Fragmentation Problem and Automated Feature Construction

Rudy Setiono and Huan Liu
School of Computing
National University of Singapore
Lower Kent Ridge Road, Singapore 119260, Singapore
{rudys, liuh}@comp.nus.edu.sg

Abstract

Selective induction algorithms are efficient in learning target concepts but inherit a major limitation - each time only one feature is used to partition the data until the data is divided into uniform segments. This limitation results in problems like replication, repetition, and fragmentation. Constructive induction has been an effective means to overcome some of the problems. The underlying idea is to construct compound features that increase the representation power so as to enhance the learning algorithm's capability in partitioning data. Unfortunately, many constructive operators are often manually designed and choosing which one to apply poses a serious problem itself. We propose an automatic way of constructing compound features. The method can be applied to both continuous and discrete data and thus all the three problems can be eliminated or alleviated. Our empirical results indicate the effectiveness of the proposed method.

1 Introduction

Selective induction algorithms all adopt a divide-and-conquer strategy in learning target concepts. That is, a selective induction algorithm selects the best feature among all to partition the data into uniform data segments in terms of classes. If a data segment is not sufficiently pure, another feature is selected to further partition it. The major advantage of this divide-and-conquer strategy is *efficiency*. The gain in efficiency comes with a price since a selective induction algorithm inherits problems of replication, repetition, and fragmentation. Let us use decision tree induction as an example of selective induction algorithms to elucidate what these problems are. Throughout this paper, we will use C4.5 [17] to illustrate the consequences of the problems and the ways to overcome them. Let the classes be denoted $\{C_1, C_2, ..., C_k\}$. To construct a decision tree from a dataset D, briefly, C4.5 recursively executes one of the following steps: (1) if D contains one or more examples, all belonging to class C_j. Stop. (2) if D contains no example, the most frequent class at the parent of this node is chosen as the class. Stop. or (3) if D contains examples belonging to a mixture of classes, information gain is then used as a heuristic to split D into partitions (branches) based on the values of a *single feature*. The replication problem can be observed if subtrees are replicated in a decision tree; the repetition (or repeated testing) problem is present if features are repeatedly tested (more than once) along a path in a decision tree; and the fragmentation problem exists if data is gradually partitioned into small fragments [14, 3]. Replication and repetition imply fragmentation, but fragmentation may occur without any replication or repetition if many features need to be tested [7].

Can we adhere to divide-and-conquer, but eliminate or at least alleviate the problems that come with it? One way is to construct compound features aiming at fewer divide-and-conquer steps. An extremely successful case would be when the target concept is actually described by one compound feature. In such a case, we would need only one divide-and-conquer step to learn the target concept.

Feature construction has been proposed as a solution to the problems of repetition, replication and fragmentation [9, 15, 12, 22]. Methods of constructive induction can be classified on the basis of the source of information that is used for searching for the compound features. Many forms of constructive induction exist [10]. There is data-driven constructive induction (DCI) in which search is based on the analysis of the input data; there is hypothesis-driven constructive induction (HCI) in which search is based on the analysis of the intermediate hypotheses; and there is knowledge-driven constructive induction (KCI) in which search relies on domain knowledge provided by the expert. Donoho and Rendell [5] studied the use of fragmentary knowledge in constructive induction.

This work is about data-driven constructive induction. The common approach to data-driven constructive induction has been to apply many different logical and mathematical operators to the original features to create new candidate features. The candidate features that score high on an attribute quality function are added to the original feature set, and the whole set is employed in the process of inductive generalization [2]. Approaches of this type share the following properties: (1) they rely on constructive operators; and (2) the majority of the existing work deals only with discrete or boolean features. Their success depends on whether there is a right set of constructive operators. The number of operators we could try can be unlimited. It is unrealistic to try all available operators for a particular application. Finding a right set of operators to try is itself an intractable problem. Therefore, instead of collecting a large set of operators that may ever grow, we propose an automated way of constructing compound features.

Our tool for automated feature construction is the standard feedforward neural networks with a single hidden layer. Such neural networks have been successfully applied in a wide variety of areas. They are easy to train and excellent neural network packages are available both commercially and from the internet. An attractive aspect of neural networks is that they generally perform well regardless of the types of the data attributes. As a result, we are able to construct features for datasets that have continuous, discrete, or mixed attributes. Three illustrative examples that we present in Sections 2 and 3 of this paper highlight the capability of our neural network based feature construction method to obtain new compound features when the original input of the data contain attributes of different types.

