An Empirical Study of Classifiers for Categorical and Binary Data

Seymour Shlien

July 6, 2017

1 Introduction

Binary and categorical (nominal) variables occur frequently in data mining applications; however, only a few techniques are designed to handle these types of variables. The standard statistical measures such as mean, variance, median, and quantiles are not applicable to this form of data.

An object with categorical or binary attributes assumes non-numerical values. For example, a categorical attribute may be a descriptor, such as colour. A binary feature may indicate the presence or absence of a particular symptom or an exposure to a particular treatment. When one is deciding whether to approve a loan, a binary feature may indicate whether the applicant is employed or has other outstanding loans.

An example of a database consisting of mostly categorical and binary features is the Titanic database, which can be downloaded from the web site [1]. An application that uses binary feaures is market basket analysis using association rules [2, 3, 4].

In machine learning, the main premise is that one starts with a collection of data describing a set of objects that belong to different classes or types. The goal of machine learning is to find a model or classifier that maps the data objects to their correct classes based on these descriptors. The classifier is developed only from the given data. In order to estimate the accuracy of the model, the database is separated into a training set and a test set. The training set is used to develop the model and the test set is used to measure how well the model performs. In this study, all the characteristics describing the objects will be represented by binary variables.

The binary decision tree and the random forest are particularly suited for processing categorical and binary data. The multilayer perceptron (also called the feed forward neural network) can also be used to analyze binary variables. In this note, we will study the behaviour of these classifiers on this data. Synthetic data will be generated using various models.

Working with synthetic data has many benefits for investigating the behaviour of classifiers and learning algorithms. The characteristics of the dataset are known in advance and the dataset's size is unlimited, in contrast to real experimental data. It is easy to determine the effect of varying the size of the training set on the learning algorithm.

Since the training set could be rather small, it may not be very representative of the data. If the classifier encounters some objects that were not present in the training set, it may fail to classify the objects based on the model that it had deduced from the training set.

The misclassified samples or generalization error are considered to result from bias and variance errors, [5],[6] and [7]. The bias error is due to the model being unable to correctly represent the data. The variance error is caused by the presence of noise, which confuses the training process and causes the model to fit the noise in the data (overfitting).

This note deals with perfectly separable classes of data. This implies that a learning algorithm should be able to achieve perfect accuracy given a sufficient number of training samples. In order for this to be true, a particular data vector can assigned to only one class. If the training vector could be assigned to one of several classes with various probabilities, than the classes would not be perfectly separable. Such data is not normally found in real situations, but will be useful for studying the behaviour of classifiers and learning algorithms. For example, if the classifier is unable to attain perfect accuracy on the training data, we know that the classifier has some bias. If the classifier attains perfect accuracy on the training data but makes many errors on the test data, we know that the classifier has poor generalization characteristics.

One of the advantages of working with binary or categorical data is that the feature space is discrete and finite. This allows us to measure the fraction of the input space that is known to the learning algorithm.

The paper will focus on decision trees, decision forest and the multilayer perceptron. All of the experiments described in this note were implemented in the R language [8]. R is a free software environment for statistical analyses and plotting. The software runs under many operating systems and computers. There are over 9000 packages, contributed by many users, that can be imported into R. There are numerous books on R and how it can be applied to machine learning, pattern recognition, and data analysis – for example, [4, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18].

We will limit this investigation to two R packages: RandomForest [19] and RSNNS [20]. The RSNNS package contains a multilayer perceptron neural net model. The RandomForest package supports both binary decision trees and decision forest.

2 Decision Trees and Random Forest

Decision trees and random forest are particularly suited for data vectors containing categorical components. Random forest classifiers were proposed by T.K. Ho [21] and further developed by Leo Breiman [22]. These techniques handle a large variety of data types. They are resistant to overfitting and provide a lot of additional information such as the variable importance. Wouter et. al. [23], Qi [24] and Pauly [25] present reviews of random forest and its applications to bioinformatics and medicine. Caruana et. al. [26, 27] show that random forest is consistently accurate for both low and high dimensional data. A balanced review of this subject can also be found on Wikipedia [28].

There are numerous tutorials on YouTube on how to create and use a binary decision tree or random forest – for example [29, 30, 31, 32]. The R environment comes with many packages such as C5.0 [33], party [34], partykit [35], randomForest [19], and rpart [36] for creating and the decision trees and the random forest.

The basic constituent of the random forest method is the binary decision tree. The binary decision tree is composed of nonterminal and terminal nodes where each of the nonterminal nodes links to two other nodes which may be terminal or nonterminal. These two nodes are called the left and right nodes, and the classifier progresses to either the left or the right, depending on the result of testing a specific feature in the object associated with that node. The progression stops when the predictor reaches a terminal node that contains the predicted class number. Typically the binary decision tree returns a result after testing only a few of the available features in the object. The subset of features that were examined adapts to the feature components of the particular object. In essence, the binary decision tree partitions the training set in a recursive manner into smaller and smaller subsets until they become as 'pure' as possible or, in other words, contain mainly one class of objects. Rokach and Maimon [37] describe many variations of the decision tree and random forest.

Dwyer and Holte [38] show that the algorithm for creating the decision tree is unstable and a minor change in the training set can result in a completely different tree. The randomForest package [19] exploits this behaviour and generates 500 different trees by taking different sets of samples from the training set.

The random forest is a type of ensemble machine learning scheme [39] that uses a large number binary decision trees. Each tree votes for a particular class, and the class with the most votes is chosen [40]. A detailed description on the effects of varying the depth of the trees or the number of trees in the random forest may be found in the Microsoft technical report [41].

The next two sections provide some simple artificial examples that illustrate the operation of the random forest on different types of binary datasets. These examples will be used to provide further details on the operation of binary decision trees and random forest.

3 Nearest Neighbour Mixture Model

In order to group the binary data vectors into clusters we will need to introduce a distance measure. The concept of distance between binary data vectors is not a notion defined in the binary data space, but rather is introduced to create the generation model.

The 'manhattan distance measure' between two vectors is the number of flips one needs in order to go from one binary vector to the other. For example, if one binary vector is represented by 001100 and another by 000111, then the two vectors differ by exactly 3 components, so their manhattan distance is 3 units.

Using this measure, we pick n centroids in the multidimensional binary space and use these centroids to create other data vectors which are close to these centroids. These data vectors will be used to train and test the binary decision tree and random forest classifier.

We would like these centroids to be spread as far apart as possible in the binary input space. This is similar to a problem encountered in error correction coding for which there are a lot of theoretical results [42]. Here, these centroids are found using a random search. The first centroid is picked at random in the binary space, then other centroids are picked sequentially by searching for vectors that are some minimum distance from all the other centroids that were picked. There is no guarantee that the algorithm will succeed in finding all the desired centroids (even if they exist), so the algorithm was designed to give up after a certain number of tries.

Table 1 lists the coordinates of the 4 binary vectors that were used here. It is easy to verify that each of these centroids are exactly 10 units away from each other.

1	1	1	0	0	0	0	1	0	1	0	1	1	1	1	1	1
2	0	0	1	0	1	1	1	1	0	1	1	0	1	0	1	1
3	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0
4	1	1	0	1	1	1	0	1	1	1	0	0	0	1	0	1

Table 1: Center vectors for creating the NN and limited NN data samples.

