ABSTRACT

In many data analytics application problems, the records in the data sets are described by a mixture of continuous and discrete variables. In the case of predictive analytics which involve assigning a sample to a class, the class membership is often determined by the interactions of a subset of these variables. In this paper we describe how the Chi2 discretization algorithm can be applied to find the right cut-offs for the relevant continuous input variables. These cut-offs correspond to the decision boundaries in the input space that are axis-parallel. When neural networks for classification are trained on input data that have been preprocessed by the Chi2 algorithm, we show accurate pruned networks with few connections and units can be obtained regardless whether the decision boundaries between the classes are axis-parallel, oblique or a combination of both.

Keywords: Neural networks, discretization, classification, decision boundary

INTRODUCTION

Before applying classification tools such as neural networks for predictive analytics, data preprocessing can often not only improve the accuracy of the prediction, but it can also yield a better understanding of data in hand. Rule extraction from trained neural networks produces classification rules that are easier for a layperson to understand how the samples are classified than the classification process of the neural networks (Tickle et al, 1998). More concise rule sets can be extracted from neural networks if the redundant units and connections have been removed by pruning and only the relevant network inputs remain.

In this paper we present the application of Chi2 algorithm as a data preprocessing tool, and the use of pruned neural networks as a classification tool. The continuous input variables in the data are discretized into a number of subintervals using Chi2 (Liu & Setiono, 1995). The main idea behind Chi2 is to represent continuous data attributes in a classification problem as discrete ones. Some classifiers may impose a restriction that the input data should have only discrete values (Liu & Tan, 1995). Some others such as Naive Bayesian classifier or decision tree methods may actually run
Hartanto and Setiono

Chi2 neural networks for predictive analytics

more efficiently or may be easier to implement if the data input attributes are all discrete. Chi2 divides the range of the continuous attributes in the data based on $\chi^2$ statistics. Two consecutive subintervals can be merged into one if the information regarding the class distribution is not lost, that is, data inconsistency introduced by this merging is still within a prespecified threshold. This means that it would still be possible to distinguish the samples from different classes with an error rate not more than this threshold.

The discretized inputs as well the original continuous values of these inputs are used in neural network training. We apply feedforward backpropagation neural networks as these networks are known to be universal function approximators (Hornik, 1989). In practice, these networks provide good predictive accuracy for a wide range of problems (Setiono, 2001). When the neural network has been trained, a network pruning processes is started. Neural network pruning will remove redundant network connections as well as those not useful inputs. As a result, the possibility of data overfitting is reduced and the pruned network can be expected to produce better prediction accuracy on new data samples than the unpruned network.

With original continuous attributes and their discretized values used as inputs to the neural network, network pruning process is even more crucial to detect and remove the many redundant inputs among those newly introduced discretized inputs as well as to prune a subset of the original continuous inputs. In the case when the decision boundaries separating samples of different classes are axis parallel, the network inputs for the original continuous inputs will be removed. On the other hand, when the decision boundaries can be better represented as linear combination of the inputs, network pruning is expected to remove the additional discretized inputs. We have shown that augmenting the input data with discretized inputs of the continuous attributes increases the network accuracy. In our earlier work (Setiono & Seret, 2012), the discretization of a continuous input attribute is achieved simply by dividing its interval into a fixed number of subintervals of equal length. Here, we utilize Chi2 to automatically determine the number of subintervals and the width of the subintervals for each continuous attributes in the data.

The outline of this paper as follows. In Section 2 we present the Chi2 algorithm for discretizing continuous input variables. Our current version of this algorithm contains a small modification of our earlier algorithm (Liu & Setiono, 1995). We present the results from discretization of 6 artificially generated data sets, some of which have been used by other researchers (Carrizosa et al, 2011). The problems represented by these data sets are binary classification problems, where the decision boundaries are axis parallel, oblique, or a combination of both. In Section 3 we report the results from training two groups of neural networks. The first group of neural networks are trained using only the original input attributes, while the second group of neural networks are trained using the original input attributes and the discretized inputs. We also report and compare the accuracy of the networks after pruning irrelevant network inputs. The networks that have been trained with additional discretized inputs consistently outperform the networks trained with only the original continuous attributes. The improvement in accuracy is even more significant for the data with axis parallel decision boundaries. The pruned networks are skeletal in the sense that they contain only
the relevant input units and a few hidden units. In Section 4 we conclude the paper.

