA population consists of several population members, which work in parallel to identify a best solution. Special search operators (evaluation, selection, crossover, and mutation) are used to bias the search towards promising solutions. A GA population member may be represented using either a binary representation or a real attribute representation. In our research, we use the real-attribute representation. Using the real-attribute representation, an initial population is generated by assigning random real numbers as the values of genes in each of the population members of a GA population. Next, a fitness evaluation operator is applied to evaluate the fitness of each individual. The evaluation function can be either maximize the total number of correctly classified cases or maximize the log maximum likelihood hypothesis. After the fitness evaluation, a selection operator is applied to select the population members with higher fitness (so that they can be assigned higher probability for survival). Under selection operator, individual population members may be allowed to live or die. There are several selection operators reported in the literature. Among the popular selection operators are ranking and tournament selection. We use tournament selection operator because [Goldberg and Deb \(1992\)](#) showed that while both ranking and tournament selection maintain strong population fitness growth potential, tournament selection operator requires lower computational overhead. In a tournament selection operator two parents are selected probabilistically based on their fitness, where high fitness population members have higher chance of selection. Crossover and mutation operators are applied to these two parents with certain probabilities called crossover probability and mutation rate to generate two children. The process of selection, crossover and mutation is repeated so that the total number of children is equal to the population size. This new population of children is called the next generation population.

There are several crossover and mutation operators available in the literature. [Pendharkar and Rodger \(2004\)](#) study reports a few of these different crossover operators, and report a superior performance of a crossover operator called the arithmetic crossover. The arithmetic crossover consists of generating children in a way such that every gene in a child is a convex combination of genes from its two selected parents. We use the arithmetic crossover operator in our study. Further, we use a single gene mutation operator, where each gene in a child, with probability equal to mutation rate, is randomly changed with a random real number.

[Fig. 2](#) illustrates the GA procedure used in our research. We use two different GA based neural network models. The only difference

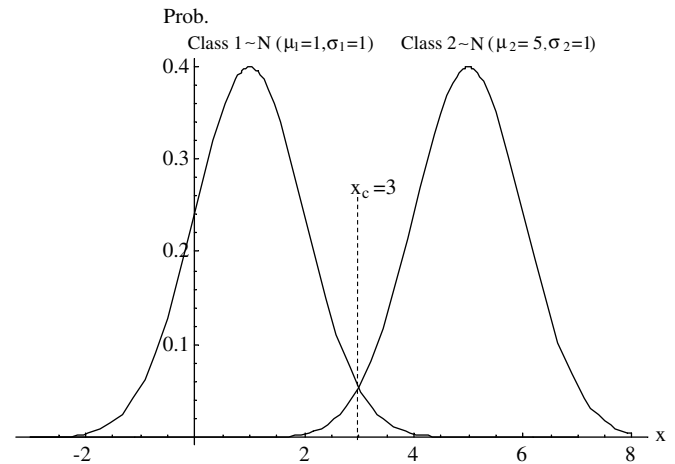


Fig. 3. Univariate statistical classification.

in the two models is the way fitness of a population member is computed. In our first model, the first fitness function maximizes the total number of correct classifications; we call this model GA-based neural network (GANN) model. In our second model, the fitness function maximizes the log maximum likelihood hypothesis (or minimizes cross entropy); we call this approach as maximum-likelihood GANN (MLGANN).

In order to benchmark the performance of our GA based NN models, we use a simple statistical z-score model. This simple statistical z-score model assumes a single continuous variable that is generated from two different normal probability density functions (pdfs) with each pdf constituting a class distribution for a two class classification problem. [Fig. 3](#) illustrates an example of such a classification problem with two classes with class means, $\mu_1 = 1$ and $\mu_2 = 5$; and standard deviations, $\sigma_1 = \sigma_2 = 1$, respectively. Since the pdf for both classes is normally distributed, the cutoff value of x , given by x_c , can be calculated by following equation ([Liang, 1992](#)).

$$x_c = \frac{\sigma_1 \mu_2 + \sigma_2 \mu_1}{\sigma_1 + \sigma_2}.$$

The cutoff value x_c provides a basic threshold for the classification of an unclassified example taking a specific value x^f . This unclassified example can be classified using the following rule ([Liang, 1992](#)).