Training and test samples associated with a centroid were created by flipping the values of up to three components of the centroids. Though the input space is embedded in a data space consisting of 2^{16} distinct vectors, the input space is much smaller. The data vectors in the neighbourhood of up to 3 units around the centroid form a cloud of 695 vectors for the 16-dimensional space. (There are exactly 16 vectors that are exactly one unit distance from the centroid. The number of vectors at a distance d is $16!/(16-d)!*d!$.) Since there are four centroids and four disjoint clouds, the input space consists of 2780

distinct vectors. (This information will be useful when we determine how the accuracy of the classifier varies with the training set size.) Training and test sets were created by sampling these 2780 vectors with replacement.

Table 2 shows the first 12 training samples in the collection that we created. The components of the vectors are correlated with each other, in particular when one looks at the vectors belonging to a particular class. For your information, Leisch [43] describes a different approach for generating binary vectors with correlated components.

Increasing the neighbourhood distance around the centroids from 3 to 4 increases the neighbourhood size from 695 to 2515. If there are 5 centroids, then the number of distinct data vectors increases to 12575. This represents almost 20 per cent of the available input space. Increasing the neighbourhood space further or the number of centroids will increase the number of possible distinct vectors; however, if the neighbourhoods overlap each other the classes will no longer be perfectly separable. Once perfect separation is lost, no classification algorithm can achieve perfect accuracy on either the training set or the test set. For our purposes, we will stay with a neighbourhood of up to 3 units of separation.

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	class
1	0	1	1	0	1	1	1	1	0	0	1	0	1	0	1	0	2
2	1	0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	3
3	1	1	0	0	1	1	1	1	1	1	0	0	0	1	0	1	4
4	0	0	0	1	1	1	0	1	0	1	0	0	0	1	0	1	4
5	1	1	0	0	0	1	1	0	0	0	1	1	1	1	1	1	1
6	0	0	1	0	0	0	1	1	0	1	1	1	1	0	1	1	2
7	1	0	0	0	0	0	0	0	1	0	1	1	1	1	1	0	1
8	0	0	1	1	1	1	1	1	1	1	0	0	1	0	1	1	2
9	0	1	0	1	0	0	1	0	1	0	0	1	1	1	1	1	1
10	1	1	0	0	1	1	0	1	1	1	0	0	0	1	0	1	4
11	0	0	0	0	0	0	1	0	1	0	1	1	1	1	1	0	1
12	0	0	1	0	0	1	1	1	0	1	0	0	1	0	1	0	2

Table 2: First 12 training vectors and their class numbers.

The randomForest package automatically creates a forest of 500 binary decision trees. We will first illustrate the operation of a binary decision tree with the simple tree shown in Table 3. This tree consisting of 25 nodes was generated by restricting the training set to 100 training vectors. There are two types of nodes indicated by the column marked 'status' : the nonterminal node (1) links to either the left or right node, depending on whether the split variable is either 0 or 1. The terminal node (-1) returns the class number prediction. Given a specific data vector, the classifier (also called the predictor) starts with the first node (root node) and tests the value of the 15th binary component in the vector. Suppose that we are classifying the second vector in the training set given in Table 2. The predictor will proceed from node 1 to 2, 4 and finally to node 8, testing variables 15, 14, and 16. Since these test variables are all 0, the predictor always goes to the left node. The terminal node 8 returns the class number 3, which is the correct value.

The randomForest algorithm assigned class 3 to node 8 because all 13 training samples that landed on this node were from class 3. Not all terminal nodes have such a clear split. For example, the training samples that landed on node 9 were split between classes 2 (6), 3 (2), and 4 (3), where the number in parenthesis indicates the number of training samples in each of the classes. If the test sample landed on

node 9, the predictor would assign it to class 2.

Each of the terminal nodes defines a subspace in the data space. For example. terminal node number 8 refers to the data subspace with variables 15, 14, and 16 all set to 0, and the remaining 13 variables can be either 0 or 1. Therefore the subspace consists of 2^{13} or 8192 possible vectors. The entire data space is partitioned into 13 subspaces corresponding to the 13 terminal nodes. The entire set of 100 training samples is distributed among the 13 subspaces.

The randomForest algorithm creates a set of 500 binary decision trees by taking different random samples (with replacement) from the training set. For example, if the training set consists of 1200 vectors, then a random sample consists of 800 of those vectors. The remaining samples, called out-of-bag, are put aside for measuring the accuracy of the decision tree. Furthermore, the randomForest algorithm restricts the selection of the split variables to a different random subset of the 16 variables for each tree generated. As a result, each of the 500 binary decision trees in the forest depends on different features and training samples. Each of the trees will return its own classification with a different confidence level. The predictor will pick the class with the largest number of votes.

node	left node	right node	split var	status	prediction
1	2	3	15	1	0
2	4	5	14	1	0
3	6	7	8	1	0
4	8	9	16	1	0
5	10	11	5	1	0
6	12	13	14	1	0
7	14	15	2	1	0
8	0	0	0	-1	3
9	16	17	4	1	0
10	18	19	3	1	0
11	0	0	0	-1	4
12	0	0	0	-1	2
13	0	0	0	-1	1
14	0	0	0	-1	2
15	20	21	3	1	0
16	0	0	0	-1	2
17	22	23	3	1	0
18	24	25	12	1	0
19	0	0	0	-1	3
20	0	0	0	-1	4
21	0	0	0	-1	2
22	0	0	0	-1	4
23	0	0	0	-1	3
24	0	0	0	-1	4
25	0	0	0	-1	1

Table 3: Computer representation of a decision tree created from 100 training samples. The node number is given in the first column, the status column indicates whether this node is a nonterminal node (1) or a terminal node (-1). Assuming the node to be nonterminal, the fourth column is the variable number used to determine whether to proceed to the left or right node.

As an example, Table 4 lists the subspaces that were used to classify training vector 2 using the first 10 trees in the forest. The predictor chose class 3 for that vector because that was the class number returned by the majority of the 10 trees.

tree no.	features	class counts
1	15,14,16	0 ,0 ,13 ,0
2	5,3,10,14,15	0 ,0 ,3 ,0
3	14,13,5	0 ,0 ,11 ,0
4	11,8,15,16	0 ,0 ,12 ,0
5	15,7,16,2	0 ,0 ,11 ,1
6	14,11,6,2,15	0 ,1 ,3 ,0
7	15,8,11	1 ,0 ,14 ,2
8	13,6,8,5	1 ,0 ,1 ,0
9	6,1,5,14	0 ,0 ,3 ,0
10	8,15,9,5,4	1 ,0 ,2 ,0

Table 4: Binary variables used to classify training vector 2 for the first 10 trees produced by randomForest. The last column indicates the distribution among the 4 classes of training samples reaching that terminal node.

The RandomForest package has numerous input parameters that control how the decision trees are generated, their maximum depth, and the number of trees. It is beyond the scope of this study to look for the best set of parameters. Nevertheless, here are a few interesting experiments that were performed on this data.

Figure 1 shows how the average number of terminal nodes in a tree varies with training sample size. The number of nonterminal nodes in the tree is approximately the same, so the total number of nodes in the tree is about double the plotted values. Each node requires 6 fields to store pointers to the left and right nodes, the splitting feature number, the splitting threshold value, the node type, and the prediction. Therefore a tree with 100 nodes requires 600 units of data. A forest of 500 trees consumes about 250K of computer memory; however, this is not an issue for modern computers.