**CHI2 FOR DISCRETIZATION OF CONTINUOUS ATTRIBUTES**

**The Chi2 algorithm**

The Chi2 algorithm (Liu & Setiono, 1995) is a widely used method for converting continuous attributes into discrete attributes. Chi2 and a number of its variants (Tay & Li, 2002; Su & Hsu, 2005; Qu et al, 2008) are based on the \( \chi^2 \) statistic for testing the null hypothesis if consecutive subintervals of an attribute value and class frequencies of the data samples that fall into these subintervals are independent. The \( \chi^2 \) test statistic is computed as follows:

\[
\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{k} \frac{(A_{ij} - E_{ij})^2}{E_{ij}} \tag{1}
\]

where:

- \( k \) = the number of classes,
- \( A_{ij} \) = the number of samples in the \( i \)th subinterval that belong in \( j \)th class,
- \( R_i \) = the number of samples in the \( i \)th subinterval, \( R_i = \sum_{j=1}^{k} A_{ij} \),
- \( C_j \) = the number of samples that belong in the \( j \)th class, \( C_j = \sum_{i=1}^{2} A_{ij} \),
- \( N \) is the total number of samples = \( \sum_{i=1}^{2} R_i = \sum_{j=1}^{k} C_j \),
- \( E_{ij} \) = the expected frequency of \( A_{ij} \) computed as \( E_{ij} = (R_i \times C_j)/N \).

The basic idea of the algorithm is to merge two rows of data (corresponding to two consecutive subintervals of one of the data continuous attributes) when the computed value of the \( \chi^2 \) statistic is below a certain threshold, that is, when the null hypothesis is not rejected. Phase 1 of the algorithm starts by placing each unique continuous value of an attribute in its own subinterval. It sets a relatively large value of significant level (sigLevel) and merges all the subintervals with \( \chi^2 \) values that falls below the corresponding threshold at this sigLevel. As long as the data consistency is maintained, the algorithm decreases sigLevel, and continues searching for subintervals that can be merged with the new threshold determined by the decreased sigLevel. Phase 1 of the algorithm ends with the same sigLevel for all attributes. It may be possible to decrease the sigLevel further for some subsets of the attributes without sacrificing data consistency. Phase 2 is designed to achieve this.

Data inconsistency occurs when two or more data sample are the same but they belong to different classes. Suppose there are \( n_1, n_2, \ldots, n_C \) samples of class \( 1, 2, \ldots, C \) that have identical attribute values, the number of inconsistencies is equal to \( N - n_\tau \) where \( N = \sum_{c} n_c \) and \( \tau \) is the class with the maximum \( n_c \).
In the original implementation of Chi2, Phase 2 is simply a finer process of Phase 1. In Phase 2, each pair of subintervals of an attribute is checked for possible merging. When it is not possible to merge any subintervals of an attribute, it simply moves on to the next attribute. In our current version, we check subintervals for possible merging not just based on its $\chi^2$ value, but also based on how many inconsistencies will be introduced when the two subintervals are merged. All subinterval-pairs with the same minimum inconsistency count will be merged as long as the data consistency is maintained within the prespecified threshold. The process is repeated by searching the next pair of subintervals with minimum inconsistency if merged, until no such pair of subintervals can be found.

Below is an outline of the two phases of our implementation of the Chi2 algorithm.

- **Phase 1:**
  ```
  set sigLevel = 0.5;
  do while (InConsistency(data) < $\delta_1$) {
    for each numeric attribute {
      Sort(attribute, data);
      chi-sq-initialization(attribute, data)
      do {
        chi-sq-calculation(attribute, data)
      } while (Merge(data))
    }
    sigLevel = decreSigLevel(sigLevel);
  }
  ```

- **Phase 2:**
  ```
  Do until no-attribute-can-be-merged {
    For each attribute that can be merged {
      Sort(attribute, data);
      Generate-chi-sq-table(attribute, data);
      minIndex = MinInconsistency(attribute, data);
      Merge(attribute, data, minIndex);
      If (Inconsistency(data) > $\delta_2$), this attribute cannot be merged;
  ```
Experimental results

We use artificially generated data sets to illustrate how the Chi2 algorithm works. Cross, chess and cube data sets are described by Carrizosa et al. (2011). The decision boundaries that separate Class 0 samples from Class 1 samples in these data sets are all axis parallel. We created X data set where the decision boundaries are oblique, and two-triangle and parallelogram-triangle data sets where a combination of oblique and axis-parallel lines form the decision boundaries. For each data set, 1000 samples were generated as training data set, and additional 1000 samples were generated to form the test data set. The algorithm Chi2 was applied to discretize the continuous attributes using only the training data samples.