2 Fragmentation problem

We choose three data sets to illustrate the problem and exhibit that replication and repetition imply fragmentation and show that fragmentation can exist by itself. In the next section, we will show how these problems are either eliminated or alleviated by using compound features constructed automatically. The three datasets are:

- DNF9b dataset [21] which has 9 binary features $x_1, x_2, \ldots, x_9$. The 512 instances are labeled as follows: (a) Class 1: $x_1x_2x_3 + \overline{x}_1\overline{x}_2 + x_7x_8x_9 + \overline{x}_7\overline{x}_9$, (b) Class 2: Otherwise.

- Iris dataset [6] which has 150 instances described by 4 continuous attributes: sepal_length ($\mathcal{A}_1$), sepal_width ($\mathcal{A}_2$), petal_length ($\mathcal{A}_3$), and petal_width ($\mathcal{A}_4$). Each pattern belongs to one of the 3 possible classes: setosa, versicolor, or virginica.

- Function 9 dataset [1]. The function involves 4 attributes: (1) Salary: uniformly distributed from 20000 to 50000, (2) Commission: is 0 if Salary is greater than 75000, it is uniformly distributed from 10000 to 75000 otherwise, (3) Elevel: discrete and uniformly chosen from $[0, 1, 2, 3, 4]$, (4) Loan: uniformly distributed from 0 to 50000.

  We generate the patterns randomly according to the function definition: (a) Class 1: *0.67(Salary + Commission) - 5000 Elevel - 0.2 Loan > 10000*, (b) Class 2: Otherwise.

In the decision trees built by C4.5, the DNF9b dataset exhibits the replication problem and the Iris dataset exhibits the repetition problem despite the relatively small number of patterns in both sets. Function 9 dataset shows that having more patterns does not help, in fact, it causes the fragmentation problem to be even more serious.

2.1 Replication implies fragmentation

Figure 1 shows a decision tree generated by C4.5 on data *DNF9b*. The tree contains 33 nodes and many identical subtrees, although the accuracy of the tree is 100%[1]. The most apparent replication is subtrees, ST1, ST2, and ST3 (as indicated in the tree), which appear at different levels, and each contains 128, 128, and 64 instances, respectively. Had replication not occurred, there could be 320 instances for subtree ST induction. The sample shortage may cause sever problem in inducing reliable decision trees. What is observed here is a replication problem that causes the data to be partitioned into small segments.

What can help us solve this replication problem? One way is to build compound features. In [14], for example, the authors proposed a system called FRINGE for finding high level (compound) features based on the decision tree built using individual features. With the new additional features, FRINGE builds the tree again.

2.2 Repetition implies fragmentation

In Figure 1, we observe replication that causes fragmentation, but no trace of repetition since no feature is tested more than once along any branch. A decision tree built by C4.5 on Iris data displays the repetition

[1] For the illustrative examples, accuracy estimation is not an issue so the whole data is used in inducing the tree. Our focus is on observing the replication problem and later the repetition problem.

problem. In the tree shown in Figure 2, features A3 and A4 have been repeatedly tested. These repetitions split data into smaller and smaller segments, hence result in fragmentation.

One way of solving the repetition problem is to build an oblique tree such that a test is performed on a linear combination of some features instead of on a single feature. OC1 [13] is one of the methods that builds this type of decision trees.

2.3 Fragmentation on its own

Using information gain to branch a node in decision tree induction will more likely choose a feature with many values, which is supported by the analysis done by Quinlan [16]. Researchers proposed some solutions to avoid choosing a feature with many values: (1) using gain ratio instead of gain; and (2) grouping feature values first so that there are two outcomes for a feature and then using gain (see the details in [17]). The question we ask is what is wrong with choosing a feature with many values. For example, the feature 'Age of Patient' could have many distinct values and using this feature to branch could cause quick decrease of the data in each branch - the fragmentation problem. Let us have two features A and B. A has 10 values and B has 2 values. Branching the data at A will virtually divide the data into 10 partitions, but one branch of B can have at least half of the data. If we assume the data to be evenly distributed, one tenth of the data is certainly less reliable than one half of it. Here we experience a case of pure fragmentation without seeing repetition or replication. In general, given a data set and two learning algorithms, if one learning algorithm generates a decision tree with more internal nodes than the other, it is a clear symptom that it suffers more from the fragmentation problem since every internal node splits data further.