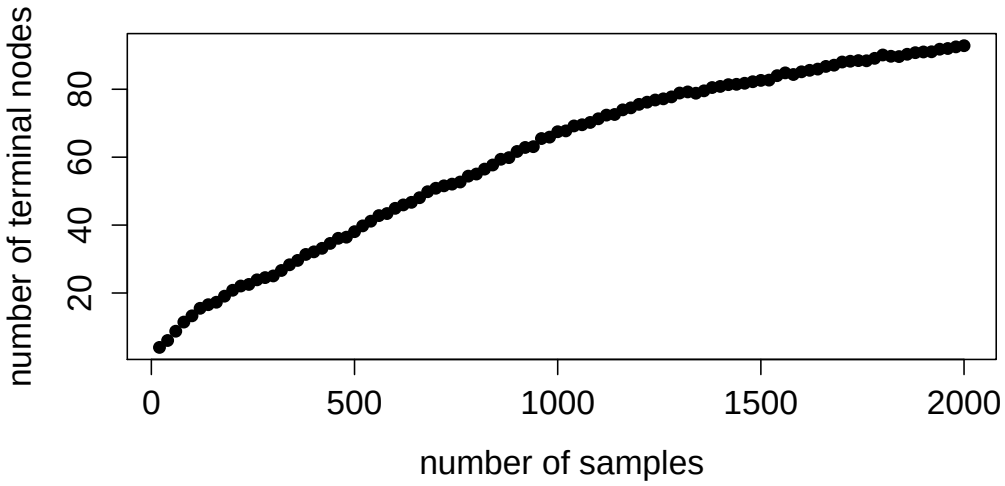

Figure 1: Average number of terminal nodes per tree in the random forest as a function of the training set size.

One way of evaluating the effectiveness of the classifier is measuring how well it handles data that was not present in the training data. It has been shown that there are 2780 distinct data vectors in the input data space. Using the R function called unique(), we can determine the number of distinct vectors that were found in the training data. The number of missing vectors in the training set is merely the number of vectors needed to add up to 2780.

If the training set contains less than 2780 data vectors, there will certainly be many data vectors missing from the space. Even if the training set contained 3000 or more vectors, some of these vectors may have occurred more than once, while others are missing. Assuming that each of the distinct 2780 vectors has an equal probability of occurring, the expected number of distinct vectors that would be found in a training set consisting of n binary vectors can be predicted. The probability that one of the n training vectors does not fall in a particular cell is given by the following.

$$P(n) = (1 - 1/2780)^n \tag{1}$$

Therefore the probability of that cell being occupied by one or more random training vectors is $1 - P(n)$. The experimental curve shown in Figure 2 follows follows this prediction very well.

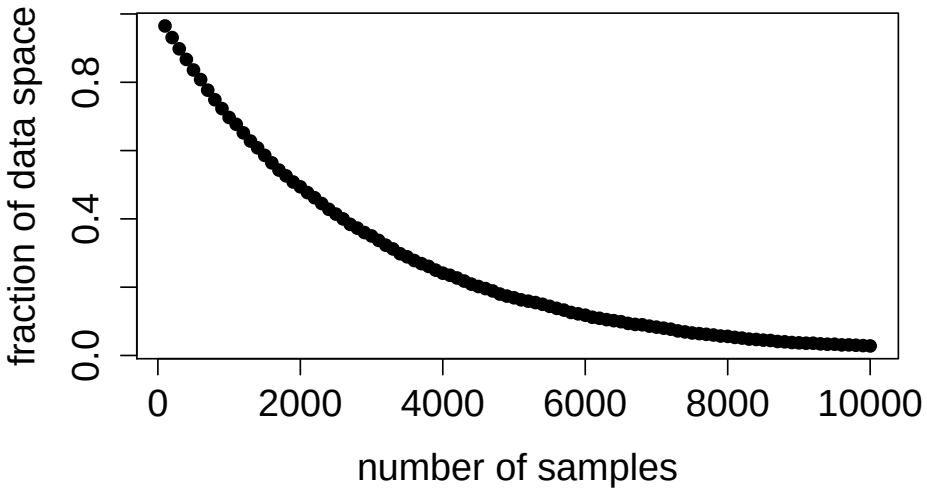

Figure 2: Fraction of data space consisting of 2780 unique vectors that are **not** covered by a random subset of training samples versus the size of training subset.

The accuracy of the binary decision tree depends on how well it can predict the class numbers of the data vectors that were missing in the training set. This depends upon the complexity of the model generating the data. We generated a series of binary decision trees by running the randomForest function repeatedly for different sizes of training sets and measured their accuracies on an independent test set. Figure 3 shows the results of these measurements versus the training set size for the single binary decision tree on the left and the random forest on the right. The blue points shows the fraction of distinct vectors in the input space that were **not** represented in the training set. The proportion of the vectors in the test set and the proportion of the vectors in the training set that were misclassified are plotted in black and red respectively. For example, when the training set consists of only 200 vectors, less than 200 of the distinct vectors or about 7 per cent of the input space are present. The remaining 93 per cent are missing

from the training set. One can expect that the test set will contain mainly of data vectors that did not exist in the training set. The fact that the binary decision tree predictor is still correct more than 80 per cent of the time on the test set indicates that the predictor has good generalization characteristics. The results are even better for a random forest consisting of 500 trees. The training error rate for the random forest was consistently zero for all training set sizes.

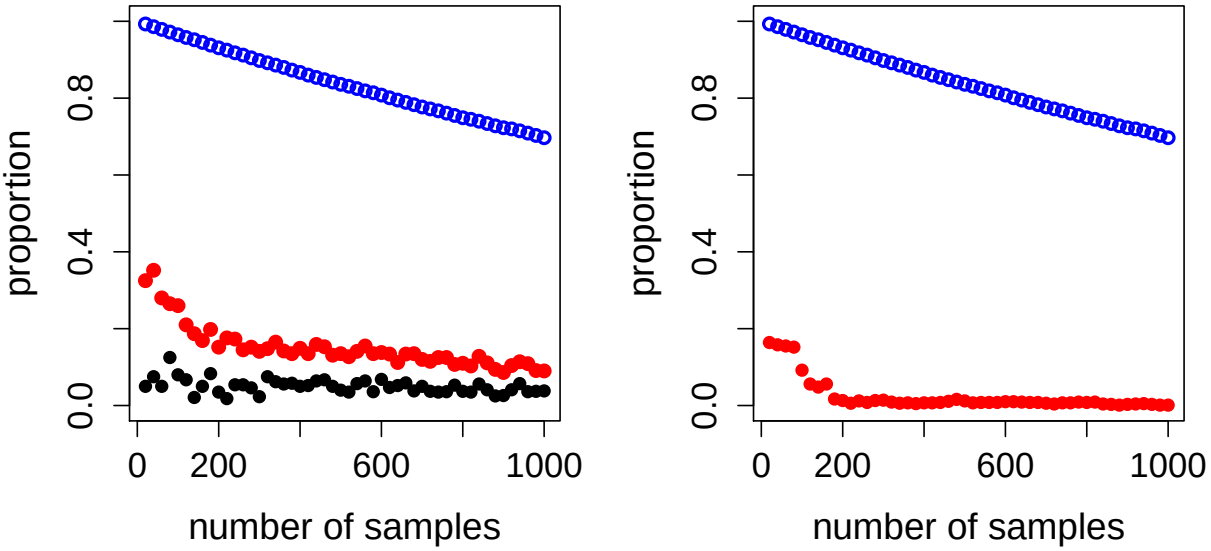

Figure 3: Left: results for a single binary decision tree. Right: results for a random forest. Fraction of test set (red) and fraction of training set (black) that were misclassified as a function of training set size. Blue curve: the fraction of all possible distinct data vectors that were not included in the training set. The samples were 16-dimensional binary vectors that were perturbed from the 4 centroids by up to 3 units.