   - Input: Two variables \( x_1, x_2 \in [0.4, 0.6] \).
   - Class membership:
     
     If \((x_1 - 0.5)(x_2 - 0.5) > 0\), then Class 1,
     
     else Class 0.

2. Chess data set.
   - Input: Two variables \( x_1, x_2 \in [0, 1] \).
   - Class membership:
     
     If \( x_1 \leq 0.25 \) and \( x_2 \leq 0.25 \), then Class 1,
     
     else if \( 0.5 \leq x_1 \leq 0.75 \) and \( x_2 \leq 0.25 \), then Class 1,
     
     else if \( 0.25 \leq x_1 \leq 0.5 \) and \( 0.25 \leq x_2 \leq 0.5 \), then Class 1,
     
     else if \( 0.75 \leq x_1 \) and \( 0.25 \leq x_2 \leq 0.5 \), then Class 1,
     
     else if \( x_1 \leq 0.25 \) and \( 0.5 \leq x_2 \leq 0.75 \), then Class 1,
     
     else if \( 0.5 \leq x_1 \leq 0.75 \) and \( 0.5 \leq x_2 \leq 0.75 \), then Class 1,
     
     else if \( 0.25 \leq x_1 \leq 0.75 \) and \( 0.75 \leq x_2 \), then Class 1,
     
     else if \( 0.75 \leq x_1 \) and \( 0.75 \leq x_2 \), then Class 1,
     
     else Class 0.

3. Cube data set.
   - Input: Three variables \( x_1, x_2, x_3 \in [0, 1] \).
- **Class membership:**
  
  If \( x_1 < 0.5 \) and \( x_2 > 0.5 \) and \( x_3 > 0.5 \), then Class 1,
  
  else if \( x_1 > 0.5 \) and \( x_2 < 0.5 \) and \( x_3 < 0.5 \), then Class 1,
  
  else Class 0.

4. **X data set.**
   - **Input:** Two variables \( x_1, x_2 \in [0, 1] \).
   - **Class membership:**
     
     If \(-x_1 + 0.9 \leq x_2 \) and \( x_2 \leq -x_1 + 1.1 \), then Class 1,
     
     If \( x_1 - 0.1 \leq x_2 \) and \( x_2 \leq x_1 + 0.1 \), then Class 1,
     
     else Class 0.

5. **Double-triangle data set.**
   - **Input:** Two variables \( x_1, x_2 \in [0, 1] \).
   - **Class membership:**
     
     If \( x_2 \geq 0.6 \) and \( x_2 \geq 0.4x_1 + 0.6 \), then Class 1,
     
     else if \( x_2 \leq 0.4 \) and \( x_2 \geq -0.4x_1 + 0.4 \), then Class 1,
     
     else Class 0.

6. **Parallelogram-triangle data set.**
   - **Input:** Two variables \( x_1, x_2 \in [0, 1] \).
   - **Class membership:**
     
     If \( x_1 \leq 0.5 \) and \( x_2 \geq x_1 \) and \( x_2 < x_1 + 0.5 \), then Class 1,
     
     else if \( x_2 \leq 0.5 \) and \( x_2 > 1 - x_1 \), then Class 1,
     
     else Class 0.

The training data samples are depicted in Figures 1 and 2 which show the two groups of samples distinguished according to the values of the relevant variables. The Chi2 algorithm was run with threshold values set to \( \delta_1 = 0.0 \) in Phase 1 and \( \delta_2 = 0.05 \) in Phase 2.

The subintervals found by Chi2 are summarized in Tables 1 and 2. For the data sets with only axis parallel decision boundaries (cross, chess, cube), the Chi2 algorithm encountered no difficulty in finding the right cut-offs. For example, for the cross data set, there is one cut-off each for inputs \( x_1 \) and \( x_2 \). The cut-off value 0.49866 implies that the original range of the continuous attribute \( x_1 \) is divided into 2 subintervals: \([0, 0.49866), [0.49866, 1]\). Similarly, the cut-off value of 0.50050
for $x_2$ indicates that knowing whether a sample falls in the first subinterval $[0, 0.50050]$ or in the second subinterval $[0.50050, 1]$ still allows a classifier to differentiate between Class 0 and Class 1 samples correctly. The Chi2 algorithm indeed found the correct subintervals in the data.