2.4 Adverse effects of fragmentation

Function 9 data highlights the shortcomings of decision tree methods which are not equipped with the capability of constructing new compound features. The C4.5 unpruned tree generated using 500 training patterns shows that all continuous attributes are tested more than once on practically all paths from the root node to the leaf nodes. The most frequent split that produces the leaf nodes on these branches is a five-way split on attribute *Elevel*. Since the number of training patterns is relatively small, some leaf nodes actually do not have any pattern at all. All the leaf nodes of the corresponding pruned tree do contain some patterns, hence providing some support to the hypotheses represented by the splitting of the nodes along the tree branches. Tree pruning reduces the size of the tree

```
A1 = 0:
|   A2 = 0: 1 (128.0)
|   A2 = 1:
|   |   (ST1)
|   |   A7 = 0:
|   |   |   A9 = 0: 1 (32.0)
|   |   |   A9 = 1: 0 (32.0)
|   |   A7 = 1:
|   |   |   A8 = 0: 0 (32.0)
|   |   |   A8 = 1:
|   |   |   |   A9 = 0: 0 (16.0)
|   |   |   |   A9 = 1: 1 (16.0)
A1 = 1:
|   A2 = 0:
|   |   (ST2)
|   |   A7 = 0:
|   |   |   A9 = 0: 1 (32.0)
|   |   |   A9 = 1: 0 (32.0)
|   |   A7 = 1:
|   |   |   A8 = 0: 0 (32.0)
|   |   |   A8 = 1:
|   |   |   |   A9 = 0: 0 (16.0)
|   |   |   |   A9 = 1: 1 (16.0)
|   A2 = 1:
|   |   A3 = 1: 1 (64.0)
|   |   A3 = 0:
|   |   |   (ST3)
|   |   |   A7 = 0:
|   |   |   |   A9 = 0: 1 (16.0)
|   |   |   |   A9 = 1: 0 (16.0)
|   |   |   A7 = 1:
|   |   |   |   A8 = 0: 0 (16.0)
|   |   |   |   A8 = 1:
|   |   |   |   |   A9 = 0: 0 (8.0)
|   |   |   |   |   A9 = 1: 1 (8.0)
```

Figure 1: C4.5 decision tree for DNF9b dataset.

```
A3 <= 1.9 : 1 (50.0)
A3 > 1.9 :
|   A4 > 1.7 : 3 (46.0/1.0)
|   A4 <= 1.7 :
|   |   A3 > 5.3 : 3 (2.0)
|   |   A3 <= 5.3 :
|   |   |   A3 <= 4.9 : 2 (48.0/1.0)
|   |   |   A3 > 4.9 :
|   |   |   |   A4 <= 1.5 : 3 (2.0)
|   |   |   |   A4 > 1.5 : 2 (2.0)
```

Figure 2: C4.5 decision tree for Iris dataset.

from 66 to 25 nodes. However, this is achieved with a cost. Although the attribute *Elevel* is relevant, it is absent in the pruned tree. As a result, the classification accuracy of the tree drops from 98.0 % to 95.6 %. This example illustrates how the fragmentation problem has caused sample shortage. It follows that a relevant attribute is removed from the decision tree and inaccurate hypotheses are induced.

For this problem, using more patterns in the training dataset improves the accuracy slightly. With 1000 patterns used to build the tree, all the relevant features are present in the pruned tree and the classification accuracy becomes to 97.1 %. The size of the tree increases to 57, more than twice the size of the tree constructed using half the patterns and the fragmentation problem persists. The figures in the table below show that having more training patterns reduces the error rate, but it increases the tree size significantly. It is very difficult, if not impossible, to comprehend trees with hundreds of nodes.

Training patterns	Tree size	Classification Error
500	25	4.4 %
1000	57	2.9 %
2000	121	1.9 %

Table 1: Increasing the number of patterns helps improve the accuracy, but at the cost of increasing the tree size.