This section showed how the random forest algorithm performs on perfectly separable classes where the complexity of the data is fairly low. In the next section, we will show how the random forest algorithm behaves when the data is complex.

4 Complex Binary Data

The sample complexity of a machine learning algorithm is defined as the number of training samples needed to classify the test data [5, 44]. As shown in the previous section, the last example did not pose a difficult problem for the random forest classifier. The data was well organized and filled a small fraction of the binary input space. In contrast, the data presented here will be irregular and will fill the entire input space. Though we cannot visualize the binary space, Figure 4 presents a 2-dimensional analogy of the type that we will be working with.

In order to limit computer processing time, the dimension of the binary space was reduced from 16 to 13 or 11. This will limit the size of the input space to either 8192 or 2048 distinct binary vectors.

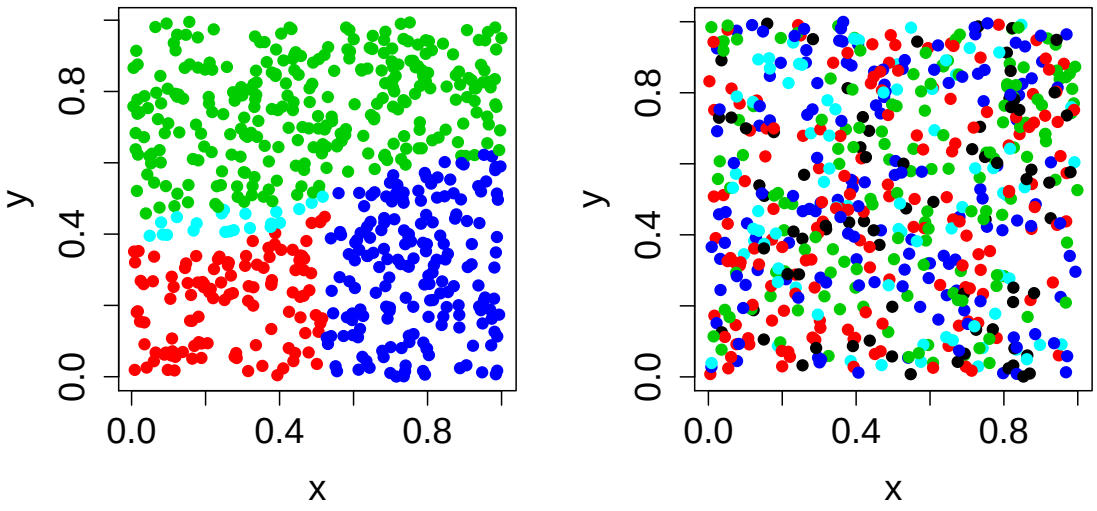

Figure 4: Left: the data is quite regular. Right: the data is very irregular.

The input space consists of all the possible binary vectors in the 13- or 11-dimensional space. These vectors are classified into 5 classes using the following rule: let n be the number of components in the vector that are set to one. The class number of the vector is then determined by the remainder when n is divided by 5. In order to make the class numbers between 1 and 5, 1 is added to the remainder. For example, for the 13-dimensional binary vector given here,

```
x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13
1  1  0  0  0  0  1  0  1   0   1   1   1
```

there are 7 bits set to 1, and the remainder when divided by 5 is 2, so the vector is assigned to class 3. This rule leads to a complicated mapping of the data space into the 5 classes.

Training data sets of various sizes were created using a random generator and applying the above rule to classify the samples. The test set consisting of 1000 samples was created in a similar fashion. The randomForest algorithm was trained on the different training sets and its accuracy was measured as a function of the training set size. Figure 5 shows how the test accuracy varies with the number of samples in the training set for the 11- and 13-dimensional cases. The training set error rate for the random forest was again consistently zero.

All the training samples were classified correctly; however, the fraction of test samples that were misclassified by the random forest predictor is much higher. In fact, the proportion of misclassified vectors in the test set follows the proportion of training vectors that were missing relative to the input space. Note that the classifier does not perform much better than the random classifier, which should be correct 20 per cent of the time for this 5-class problem. (This explains the small separation of the blue and red curves when the training sample size is small.)

The test set misclassification rate eventually approaches zero when the training set is sufficiently large to include all the distinct vectors in the input space. The random forest classifier is still effective, but its generalization capability is low. It does not infer the correct classification of a vector that was missed during the training phase. This pattern is even more evident in the 13-dimensional input space.

Nevertheless, the fact that the training set error rate was consistently zero implies that the random forest is still good at memorizing the mapping of the data vectors that were present in the training set.

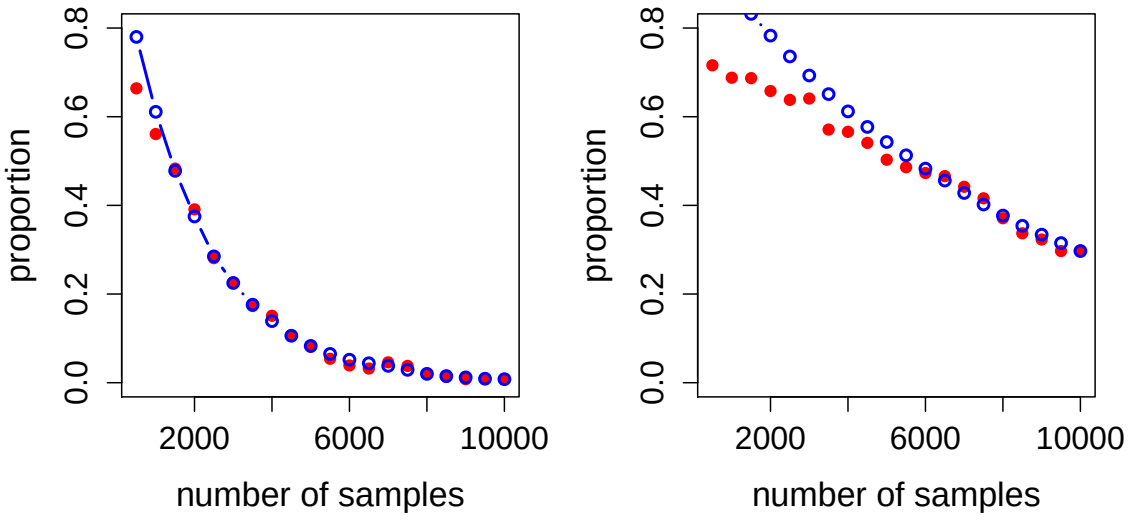

Figure 5: Red points: fraction of test set consisting of 1000 samples that were misclassified by the randomForest predictor Blue curve: the fraction of all possible distinct data vectors that were not included in the training set. Left: 11-dimensional binary vectors. Right: 13-dimensional binary vectors.

These results do not exclude the fact that there may exist another type of classifier that may be able to extract the patterns in the data. In fact, better results were obtained when the multilayer perceptron (mlp) was used to classify the data. As seen in Figure 6, the test set error rate indicated that the classifier was capable of predicting the correct class of the missing data vectors which formed the majority of the test set. The training set error rate shown in black was no longer zero. Though the complexity of the data exceeded the capabilities of the random forest algorithm, the mlp algorithm managed to detect some pattern in the data, which it used to classify the vectors in the input space that were missing from the training set.