IF $x^f \leq x_c$ THEN class 1.

Else class 2.

Since the pdfs for both classes are normal distributions, z-score values can be used to compute the probability that the example x^f belongs to a given class. For example, the z-value of the example x^f for class 1 (z^1) and class 2 (z^2) can be computed as: $z^1(\frac{x^f - \mu_1}{\sigma_1})$ and $z^2(\frac{x^f - \mu_2}{\sigma_2})$. Once the z-values are known they can be used to compute the area under the standard normal distribution to compute the probability that the example belongs to a given class.

3. Neural network design factors and model evaluation criteria

Among the factors that impact the performance of a NN are the network architecture and number of hidden nodes in a NN ([Patauwo, Hu, & Hung, 1993](#)). The architecture of a NN consists of a two-layer NN or a three layer NN. A two layer ANN is suitable for learning linear or quadratic classification functions ([Patauwo et al., 1993](#)). However, a three-layer NN can learn any desired classification function. It is possible to have more than three-layers, but the literature suggests that having more than three layers

Genetic Algorithm Procedure

Generation $\leftarrow 0$

Initialize Random Population

Evaluate Fitness of the Initial Population

Repeat

Generation \leftarrow *Generation* + 1

Tournament Selection Operation

Crossover Operation

Mutation Operation

Evaluate Fitness of New Population

If (*Generation* \geq *maximum generations*) Then Done

Until Done

Fig. 2. The pseudo-code for GA procedure used in learning connection weights.

may not result in significant performance improvements (Patuwo et al., 1993). Thus, in our study, we use a three-layer NN.

If a three-layer NN is chosen for learning classification function then a decision-maker has to select the number of hidden nodes in the hidden layer (Patuwo et al., 1993; Pendharkar, 2002). Increasing the number of hidden nodes in an NN increases the training performance for an NN, but often results in poor generalization (Patuwo et al., 1993; Pendharkar, 2002). For n inputs, Pendharkar (2002) suggests trying at least three different configurations each with the number of hidden nodes equal to n , $2n$, and $3n$.

The literature on classification provides several criteria to evaluate classification models. A common performance metric for classification problems is the accuracy measured as either the total number of correct classifications or percentage of correct classifications (Bhattacharyya & Pendharkar, 1998; Patuwo et al., 1993). The accuracy performance metric may not be a good performance metric when the data sets are biased in that they have uneven distribution of examples belonging to any one particular class (Han & Kamber, 2006). For example, a medical dataset containing 90% cases of healthy patients and 10% cases of breast cancer, a simple prediction of healthy status for all examples in the database will lead to 90% accuracy. To account for uneven class distribution in training and test datasets, two additional performance metrics called sensitivity and specificity are used in the literature (Han & Kamber, 2006).

Assume a binary classification problem with two classes, positives and negatives. Assume that a classifier correctly classifies certain examples belonging to the positive class, and the value of such correct classification is given by the variable t_{pos} . Similarly, let the correct number of classifications for the negative class be given by the variable t_{neg} . If the variables pos and neg denote the number of examples belonging to the positive and the negative class in the training data then sensitivity, specificity and accuracy performance metrics are given by the following expressions (Han & Kamber, 2006).

$$\begin{aligned} \text{sensitivity} &= \frac{t_{pos}}{pos}, \\ \text{specificity} &= \frac{t_{neg}}{neg}, \\ \text{accuracy} &= \text{sensitivity} \times \frac{pos}{(pos + neg)} + \text{specificity} \times \frac{neg}{(pos + neg)}. \end{aligned}$$

While sensitivity and specificity are useful criteria when datasets are biased, several researchers suggest that traditional criteria (accuracy, sensitivity and specificity) do not consider asymmetries in misclassification costs. Since most real-world decision making situations involve unequal misclassification costs (Baesens et al., 2002) and these misclassification costs are hard to estimate and are subject to change frequently (Fawcett & Provost, 1997), several researchers suggest developing a receiver operating characteristic (ROC) curve for a classifier. The ROC is drawn by plotting sensitivity values on y-axis and false positive rate (1-specificity) on x-axis. A ROC curve is a very robust measurement criterion that measures classifiers independent of class distribution and misclassification error cost (Tan, Steinbach, & Kumar, 2006). The ROC curve requires classifiers to generate continuous value attributes. If a classifier generates discrete output then ROC curve cannot be drawn, and only sensitivity and specificity ratios can be reported (Tan et al., 2006). Since all of our models provide continuous output, we use ROC curve to compare our classifiers.