3 Automatic feature construction

Replication and repetition problems have been demonstrated through three datasets in the previous section. Replications and repetitions are undesirable because they cause the fragmentation problem which can quickly weaken the statistical reliability of the data. The decision trees become unnecessarily large, less intelligible, or less accurate. It is important therefore to attack the fragmentation problem by removing replications and repetitions and avoiding branching on features with many values. Many solutions to these problems have been investigated such as global data analysis [21], constructive induction [10], multivariate or oblique decision trees [8], and discretization [4].

All these have been studied in isolation. This work proposes a solution based on constructive induction and global data analysis to create compound features. The solution does not require feature discretization, and compound features can be used in generating multivariate decision trees.

We apply the standard feedforward neural networks with a single hidden layer [18] to construct new features. The choice of using neural networks as a tool is based on the inherent differences between neural networks and decision tree methods for pattern classification. All the patterns in a dataset are contributing information to the updating of the network weights during training. Decision tree methods, on the other hand, have smaller amount of data for the tree nodes that are farther away from the root node due to fragmentation. Most decision tree methods also consider a single best attribute to split the data during tree generation. In contrast, neural networks consider the contribution of all attributes of the input data as a group during training. Clearly, many datasets contain attributes that are redundant and/or irrelevant. Neural network training can be finetuned so that such attributes can be identified and removed after the training has terminated. The algorithm used for network pruning is described in our previous work [19].

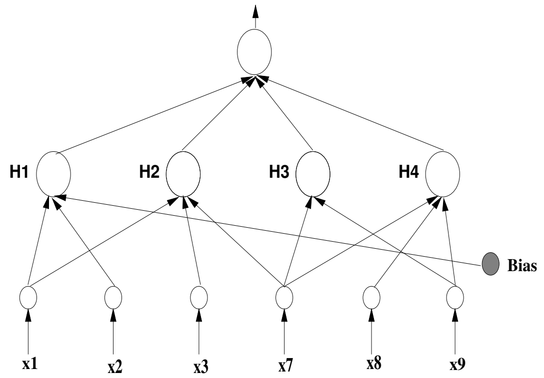

Figure 3: A neural network for DNF9b dataset.

We return to the three examples in the previous section to illustrate how new features can be constructed.

1. DNF9b.

A fully connected network was trained until all 512 patterns are correctly classified. Redundant and irrelevant connections were pruned while the accuracy is maintained. The resulting pruned network is shown in Figure 3.

There are 4 hidden units left in the pruned network, this number is the dimensionality of the new features. In other words, from a dataset with 9 binary attributes, we have obtained a dataset with 4 continuous features. These new features are the hidden unit activations of the network with their values range in the interval $[-1, 1]$. An activation value for input x^i is the hyperbolic tangent of the weighted original fea-

```
H3 <= 0 : 1 (128.0)
H3 > 0 :
|   H4 > 0.30 : 1 (64.0)
|   H4 <= 0.30 :
|   |   H2 <= -0.07 : 0 (120.0)
|   |   H2 > -0.07 :
|   |   |   H1 <= -0.61 : 0 (80.0)
|   |   |   H1 > -0.61 : 1 (120.0)
```

Figure 4: C4.5 decision tree built using network hidden activations as input.

tures:

$$H^i = \delta(\sum_j x^i_j w_j), \quad \delta(\xi) = (e^{\xi} - e^{-\xi})/(e^{\xi} + e^{-\xi}). \quad (1)$$

A classification method such as C4.5 or k-nearest neighbor can then be applied to the dataset with the newly constructed features. C4.5 is chosen as the classification method since it will enable us to see the effect of taking the extra step of feature construction.

We denote the activation values in the 4 hidden units by H1, H2, H3, and H4. The decision tree generated when these activation values were used as input is depicted in Figure 4. The tree is very much smaller than that shown in Figure 1. Given this decision tree, we can remove the nonlinearity from the activation function (1) by the fact that the hyperbolic tangent function is a one-to-one function. An equivalent tree can be obtained by simply applying the inverse of this function on both sides of each node splitting condition. The relevant values for node splitting are $0, 0.30, -0.07$ and -0.61. We apply the inverse of the hyperbolic function to these 4 values and get $0, 0.31, -0.07, -0.70$, respectively. An equivalent tree where the nonlinearity of the activation function has been removed from each node splitting condition is shown in Figure 5.