In this example, we used a single hidden layer with 30 nodes; however, subsequent trials indicated that as long as there were at least 15 hidden nodes the results were more or less the same. The mlp did not achieve a zero error rate even for the large training sets. The jitter in the graph of Figure 6 is likely due to the failure of the backpropagation algorithm to find the absolute minimum. Each time the backpropagation is started, it is initialized with different weighting coefficients. Therefore there is a certain amount of randomness in these results. In this study the backpropagation was executed for 1500 iterations, and in most instances it had reached convergence.

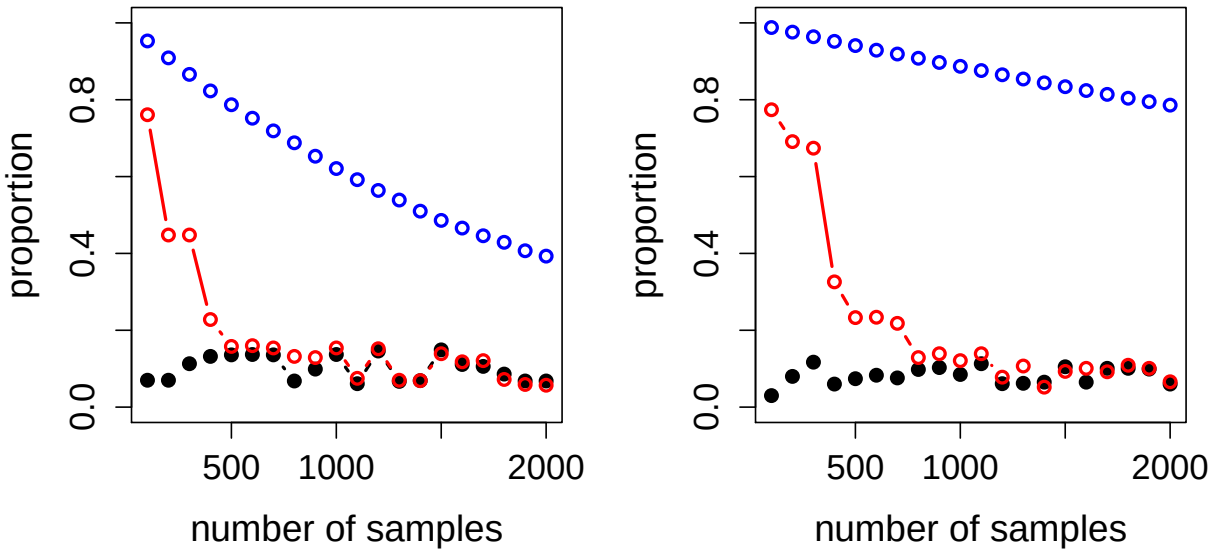

Figure 6: Black and red points: fraction of vectors in the training set and the test set that were misclassified by the RSNNS mlp predictor as a function of training set size. Blue curve: the fraction of all possible distinct data vectors that were not included in the training set. Left: 11-dimensional binary vectors. Right: 13-dimensional binary vectors.

Unlike the random forest, the mlp algorithm requires a certain amount of experimentation to find the ideal architecture. The mlp is computationally more demanding owing to the large number of iterations required by the backpropagation algorithm.

There are many other approaches that may yield better results. For example, an ensemble of neural networks ([45, 46]) may perform better on this dataset. Numerous variations of the decision tree and random forest are described in the literature [37].

As a final experiment, a new dataset that was completely devoid of any structure was created. Random class numbers were assigned to the vectors in the input space using the following scheme. All the distinct vectors in the input space were enumerated, stored in a table, and assigned a random class number between 1 and 5. For the 11- and 13-dimensional spaces, this required lists of 2048 and 8192 vectors. The training and test sets were then built by taking random samples (with replacement) from this list.

The results shown in Figure 7 for the mlp classifier indicate that it did not succeed with this data. Normally, the training error should decrease as more training samples are included; however, the graph indicates the opposite. When the number of training samples is small, the 300 or so weighting factors are able to limit the classification error rate. As the number of training samples increases, the true complexity of the data is revealed and this exceeds the capabilities of the mlp classifier. The test set accuracy was no better than a random classifier. In contrast, the randomForest still managed to memorize the classifications of the vectors as indicated by Figure 8.

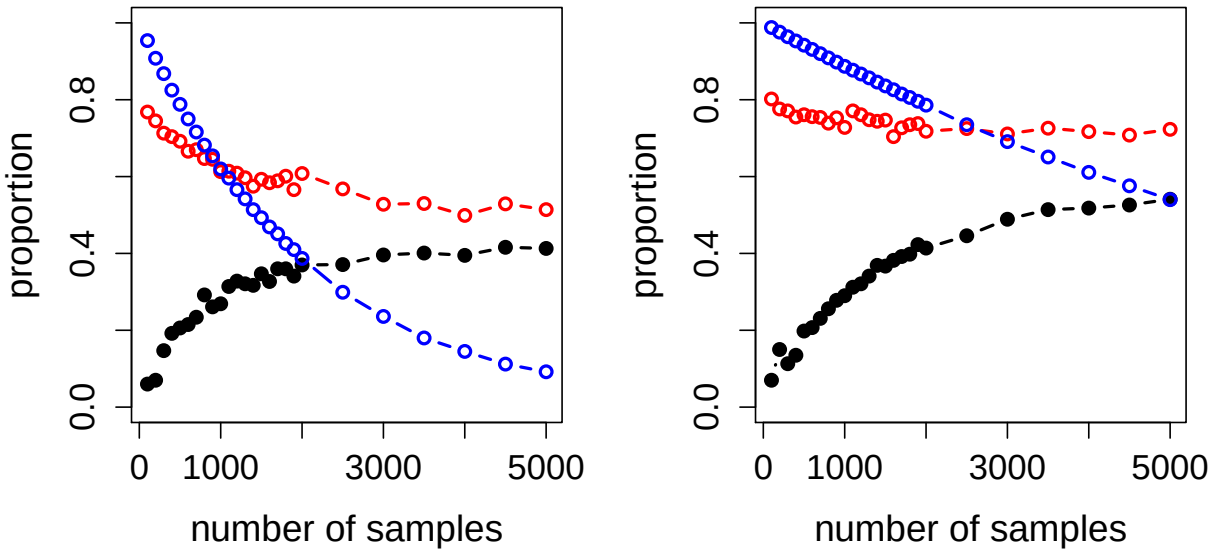

Figure 7: Black and red points: fraction of vectors in the training set and the test set that were misclassified by the RSNNS mlp predictor as a function of training set size. Blue curve: the fraction of all possible distinct data vectors that were not included in the training set as a function of the number of samples in the training set. Left: 11-dimensional binary vectors. Right: 13-dimensional binary vectors. Class numbers were chosen at random for the particular vectors.