Figure 1: Artificially generated data sets: a circle represents a Class 0 sample, while a blue diamond represents a Class 1 sample. The boundaries between classes are axis-parallel in cross, chess and cube data sets, and oblique in X data set.
For the cube data set, cut-off values close to the correct values of 0.5 for the three attributes $x_1$, $x_2$, and $x_3$ were discovered by Chi2. These cut-offs allow neural networks trained with the discretized inputs to obtain high predictive accuracy rates, while network pruning process removes the original continuous attributes as they are no longer needed.

For data sets with oblique decision boundaries, the range of the continuous attributes was divided into many more subintervals in order to maintain data consistency as expected. The algorithm, however managed to find the cut-offs that are relevant for classification among these many subintervals. For attribute $x_2$ in the double-triangle data set, the cut-off values of 0.40113 and 0.62141 were obtained. For the parallelogram-triangle data set, the important cut-offs are 0.50410 for $x_1$ and 0.51374 for $x_2$.

![Double-triangle data set.](image)

![Parallelogram-triangle data set.](image)

Figure 2: Artificially generated data sets: a circle represents a Class 0 sample, while a blue diamond represents a Class 1 sample. The boundaries between classes are a mixture of axis-parallel and oblique lines.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Cross</th>
<th>Chess</th>
<th>Cube</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_1$</td>
</tr>
<tr>
<td></td>
<td>0.49866</td>
<td>0.50050</td>
<td>0.25453</td>
</tr>
</tbody>
</table>

Table 1: The cut-offs obtained by the Chi2 discretization algorithm for cross, chess and cube data sets.
Table 2: The cut-offs obtained by the Chi2 discretization algorithm for X, double-triangle and parallelogram-triangle data sets.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Original inputs</th>
<th>Discretized inputs</th>
<th>( I_3 )</th>
<th>( I_4 )</th>
<th>( I_5 )</th>
<th>( I_6 )</th>
<th>( I_7 )</th>
<th>( I_8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.09827</td>
<td>0.13122</td>
<td>0.10943</td>
<td>0.09730</td>
<td>0.12267</td>
<td>0.19843</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.16007</td>
<td>0.23736</td>
<td>0.24117</td>
<td>0.29707</td>
<td>0.15186</td>
<td>0.38612</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.20358</td>
<td>0.35714</td>
<td>0.53059</td>
<td>0.40113</td>
<td>0.18517</td>
<td>0.51374</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.31161</td>
<td>0.49327</td>
<td>0.77080</td>
<td>0.62141</td>
<td>0.33507</td>
<td>0.82183</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.40772</td>
<td>0.51512</td>
<td>0.77904</td>
<td>0.71725</td>
<td>0.42265</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.48611</td>
<td>0.64170</td>
<td>0.79686</td>
<td>0.83896</td>
<td>0.42444</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.50470</td>
<td>0.72723</td>
<td>0.85735</td>
<td></td>
<td>0.45104</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.55676</td>
<td>0.82868</td>
<td>0.91164</td>
<td></td>
<td>0.50410</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.60583</td>
<td>0.90269</td>
<td></td>
<td></td>
<td>0.55305</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.72067</td>
<td></td>
<td></td>
<td></td>
<td>0.65030</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.76137</td>
<td></td>
<td></td>
<td></td>
<td>0.67906</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.86389</td>
<td></td>
<td></td>
<td></td>
<td>0.72410</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.89952</td>
<td></td>
<td></td>
<td></td>
<td>0.72314</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.96890</td>
<td></td>
<td></td>
<td></td>
<td>0.86301</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NEURAL NETWORK TRAINING WITH AUGMENTED DISCRETE INPUT VARIABLES**

**Encoding the discretized inputs**

When the original continuous input variable is divided by Chi2 algorithm into \( N \) subintervals, \( N - 1 \) binary inputs are added for training the neural network. For example, there are 4 subintervals in the chess data set for inputs \( x_1 \) and \( x_2 \) with cut-offs equal to 0.25453, 0.51164 and 0.75526. An input \( x_1 \) that is less than 0.25453 will be represented as \((1, 1, 1)\), an input between 0.25453 and 0.51164 as \((0, 1, 1)\), an input between 0.51164 and 0.75526 as \((0, 0, 1)\), and an input greater than 0.75526 as \((0, 0, 0)\). Below is a table that illustrates how 4 data samples in \( x_1 \) and \( x_2 \) are augmented with the additional discretized inputs \( I_3, I_4, \ldots I_8 \):
Hartanto and Setiono

Chi2 neural networks for predictive analytics

Figure 3: A three-layer feedforward neural network with 5 input units, 3 hidden units and 2 output units.