For a dataset that contains only discrete features, it may be desirable to obtain splitting conditions in terms of compound features that do not involve the neural network weights. For the tree in Figure 5, we trace the path form the root to each of the leaves:

H*3. The splitting condition is

$$3.70x_7 + 3.70x_9 \leq 0$$

Clearly, H*3 ≤ 0 if and only if $x_7 = x_9 = 0$.

H*4. The splitting condition is

$$0.15x_7 + 0.15x_8 + 0.15x_9 \geq 0.31$$

```
H*3 <= 0 : 1 (128.0)
H*3 > 0 :
|   H*4 > 0.31 : 1 (64.0)
|   H*4 <= 0.31 :
|   |   H*2 <= -0.07 : 0 (120.0)
|   |   H*2 > -0.07 :
|   |   |   H*1 <= -0.70 : 0 (80.0)
|   |   |   H*1 > -0.70 : 1 (120.0)
```

Figure 5: An equivalent tree for DNF9b dataset where the left hand side of each condition is a linear inequality involving the original input features.

It is true that H*4 ≥ 0.31 if and only if $x_7 = x_8 = x_9 = 1$.

H*2. The splitting condition is

$$3.13x_1 - 3.20x_2 + 0.65x_3 \leq -0.07$$

In order to have H*2 ≤ -0.07, x_2 must be 1 and 1 of the other 2 inputs, x_1 or x_3 must be zero.

H*1. The splitting condition is

$$-1.45x_1 + 0.95x_2 + 0.73 \leq -0.71$$

For this hidden unit, 0.73 is the bias or threshold. Only one possible combination of x_1 and x_2 will give H*2 ≤ -0.71, $x_1 = 1$ and $x_2 = 0$.

Hence, when the network weights have been removed, we obtain rather complex new compound features with binary (Yes or No) values. From the hidden units H3, H4, H2 and H1, we obtain the new features $\overline{x}_7\overline{x}_9$, $x_7x_8x_9$, $x_2\overline{x}_1 + x_2\overline{x}_3$, and $x_1\overline{x}_2$, respectively. It is worth noting that some of these new features correspond to the M-of-N concept [20] present in the dataset. For both $H3$ and $H4$, the identical magnitudes of the weights certainly suggest that equivalent M-of-N rules can be used to differentiate the values of the new features. From input weights of unit H3, we conclude that if 2 of $(\overline{x}_7, \overline{x}_9)$ are true, then Class 0, while from those of hidden unit H4, if 3 of (x_7, x_8, x_9) are true, then Class 1. We show yet another equivalent of the decision tree with branching conditions given in terms of the compound features in Figure 6.

2. Iris

The neural network shown in Figure 7 was constructed using 150 patterns in the dataset. All but one of these patterns are correctly classified by the network. The plot of the hidden activations of the

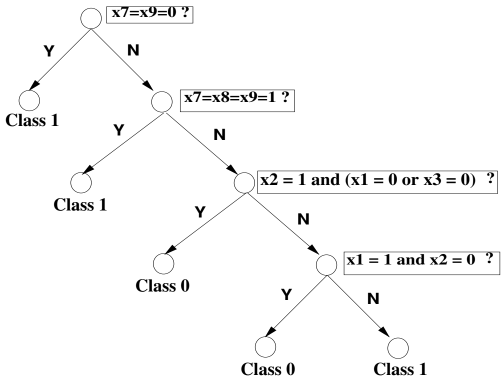

Figure 6: A decision tree for DNF9b dataset with the new features as input.