Churn studies often use a top 10% decile lift performance metric to evaluate classifiers (Coussement and Van den Poel, 2008). To construct a top 10% decile lift, the unclassified cases are first sorted in descending order based on the likelihood that the case will be characterized as churn. Next, top 10% of cases are extracted from

the sorted set. The ratio of the percentage of correct classification of churners in this top 10% of cases with the percentage of actual churners in the entire holdout dataset provides the 10% decile lift (Coussement and Van den Poel, 2008). The higher value of top 10% decile lift is a hallmark of a good classifier. We use the top 10% decile lift performance metric to compare our classifiers.

4. Data, experiment and results

The dataset used in our study was provided by the Teradata Center for Customer Relationship Management at Duke University. The data set contained real-world customer information on 195,956 customers from a wireless company. There were six attributes for each record. These six attributes were: subscriber ID number, billing month, subscription plan, monthly total peak usage in minutes, promotional mailing variable, and churn indicator. We do not use subscriber ID number and billing month information in our research. The subscription plan variable took four discrete values as shown in Table 1. For each plan, the first column contains the actual value of variable in the dataset. The second column contains the maximum peak minutes allowed in the plan, the third column contains the monthly cost for using the peak minute service for less than or equal to peak minutes, and the fourth column contains the cost per minutes that the customer has to pay for the ongoing or new peak minute calls that extend beyond the maximum peak minutes limit covered by the plan. The promotional mailing variable and churn variables were binary variables, and the monthly total peak usage in minutes variable was a continuous variable.

For our NN classification models, we considered three variables – subscription plan, monthly total peak usage in minutes, and promotional mailing variable – as inputs, and churn variable as an output. For our z-score classification model, we computed two churn class means and standard deviations for monthly total peak usage for each of subscription plan variable value and promotional mailing variable combinations for the training dataset.¹ Depending on the unclassified example's subscription variable value and promotional mailing variable value, we used the appropriate two class means and standard deviation pair and used the example's total peak usage in minutes to compute z-values for each class. Once the z-value information was available, the example was classified into appropriate class by considering the probabilities that the example belongs to each of the two churn classes.

For our experiments, we split the original set of 195,956 examples, into five training and holdout sample pairs. Each training and holdout sample pair had approximately 70% of the original 195,956 examples in the training dataset and remaining 30% of the examples are used to create a holdout sample dataset. We used a C++ program with a random number generator to create these 5 training and holdout sample pairs. For each of 195,956 examples, the random number generator generated a random value between 0 and 1. If this random value was less than or equal to 0.7 then the example was copied into the training dataset, otherwise, it was copied to the holdout dataset. Using this approach to split the original set of 195,956 examples, we used five different random seed values to create five different training and holdout data sample pairs. Each training and holdout sample pair contained non-overlapping examples and when all the examples from a given training and holdout sample pair were combined, the total was exactly equal to original 195,956 examples.

We implemented the NN and z-score models in the C++ programming language. Everything from reading the input datasets

¹ Since subscription plan variable can take one of four discrete values and promotional mailing variable can take one of two discrete values, there were $4 \times 2 = 8$ unique subscription plan and promotional mailing variable combinations.

Table 1

The wireless services subscription plan

Subscription variable value	Number of minutes	Cost (\$)	Price/minute
1	200	30	\$0.40
2	300	35	\$0.40
3	350	40	\$0.40
4	500	50	\$0.40

(training and holdout), learning and computing the ROC curve was implemented in the source code. Additionally, for the z-score model, we implemented the source code to compute the z-score, and class probabilities associated with the z-score.

We use each of the training and holdout sample pairs for our experiments. Since we had five pairs, we performed a total of five experiments of the z-score model. For each experiment with the z-score model, we use the training dataset to learn the means and standard deviations for two classes (churn and no churn) for each of subscription variable value and promotional mailing variable value combination. Using these means and standard deviations, we classify the examples in the holdout sample. All of our experiments were conducted on a PC with a 2.8 GHZ Pentium processor and 1 GB of RAM. Table 2 illustrates the results of our z-score model. The CPU time represents the time it took to learn the two class means and standard deviations for all of subscription plan variable value and promotional mailing variable value combinations using the training data.