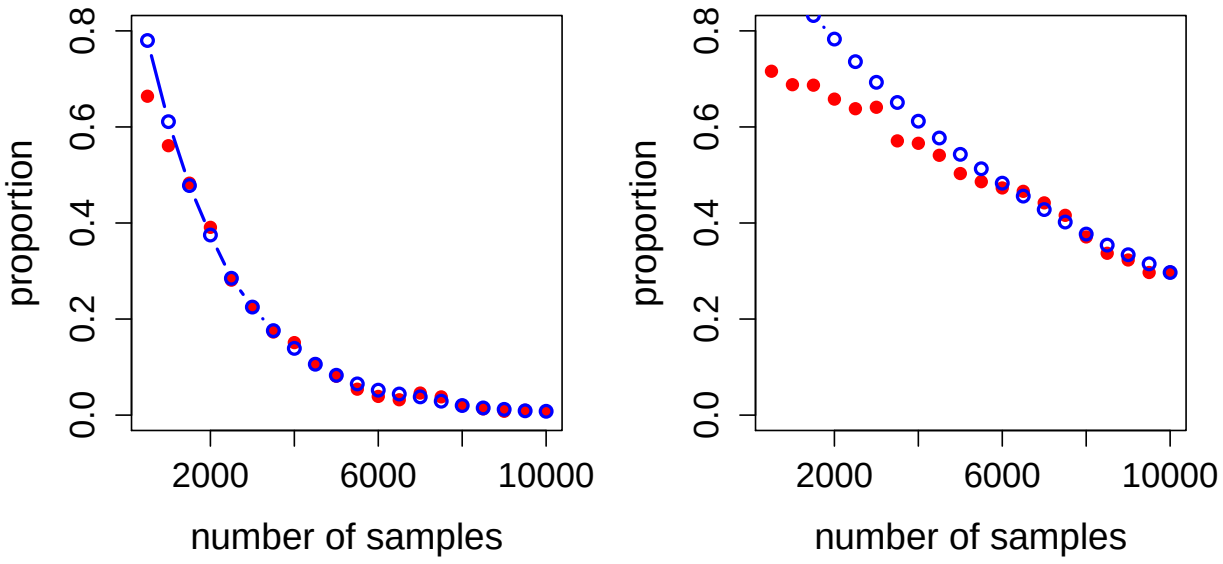

Figure 8: Red points: fraction of vectors in the test set that were misclassified by the randomForest predictor as a function of training set size. Blue curve: the fraction of all possible distinct data vectors that were not included in the training set as a function of the number of samples in the training set. Left: 11-dimensional binary vectors. Right: 13-dimensional binary vectors. Class numbers were chosen at random for the distinct vectors.

5 Discussion

This paper investigated the performance of the random forest classifier on synthetic binary data. The data was generated in a manner that ensured that the data can be classified correctly with perfect accuracy given the right predictor. The purpose of these experiments was to gain experience using the random forest classifier when applied to binary data.

In the last few years, large collections of data for analysis have become available. Complex relationships may exist among the components of the data, and discovering these relationships is a difficult problem.

The neural network and random forest have become standard methods in data mining. In particular, the random forest can handle a large variety of data and can provide insight on the importance of the different variables in the data.

A study like this does not have an end. There are many other methods for extending ensemble classifiers, binary decision trees and random forest. The R source code of this study is being made available so that other researchers can replicate or extend these results [47].

This paper has gone through several reorganizations. The appendices cover other topics that did not fit exactly within the theme of this paper.

References

[1] M. Lichman. UCI machine learning repository, 2013. http://archive.ics.uci.edu/ml.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, 2010.

[3] Jure Leskovec, Anand Rajaraman, and Jeff Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014.

[4] Brett Lantz. *Machine Learning with R, Second Edition*. Packt Publishing, Birmingham - Mumbai, 2015.

[5] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AML Book, 2012.

[6] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[7] Gilles Louppe. *Understanding Random Forests from Theory to Practice*. PhD dissertation, 2014.

[8] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. http://www.R-project.org/.

[9] Chiu Yu-Wei. *Machine Learning with R Cookbook*. Packt Publishing, Birmingham - Mumbai, 2015.

[10] Dr. Mark Gardener. *Beginning R, the Statistical Programming Language*. John Wiley & Sons, Inc, Indianapolis, 2012.

[11] Johannes Ledolter. *Data Mining and Business Analytics with R*. John Wiley & Sons, Inc, Harboken, NJ 07030, 2013.

[12] Ozgur Ergul. *Guide to Programming and Algorithms Using R*. Springer, London, 2013.

[13] Nina Zumel and John Mount. *Practical Data Science with R*. MANNING, Shelter Island, N.Y., 11964, 2014.

[14] Paul Teetor. *R Cookbook*. O'Reilly Media, Inc., Sebastopol, CA 95472, 2011.

[15] Dan Toomey. *R for Data Science*. Packt Publishing, Birmingham - Mumbai, 2014.

[16] Winston Chang. *R Graphics Cookbook*. O'Reilly Media, Inc., Sebastopol, CA 95472, 2012.

[17] Joseph Adler. *R in a Nutshell, 2nd Edition*. O'Reilly Media, Inc., Sebastopol, CA 95472, 2012.

[18] Robert I. Kabacoff. *R in Action, 2nd Edition*. MANNING, Shelter Island, N.Y., 11964, 2015.

[19] Leo Breiman, Adele Cutler, Andy Liaw, and Matthew Wiener. *Package 'randomForest'*, 2015. https://cran.r-project.org/web/packages/randomForest/randomForest.pdf.

[20] Christoph Bergmeir and Jose M. Benitez. *Package 'RSNNS'*, 2016. https://cran.r-project.org/web/packages/RSNNS/RSNNS.pdf.

[21] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8), 1998.

[22] Leo Breiman and Adele Cutler. *Random Forests*. https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm.

[23] Wouter G. Touw, Jumamurat R. Bayjanov, Lex Overmars, Lennart Backus, Joe Boekhorst, Michiel Wels, and Sacha A.F.T. van Hijum. *Data Mining in the Life Sciences with Random Forest: a walk in the park or lost in the jungle?* https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3659301/.

[24] Y Qi. Random forest for bioinformatics. In *Ensemble Machine Learning Methods and Applications*, 2012.

[25] Olivier Pauly. *Random Forests for Medical Applications*. Phd Thesis, 2012.

[26] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of unsupervised learning algorithms. In *ICML '06 Proceedings of the 23rd international conference on Machine Learning*, pages 161–168, 2006.

[27] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *ICML '08 Proceeding of the 25th international conference on Machine learning*, pages 96–103, 2008.

[28] *Random forest*. https://en.wikipedia.org/wiki/Random_forest.

[29] Derek Kane. *Data Science - Part V - Decision Trees and Random Forests*. https://www.youtube.com/watch?v=OByOgGXq76A.

[30] Nando de Freitas. *Machine learning - Decision trees*. https://www.youtube.com/watch?v=-dCtJjlEEgM.

[31] P. Dasgupta. *Lecture -26 Learning: Decision Trees*. https://www.youtube.com/watch?v=pMHOPezBUfU.

[32] Thales Sehn Korting. *How Random Forest algorithm works*. https://www.youtube.com/watch?v=loNcrMjYh64.

[33] Max Kuhn, Steve Weston, Nathan Coulter, Mark Culp. C, and R. Quinlan. *Package 'C50'*, 2015. https://cran.r-project.org/web/packages/C50/C50.pdf.

[34] Torsten Hothorn, Kurt Hornik, Carolin Strobl, and Achim Zeileis. *Package 'party'*, 2017. https://cran.r-project.org/web/packages/C50/C50.pdf.

[35] Torsten Hothorn and Achim Zeileis. *Package 'partykit'*, 2016. https://cran.r-project.org/web/packages/partykit/partykit.pdf.

[36] Terry Therneau, Beth Atkinson, and Brian Ripley. *Package 'rpart'*, 2015. https://cran.r-project.org/web/packages/rpart/rpart.pdf.

[37] Lior Rokach and Oded Maimon. *Data Mining with Decision Trees Theory and Applications*. World Scientific, second edition edition, 2015.