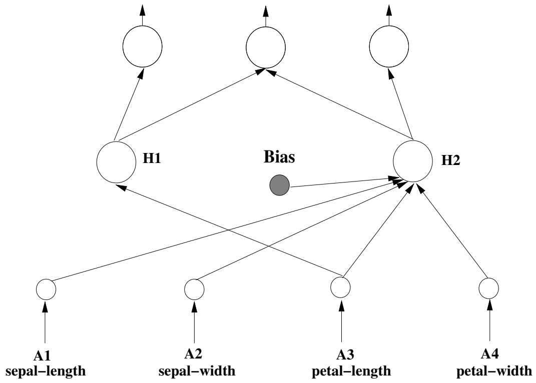

Figure 7: A neural network for Iris classification.

patterns is shown in Figure 8. This plot shows how the original patterns of the same class are grouped together in the 2-dimensional new feature space. Patterns of Iris virginica form a cluster at the upper left hand corner, those of Iris versicolor at the lower left hand corner, while those of Iris setosa have hidden unit 1 activation values between -0.40 and 0 and hidden unit 2 activation of 0 form the third cluster.

When C4.5 is run using the new features, the decision tree shown in Figure 9 is obtained. In the original attribute space, the equivalent conditions for classification of the Irises are as follows:

- If $petal\_length$ > 2.23, then Iris setosa.
- Else if $-$ $sepal\_length$ $-$ 1.57 $sepal\_width$ $+$ 3.57 $petal\_length$ $+$ 3.56 $petal\_width$ > 12.63, then Iris versicolor.
- Else Iris virginica.

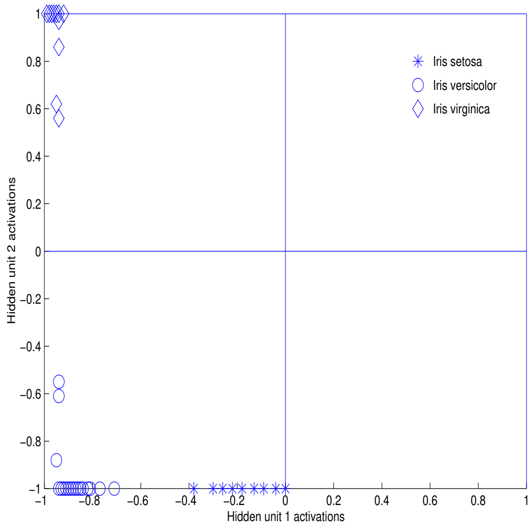

Figure 8: The transformed patterns in 2-dimensional new feature space.

```
A1 > -0.71 : setosa (50.0)
A1 <= -0.71 :
|   A2 <= -0.55 : versicolor (49.0)
|   A2 > -0.55 : virginica (50.0)
```

Figure 9: The classification of Iris patterns in terms using the new features

3. Function 9

The dataset contains both discrete and continuous attributes. All continuous attributes are normalized so that their transformed values fall between 0 and 1. The values of the discrete attribute *Elevel* are binary coded. Since there are 5 possible levels, 4-bit binary strings are used. Values of *Elevel* equal 0,1,2,3 and 4 are coded as $(0,0,0,0), (0,0,0,1), (0,0,1,1), (0,1,1,1)$, and $(1,1,1,1)$ respectively. A neural network trained to achieve 100 % accuracy on the 500 randomly generated training patterns is shown in Figure 10.

In terms of the hidden activation values of the network, a tree with one decision node distinguishes between patterns of Class 1 and those of Class 2. If the activation of a pattern in hidden unit 1 is greater than -0.38, then it is classified as Class 1. Since $\delta^{-1}(-0.38) = -0.40$, the patterns are effectively divided into the 2 classes by the hyperplane $28.55\,Salary + 16.95\,Commission - 1.76\,Elevel1 - 1.34\,Elevel2 - 1.19\,Elevel3 - 1.79\,Elevel4 -$

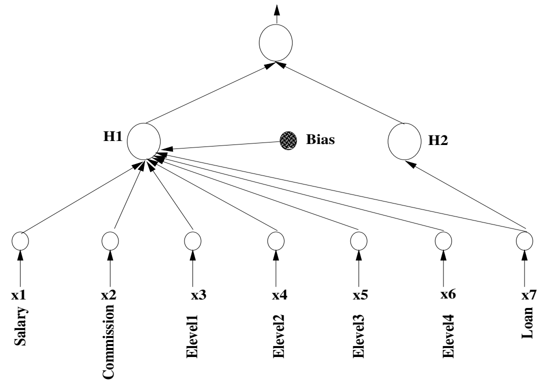

Figure 10: A neural network for Function 9.