For our NN experiments, we use the same pair of five training and holdout sample pairs for each of GANN and MLGANN models described in Section 2. For each neural network model and pair of training and holdout data sample, we conduct three different tests – one each for number of hidden nodes equal to three, six and nine respectively. The values of GA parameters were selected by conducting initial experimentation. These values were: cross-over rate = 0.3, mutation rate = 0.1, maximum learning generations = 500, and population size = 50. Unlike the z-score model, the neural network models, depending on the number of hidden nodes, needed a learning time of between 2 h to about 6 h. Table 3 illustrates the holdout sample results for the cross-entropy objective function based MLGANN model, and Table 4 illustrates the holdout sample results for the correct classification objective function based GANN model.

The results from Tables 3 and 4 indicate that both neural network models perform very similarly in regards to correct classification criterion. Additionally, the number of hidden nodes does not appear to impact the correct classification performance of the neural network models as the correct classification percentage remains unchanged for different values of hidden nodes. However, the decile lift numbers seem to favor six hidden nodes model for the GANN model and nine hidden nodes model for the MLGANN model.

We compare the three classification models based on the ROC curve performance metric. While the ROC curve data was generated for each of our experiments, the individual ROC curves for each model were somewhat similar and we only report ROC curves

Table 2

The holdout sample results for the z-score model based classification

Experiment number	Correct classification (%)	10% Decile lift	CPU time
1	68.7	1.349	Less than 1 s
2	66.2	1.063	Less than 1 s
3	74	1.219	Less than 1 s
4	75.8	1.285	Less than 1 s
5	54.9	1.342	Less than 1 s

Table 3

The holdout sample results for the MLGANN model based classification

No. of hidden nodes	Experiment number	Correct classification (%)	10% Decile lift	CPU time in seconds
3	1	97.5	3.599	7204
	2	97.4	3.789	6737
	3	97.4	3.699	7591
	4	97.6	3.657	6790
	5	97.6	3.991	6799
6	1	97.5	3.654	14,287
	2	97.4	3.769	14,400
	3	97.4	3.780	16,415
	4	97.6	3.636	13,410
	5	97.6	3.991	14,952
9	1	97.5	3.578	19,723
	2	97.4	3.809	20,753
	3	97.4	3.834	21,771
	4	97.6	3.671	20,227
	5	97.6	3.991	19,720

Table 4

The holdout sample results for the GANN model based classification

No. of hidden nodes	Experiment number	Correct classification (%)	10% Decile lift	CPU time in seconds
3	1	97.5	3.688	8606
	2	97.4	3.803	8433
	3	97.4	3.759	8252
	4	97.6	3.693	8233
	5	97.6	3.969	8235
6	1	97.5	3.647	14,172
	2	97.4	3.816	15,577
	3	97.4	3.780	14,161
	4	97.6	3.678	14,361
	5	97.6	3.969	13,983
9	1	97.5	3.578	20,673
	2	97.4	3.749	22,032
	3	97.4	3.787	21,212
	4	97.6	3.572	20,886
	5	97.6	3.833	20,808

for our best performing experiment number 5 as both the correct classification and the decile lift values for this experiment were generally higher than the other experiments. Figs. 4–6 illustrate the ROC curves for the three different models and three different

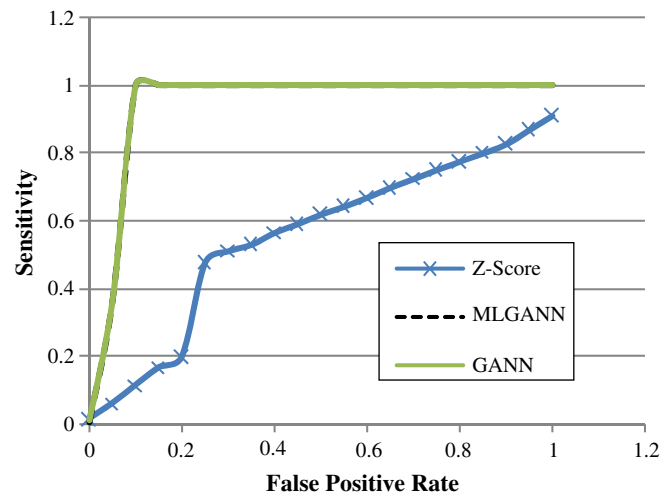


Fig. 4. The ROC curve for 3-hidden nodes NN and z-score models for 5th experiment.