With the additional discretized inputs, the total number of input units required in the neural networks is 8.

**Network training and pruning**

The three-layer feedforward neural network consists of an input layer, a hidden layer and an output layer (Figure 3). The input layer and the hidden layer in turn consists of one or more processing units. Each network unit processes its inputs and generates an output value which is transmitted to the unit in the next layer. The classification of the input samples depends on the values obtained at the output unit.

A penalty term is added to the usual sum of squared error function during network training to encourage weight decay. Redundant network connections are expected to have weights with small magnitude that when they are removed, the accuracy of the network is not adversely affected. The penalty function we use is the quadratic penalty function

\[
P(W, V) = \epsilon \left( \sum_{h=1}^{H} \sum_{j=1}^{n} W_{h,j}^2 + \sum_{m=1}^{O} \sum_{h=1}^{H} V_{m,h}^2 \right),
\]

where \(\epsilon\) is a small positive parameter.
Redundant connections in the neural network are removed in three steps. First, hidden units are checked for possible removal. When there is no more hidden units that can be removed without dropping the accuracy below a preset threshold, the algorithm attempts to remove individual input units in the second step. Input units that can removed corresponds to input data attributes that are not useful for classification. Hence, we accomplish feature selection by removing such input units. Finally, in the third step the algorithm tries to remove one network connection at a time.

The outline of our pruning algorithm is as follows:

- Hidden unit removal:
  
  i. Group the hidden units into two subsets: $\mathcal{P}$ and $\mathcal{Q}$, these are the sets of hidden units that are still present in the network and those that have been checked for possible removal in the current stage of pruning, respectively. Initially, $\mathcal{P}$ corresponds to all the hidden units in the trained network and $\mathcal{Q}$ is the empty set.
  
  ii. Save a copy of the weight values of all connections in the network.
  
  iii. Find a set of connections from the input units to the hidden unit $h \in \mathcal{P}$ and $h \notin \mathcal{Q}$ such that when the weight values of the connections from the input units to $h$ are set to 0, the accuracy of the network is least affected.
  
  iv. Set the weights for network connections from the input units to the hidden unit $h$ to 0 and retrain the network.
  
  v. If the accuracy of the network is still satisfactory, then
     
     (a) Remove $h$, i.e. set $\mathcal{P} := \mathcal{P} - \{h\}$.
     
     (b) Reset $\mathcal{Q} := \emptyset$.
     
     (c) Go to Step (ii).
  
  vi. Otherwise,
     
     (a) Set $\mathcal{Q} := \mathcal{Q} \cup \{h\}$.
     
     (b) Restore the network weights with the values saved in Step (ii) above.
     
     (c) If $\mathcal{P} \neq \mathcal{Q}$, go to Step (ii). Otherwise, Stop.

- Input unit removal done in similar way as for hidden unit removal above.

- Network connection removal done in similar way as for hidden unit removal above.

**Experimental results**

We trained and pruned 10 neural networks for each data set. The error backpropagation method (Bishop, 1995; Mitchell, 1997) was applied to find the minimum of the network error function.
Hartanto and Setiono

Chi2 neural networks for predictive analytics

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Before pruning</th>
<th>After pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training Set</td>
<td>Test Set</td>
</tr>
<tr>
<td>Cross</td>
<td>63.57</td>
<td>61.38</td>
</tr>
<tr>
<td>Chess</td>
<td>54.95</td>
<td>53.38</td>
</tr>
<tr>
<td>Cube</td>
<td>89.68</td>
<td>88.82</td>
</tr>
<tr>
<td>X</td>
<td>91.81</td>
<td>90.08</td>
</tr>
<tr>
<td>Double-triangle</td>
<td>65.38</td>
<td>65.48</td>
</tr>
<tr>
<td>Parallelogram-triangle</td>
<td>84.39</td>
<td>84.77</td>
</tr>
</tbody>
</table>