31.91 $Loan$ $- 0.40 = 0$, where the coefficients are the weights of the corresponding network connections.

The network was trained using the transformed attributes values, hence it is useful to transform the attributes back to their original intervals. The equivalent hyperplane that splits the original attribute space into 2 classes is given by the equation

$$0.67(Sal\ +\ 1.03\,Com)\ -\ 5370\,El1\ -\ 4080\,El2$$
$$-\ 3640\,El3\ -\ 5450\,El4 -\ 0.2\,Loan - 12240 = 0 \quad (2)$$

By substituting the relevant values for Elevel1-Elevel4, 5 parallel hyperplanes that separate patterns from Class 1 and those from Class 2 can be obtained for the 5 different levels of Elevel:

Elevel 0: $0.67(Sal\ +\ Com)\ -\ 0.2\,Loan - 12240 > 0$

Elevel 1: $0.67(Sal\ +\ Com)\ -\ 0.2\,Loan - 17690 > 0$

Elevel 2: $0.67(Sal\ +\ Com)\ -\ 0.2\,Loan - 21330 > 0$

Elevel 3: $0.67(Sal\ +\ Com)\ -\ 0.2\,Loan - 25410 > 0$

Elevel 4: $0.67(Sal\ +\ Com)\ -\ 0.2\,Loan - 30780 > 0$

In this particular example, the new constructed feature is a linear combination of the original features given by the left hand side of Equation (2). There are 2 possible values of this new feature, they are "greater than 0" or "less than or equal to zero". Decision making in the new feature space becomes very trivial, if the value obtained from an input pattern is "greater than 0" then we classify the pattern as Class 1, otherwise Class 0.

4 Empirical Study and Analysis

Constructing new compound features is useful only if it alleviates the inherent limitation of decision tree methods that test on single features. We carried out experiments using both artificial problems and real-world problems to investigate the effects of having compound features constructed by neural networks. In these experiments, we were concerned with the accuracy and the number of nodes in the trees built using the original and the compound features. A smaller tree implies less data fragmentation than a larger tree. We also expect improvement in accuracy. The results of the experiments are summarized in Table 2. The first 18 problems are artificial problems given in [21]. All of these problems have binary features. The numbers 9 and 12 in problem names indicate the number of features. The remaining seven problems are real-world problems where the data have been obtained from the UC-Irvine machine learning database repository [11].

For each problem, ten-fold cross-validation experiments were conducted. In each of the 10 runs, 80 % of the patterns were used for neural network training, 10 % for cross-validation, and the remaining 10 % for testing. The patterns in the cross-validation set were used to determine when neural network pruning should be terminated. In Table 2 we show the average accuracy rates of the pruned neural networks, the average accuracy and the average size of the C4.5 trees generated using the original features (C4.5[1]) and using the neural network generated features (C4.5 [2]). From the figures in the table we observe the following:

- Accuracy: the accuracy of the tree generated using the new features is similar or significantly better than the accuracy of the tree generated using the original features.

- Tree size: after constructing compound features, the numbers of nodes are significantly reduced. The fragmentation problem is eliminated in trees with three nodes (one root and two leaves).

5 Conclusion

Selective induction algorithms inherit the fragmentation problem which has been illustrated through three data sets with various feature types. Compound features can help eliminate or alleviate the problem. By using a neural network in constructing features, we adopt the principle of global data analysis so as to minimize data fragmentation. Automated feature construction avoids trying a long list of constructive operators and finds compound features for an application by constructing them from a trained neural network. Our empirical study using both artificial and real-world datasets suggests the effectiveness of the proposed method.

Table 2: The accuracy of the neural networks (NN), C4.5 on the original datasets (C4.5[1]), C4.5 on the transformed datasets (C4.5[2]). The average number of nodes in the C4.5 trees are in the last 2 columns.