[38] Kenneth Dwyer and Robert Holte. Decision tree instability and active learning. In *ECML '07 Proceedings of the 18th European conference on Machine Learning*, pages 128–139. Springer-Verlag Berlin, Heidelberg, 2007.

[39] Robi Polikar. Ensemble learning. In Cha Zhang and Yunqian Ma, editors, *Ensemble Machine Learning Methods and Applications*. Springer Science+Business Media, 2012.

[40] Micheline Kamber Jiwaei Han and Jian Pei. *Data Mining Concepts and Techniques 3rd Edition*. Morgan Kaufmann Publishers, 225 Wyman Street, Waltham, MA 02451, U.S.A., 2012.

[41] A. Criminisi, J.Shotton, and E. Konukoglu. *Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning*. https://www.microsoft.com/en-us/research/publication/decision-forests-for-classification-regression-density-estimation-manifold-learning-and-semi

[42] William Carey Huffman. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003.

[43] Friedrich Leisch, Andreas Weingessel, and Kurt Hornik. *On the Generation of Correlated Artificial Binary Data*. https://www.researchgate.net/publication/2600074_On_the_Generation_of_Correlated_Artificial_Binary_Data.

[44] *Sample Complexity*. https://en.wikipedia.org/wiki/Sample_complexity.

[45] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[46] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1-2):239–263, 2002.

[47] Seymour Shlien. *Binary models*. http://ifdo.ca/~seymour/R/binary_examples.zip.

6 Appendix A: Illustration of the Random Forest on Numeric Data

As an example, Figure 9 illustrates how the binary decision tree partitions data in a 2-dimensional rectangular space. A multilayer perceptron (mlp) with random weighting coefficients was used to classify random vectors inside the rectangle bounded by 0 and 1. (Details on the design of the mlp are given in the next section.) The top left corner of the figure shows how the mlp classified the data. A binary decision tree was created in stages up to a certain number of nodes. The manner in which the tree classifies the data is shown in each of the panels going left to right as the tree expands. Since the tree looks at only one feature at a time, the decision boundaries are always vertical or horizontal.

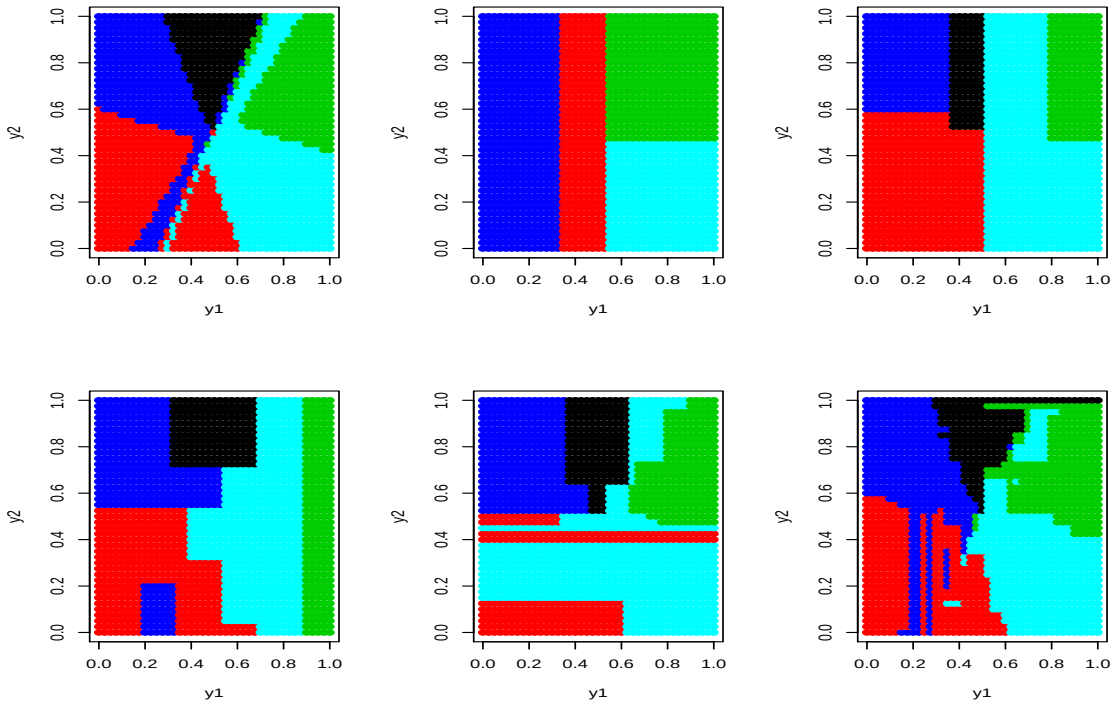

Figure 9: Top corner, training data. Remaining plots are the predicted classification of the test data as a function of the number of nodes in the decision tree.

The mappings of each of the 4 trees generated by the randomForest package are shown in Figure 10

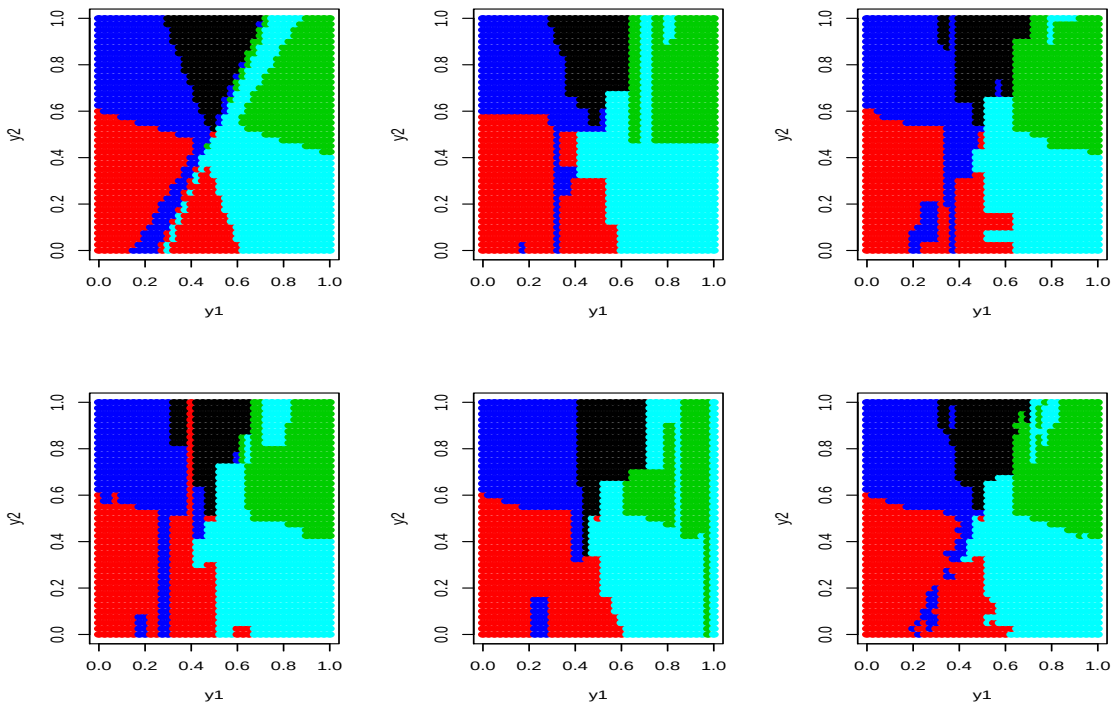

Figure 10: Top left, training data. The next 4 plots are the mapping of the classifications of the data using the 4 individual trees produced by randomForest. The bottom right shows the mapping function produced by the combination of the 4 trees. The individual trees contained a maximum of 63 nodes determined from 1681 training samples. The training samples were produced by a random multilayer perceptron.