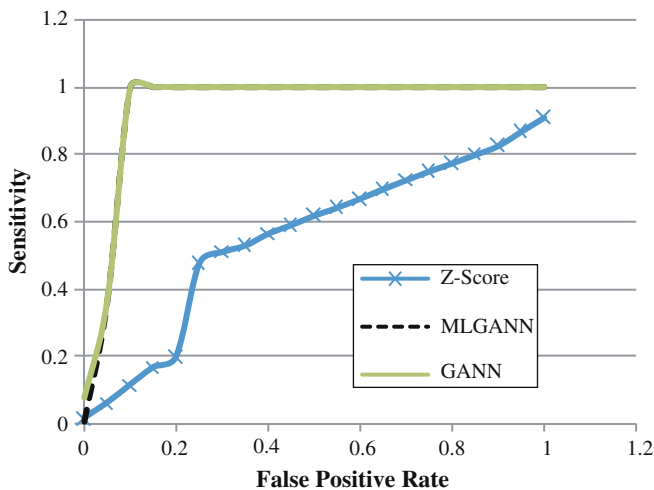


Fig. 5. The ROC curve for 6-hidden nodes NN and z-score models for 5th experiment.

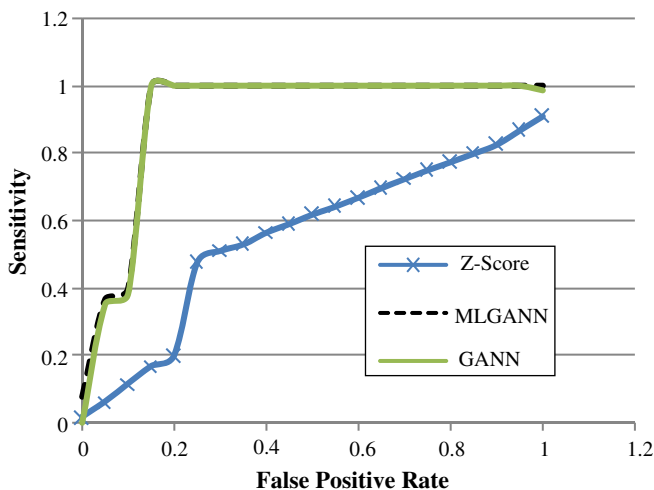


Fig. 6. The ROC curve for 9-hidden nodes NN and z-score models for 5th experiment.

numbers of hidden nodes. Since experiment number 5 was performed only once for the z-score model, the ROC curve for z-score model remains unchanged in all three figures.

When the area under the ROC curve is considered, the impact of neural network architecture (in terms of hidden nodes) becomes clearer. While both neural network models dominate the z-score model in all the performance metrics, it appears that larger size neural network models (those containing 9 hidden nodes) appear to have lower area under the ROC curve than the medium sized (6 hidden nodes) and small sized (3 hidden nodes) models. Further, the performances of two neural network models, GANN and MLGANN, were very similar. Considering all the performance metrics, we believe that the six hidden node GANN model may be a good model for our experiments. We emphasize the word “good” because our assessment does not have a strong statistical basis. Statistically, the GANN and MLGANN models do not show any significant difference in performance.

While the neural network models perform better than the z-score model, the z-score model is computationally very efficient. The learning phase of the z-score model takes less than one second, whereas, depending on the network architecture, neural network models take several hours to predict potential churners.

5. Summary, conclusions and directions for future work

We have developed and used genetic algorithm based neural networks for predicting potential churn in cellular wireless services. Using a real-world data, we have tested our genetic algorithm based neural network models and compared these models with a statistical z-score model. The results of our experiments indicate that the genetic algorithm based neural networks perform better than the z-score model, but are computationally expensive. We have also found that the number of hidden nodes in a neural network plays some role in the predictive performance. Specifically, medium size neural networks (with number of hidden nodes equal to twice the number of inputs) appear to perform well.