Problem	Accuracy (%)			Tree size	
	NN	[1]	[2]	[1]	[2]
DNF9a	100.0	100.0	100.0	15.0	5.0
DNF9b	100.0	99.8	99.9	37.0	13.4
DNF12a	100.0	100.0	100.0	31.8	9.0
DNF12b	100.0	100.0	99.9	41.8	21.6
CNF9a	100.0	100.0	100.0	15.0	5.0
CNF9b	100.0	100.0	100.0	17.4	9.0
CNF12a	100.0	100.0	100.0	41.0	7.8
CNF12b	99.2	99.2	99.3	107.4	35.8
MAJ9a	100.0	88.5	100.0	7.0	3.0
MAJ9b	100.0	63.1	100.0	44.4	3.0
MAJ12a	100.0	93.4	100.0	29.0	3.0
MAJ12b	100.0	74.2	100.0	223.4	3.0
PAR9a	100.0	100.0	100.0	15.0	7.0
PAR9b	98.6	90.3	98.2	115.4	11.0
PAR12a	100.0	100.0	100.0	31.0	11.0
PAR12b	99.3	92.3	99.5	484.6	24.2
MUX9	100.0	100.0	100.0	42.6	11.8
MUX12	100.0	100.0	99.9	210.8	22.4
Australian	86.5	84.2	85.7	68.7	5.4
B-Cancer	96.0	95.3	96.1	22.2	3.4
Heart	83.2	72.0	82.8	51.9	3.0
Housing	86.0	82.0	85.8	53.8	9.6
Ionosphere	88.3	90.0	88.1	24.8	3.0
Pima	76.3	70.9	76.4	122.4	3.0
Sonar	86.6	85.5	86.1	31.0	3.0

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Trans. on Knowledge and Data Engineering*, 5(6):914–925, 1993.

[2] E. Bloedorn and R.S. Michalski. Data driven constructive induction in aq17-pre: A method and experiments. In *Proceedings of the Third International Conference on Tools with AI*, 1991.

[3] C.E. Brodley and P.E. Utgoff. Multivariate decision trees. *Machine Learning*, 19:45–77, 1995.

[4] J. Catlett. On changing continuous attributes into ordered discrete attributes. In *European Working Session on Learning*, 1991.

[5] S. Donoho and L. Rendell. Constructive induction using fragmentary knowledge. In L. Saitta, editor, *Proceedings of International Conference on Machine Learning (ICML-96)*, pages 113–121. Morgan Kaufmann Publishers, 1996.

[6] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 7(2):179–188, 1936.

[7] J.H. Friedman, R. Kohavi, and Y. Yun. Lazy decision trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 717–724, 1996.

[8] D. Heath, S. Kasif, and S. Salzberg. Learning oblique decision trees. In *Proceedings of the Thirteenth Internaional Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1002–1007, 1993.

[9] C. Matheus and L. Rendell. Constructive induction on decision trees. In *Proceedings of IJCAI*, pages 645–650, August 1989.

[10] C. Matheus. The need for constructive induction. In L.A. Birnbaum and Collins G.C., editors, *Machine Learning - Proceedings of the Eighth International Workshop*, pages 173–177, June 1991.

[11] C.J. Merz and P.M. Murphy. UCI repository of machine learning databases. Irvine, CA: Uni. of California, Dept. of Information and Computer Science, 1996. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[12] R.S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20(2):111–161, 1983.

[13] S. Murthy, S. Kasif, S. Salzberg, and R. Beigel. Oc1: Randomized induction of oblique decision trees. In *Proceedings of AAAI Conference (AAAI'93)*, pages 322–327. AAAI Press / The MIT Press, 1993.

[14] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.

[15] G. Pagallo. Learning dnf by decision trees. In *Proceedings of IJCAI*, pages 639–644. Morgan Kaufmann, 1989.

[16] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[17] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[18] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group. *Parallel Distributed Processing*, volume 1. Cambridge, Mass. The MIT Press, 1986.

[19] R. Setiono. A penalty-function approach for pruning feedforward neural networks. *Neural Computation*, 9(1):185–204, 1997.

[20] Towell, G.G. & Shavlik, J.W. (1993) Extracting refined rules from knowledge-based neural networks. *Machine Learning*, **13(1)**, 71–101.

[21] R. Vilalta, G. Blix, and L. Rendell. Global data analysis and the fragmentation problem in decision tree induction. In M. van Someren and G. Widmer, editors, *Machine Learning: ECML-97*, pages 312–326. Springer-Verlag, 1997.

[22] J. Wnek and R.S. Michalski. Hypothesis-driven constructive induction in aq17-hci: A method and experiments. *Machine Learning*, 14, 1994.