Table 3: The average accuracy (in %) of 10 neural networks trained with the original continuous inputs.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Before pruning</th>
<th>After pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training Set</td>
<td>Test Set</td>
</tr>
<tr>
<td>Cross</td>
<td>99.38</td>
<td>99.10</td>
</tr>
<tr>
<td>Chess</td>
<td>97.75</td>
<td>96.20</td>
</tr>
<tr>
<td>Cube</td>
<td>97.65</td>
<td>97.81</td>
</tr>
<tr>
<td>X</td>
<td>95.08</td>
<td>90.04</td>
</tr>
<tr>
<td>Double-triangle</td>
<td>95.98</td>
<td>90.90</td>
</tr>
<tr>
<td>Parallelogram-triangle</td>
<td>95.50</td>
<td>91.73</td>
</tr>
</tbody>
</table>

Table 4: The average accuracy (in %) of 10 neural networks trained with additional discretized inputs obtained by Chi2.

The learning rate was set to 0.1. In order to reduce computation time, the training process was terminated after 50 epochs, while network retraining during the pruning process was limited to a maximum of 10 epochs. The results from training the neural networks with the original data are summarized in Table 3, while the results from training with the original data plus the discretized inputs are summarized in Table 4. The rates were computed as averages from 10 neural networks.

Comparing the numbers in these two tables, we note that the accuracy rates of the unpruned networks were much higher when they had been trained with the additional discretized inputs. Within just 50 epochs, an accuracy rate of more than 90% was achieved on both the training data set and the test data set. On the other hand, having just the original continuous attributes, the neural networks were not successful in obtaining good accuracy, specially for data set where the data samples were separated by axis parallel decision boundaries such as the cross and the chess data sets.

After pruning, the networks trained with the discretized inputs maintained their high accuracy rates. For 5 of the 6 data sets, the predictive accuracy rate of the pruned networks with discretized inputs were at least 90% and higher than the predictive accuracy of the pruned networks with

671862-12
Table 5: The average number of hidden units $H$, input units $I$ and connections $C$ in the original networks and pruned networks.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Before pruning</th>
<th>After pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$H$</td>
<td>$I$</td>
</tr>
<tr>
<td>Cross</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Chess</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Cube</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>X</td>
<td>8</td>
<td>25</td>
</tr>
<tr>
<td>Double-triangle</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>Parallelogram-triangle</td>
<td>8</td>
<td>21</td>
</tr>
</tbody>
</table>

original continuous inputs. The exception is the X data set where the two classes of data samples are separated by oblique decision boundaries.

With the additional discretized inputs, the original inputs and some or all of the newly added discretized inputs may be redundant. The pruning process are designed to remove them. The effect from network pruning on the network structure is summarized in Table 5. In this table, we show the average number of hidden units, input units and network connections when pruning process was terminated. In general the pruning process was successful in eliminating or redundant network inputs and connections. The one data set for which the number of connections and inputs remained large after pruning is X data set. A possible explanation for this result is the relatively large number of subintervals found by Chi2 for the two original continuous inputs $x_1$ and $x_2$ as the data samples are separated by four oblique decision boundaries.

CONCLUSION

We proposed an effective system for predictive analytics that consists of two components. The first component is the Chi2 algorithm for data preprocessing. This algorithm divides the continuous interval of a data attribute into a number of subintervals. Using $\chi^2$ statistics, it finds cut-off values such that if data attribute values that fall in the same subinterval are replaced by the same discretized value, data consistency is maintained. Adding the discretized attribute values as inputs helps the second component of the system learn the data patterns faster. We use feedforward error backpropagation neural networks as the second component of our system. Such networks have been shown to be effective in analyzing data for classification and regression in a wide variety of problem domains.

Our experimental results involving six artificially generated data sets show that by adding the discretized inputs, neural networks can be trained to achieve high accuracy rates in relatively small number of epochs. The pruning process that followed removes the redundant network inputs that may originally be present in the data or that may be introduced by the additional discretized inputs.
After being pruned, the networks have only few relevant input and hidden units left. Algorithms that extract classification rules could then be applied to these networks so that the decision made by the networks to classify a sample as Class 0 or Class 1 can be expressed as comprehensible set of rules.

Our future works include conducting more experiments on combining input discretization with neural networks for classification and regression. We will be testing the proposed approach on real world data arising from business applications areas such as credit scoring, bankruptcy prediction, consumer preferences rating and customer churn analysis. For the Chi2 algorithm itself, we plan to extend it so that it can handle not only data with continuous attributes, but also data with mix attributes. As for neural network training and pruning, faster and more effective training and pruning algorithms are being explored.

REFERENCES