Since our dataset contained only three inputs, we used all the available inputs for learning connection weights for the genetic algorithm based neural network models. We believe that our fitness functions can be modified to rank the relevant inputs. For example, Baesens et al. (2002) illustrated how Bayesian approaches can be used to rank relevant inputs in neural networks. While their approach, due to the assumption that the objective function can be differentiated, cannot be directly used in our case, we believe that the GA objective function can be modified in a way that input relevance can be computed directly. We also believe that the parallel global search behavior of the GA may be beneficial in computing the input relevance directly. Input relevance ranking will increase the computational overhead of our algorithms. An efficient integration of selection of inputs and dynamic selection of number of hidden nodes in a genetic algorithm based neural networks is a potentially promising area of future research.

Acknowledgements

I thank the Teradata Center for Customer Relationship Management at Duke University for providing me the cellular wireless services data used in this research.

References

- Baesens, B., Viaene, S., Van den Poel, D., Vanthienen, J., & Dedene, G. (2002). Bayesian neural network learning for repeat purchase modeling in direct marketing. *European Journal of Operational Research*, 138(1), 191–211.
- Bhattacharyya, S., & Pendharkar, P. C. (1998). Inductive, evolutionary and neural techniques for discrimination: A comparative study. *Decision Sciences*, 29, 871–900.
- Burez, J., & Van den Poel, D. (2006). CRM at a Pay-TV Company: Using analytical models to reduce customer attrition by targeted marketing for subscription services. *Expert Systems with Applications*, 32(2), 277–288.
- Coussemont, K., & Van den Poel, D. (2008). Churn prediction in subscription services: An application of support vector machines while comparing two parameter-selection techniques. *Expert Systems with Applications*, 34(1), 313–327.
- Fawcett, T., & Provost, F. (1997). Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1–3, 291–316.
- Goldberg, D. E., & Deb, K. (1992). A comparative analysis of selection schemes used in genetic algorithms. In G. Rawlins (Ed.), *Foundations of genetic algorithms* (pp. 69–93). San Mateo, CA: Morgan Kaufmann.
- Han, J., & Kamber, M. (2006). *Data mining: Concepts and techniques*. San Francisco, CA: Morgan Kaufmann Publishers.
- Hung, S. Y., Yen, D. C., & Wang, H. Y. (2006). Applying data mining to telecomm churn management. *Expert Systems with Applications*, 31(3), 515–524.
- Liang, T. P. (1992). A composite approach to inducing knowledge for expert systems. *Management Science*, 38(1), 1–17.
- Mitchell, T. M. (1997). *Machine learning*. New York, NY: McGraw-Hill.
- Patuwo, E., Hu, M. Y., & Hung, M. S. (1993). Two group classification problem using neural networks. *Decision Sciences*, 24(4), 825–846.
- Pendharkar, P. C. (2002). A computational study on the performance of ANNs under changing structural design and data distributions. *European Journal of Operational Research*, 138, 155–177.
- Pendharkar, P. C. (2007). A comparison of gradient ascent, gradient descent and genetic-algorithm based artificial neural networks for the binary classification problem. *Expert Systems*, 24(2), 65–86.

- Pendharkar, P. C., & Nanda, S. (2006). A misclassification cost minimizing evolutionary-neural classification approach. *Naval Research Logistics*, 53(5), 432–447.
- Pendharkar, P. C., & Rodger, J. A. (2004). An empirical study of impact of crossover operators on the performance of non-binary genetic algorithm based neural approaches for classification. *Computers and Operations Research*, 31, 481–498.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition. Chapter on learning internal representations by error propagation* (Vol. 1, pp. 318–362). MIT Press.
- Sexton, R. S., Allidae, B., Dorsey, R. E., & Johnson, J. D. (1998). Global optimization of artificial neural networks: A tabu search application. *European Journal of Operational Research*, 106, 570–584.
- Sexton, R. S., Dorsey, R. E., & Johnson, J. D. (1999). Optimization of neural networks: A comparative analysis of the genetic algorithms and simulated annealing. *European Journal of Operational Research*, 114, 589–601.
- Tan, P. N., Steinbach, M., & Kumar, V. (2006). *Introduction to data mining*. Boston, MA: Addison Wesley Publishing.