7 Appendix B: Multilayer Perceptron Model

The multilayer perceptron (mlp) is known for its ability to approximate a large range of interesting functions. By choosing random weighting values we can create various interesting mapping functions to our synthesized data. In this paper, we do not apply the random mlp to map the binary data vectors into classes; however, in the next section, we use it to demonstrate how the binary decision tree and random decision forest approximate a 2-dimensional numeric data.

It is fairly easy to implement the mlp using plain matrix operations. For illustration, assume a 3-layer neural network where each layer has exactly 2 nodes. Thus there is one hidden layer with 2 nodes and the input and output also have 2 nodes each. For the above model the weighting matrix resembles the following:

$$\mathbf{W} = \begin{bmatrix} 0 & 0 & w & w & 0 & 0 \\ 0 & 0 & w & w & 0 & 0 \\ 0 & 0 & 0 & 0 & w & w \\ 0 & 0 & 0 & 0 & w & w \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

where w are the weighting factors. We shall demonstrate the predictor for a single input vector consisting

of 2 components. In order to apply the weighting matrix to the vector, it is necessary to extend it to 6 components by adding zeros. Suppose the input data is (x_1, x_2), then we create the vector $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0})$. To simulate the neural network, we compute

$$h = x * W \tag{3}$$

where * denotes matrix multiplication, then add the bias vector and apply the activation function to all the nonzero components of y. (The activation function is a nonlinear function which maps real numbers to a range between -1.0 and 1.0.) This returns a new vector containing the activations in the hidden layer which looks like $h = (0, 0, h_1, h_1, 0, 0)$. Repeating this process on h, the output values

$$y = h * W \tag{4}$$

appear at the tail end of the vector $y = (0, 0, 0, 0, y_1, y_2)$.

For purposes of demonstration, we use a neural network with 2 hidden layers where the first layer contains 4 nodes and the second layer contains 6 nodes (see Figure 11. There are 5 output nodes corresponding to 5 possible classes and the input data is numeric and limited to two dimensions. Given a random data vector containing two components, the mlp computes the activations of each of the 5 output nodes. The data vector is assigned to the class associated with the output node with the highest activation.

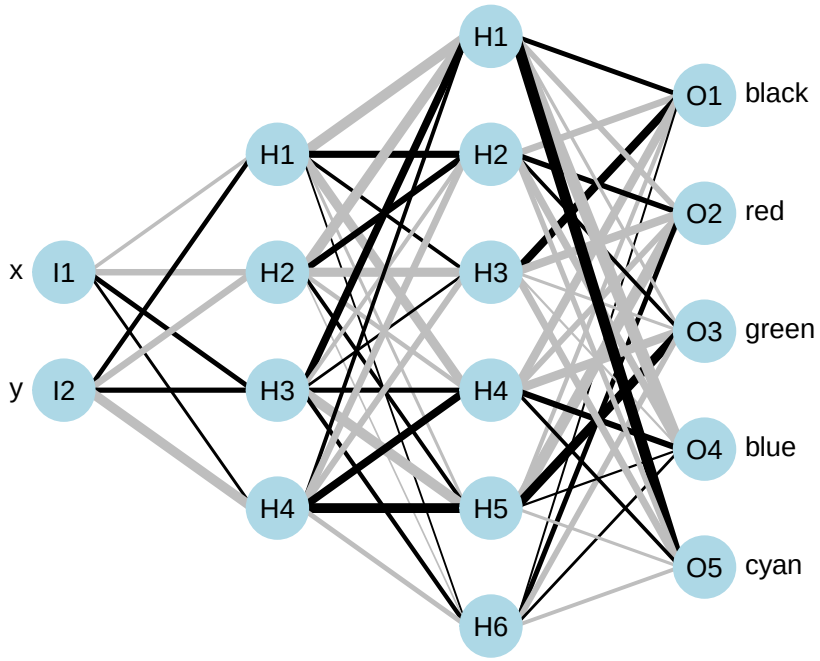

Figure 11: Architecture of a sample multilinear perceptron used in this study.

To create a random mlp, we need a random weighting matrix and a suitable bias vector. The random weighting coefficients were picked from a normal distribution with zero mean and a standard deviation of 20.0. The node bias was chosen so that the average activation for the training data at the node was zero. (A poor choice for the bias would result in all the training samples being mapped to a single class, producing a rather uninteresting distribution.) Figure 12 shows some sample random mapping functions that were produced in this manner.

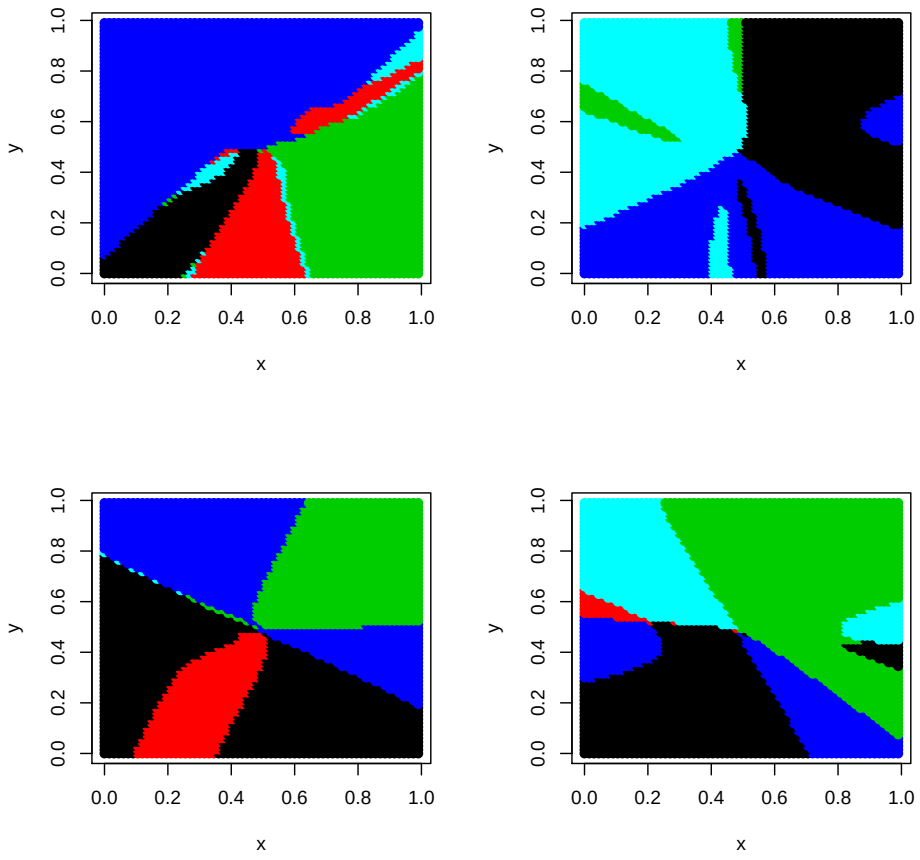

Figure 12: Sample mapping functions produced by the random mlp.

It is not necessary to always assign the class to the output node with the highest activation. For example, one gets interesting results by assigning the class numbers on the basis of the node with the second (or third) highest activation. For example, see Figure 13.

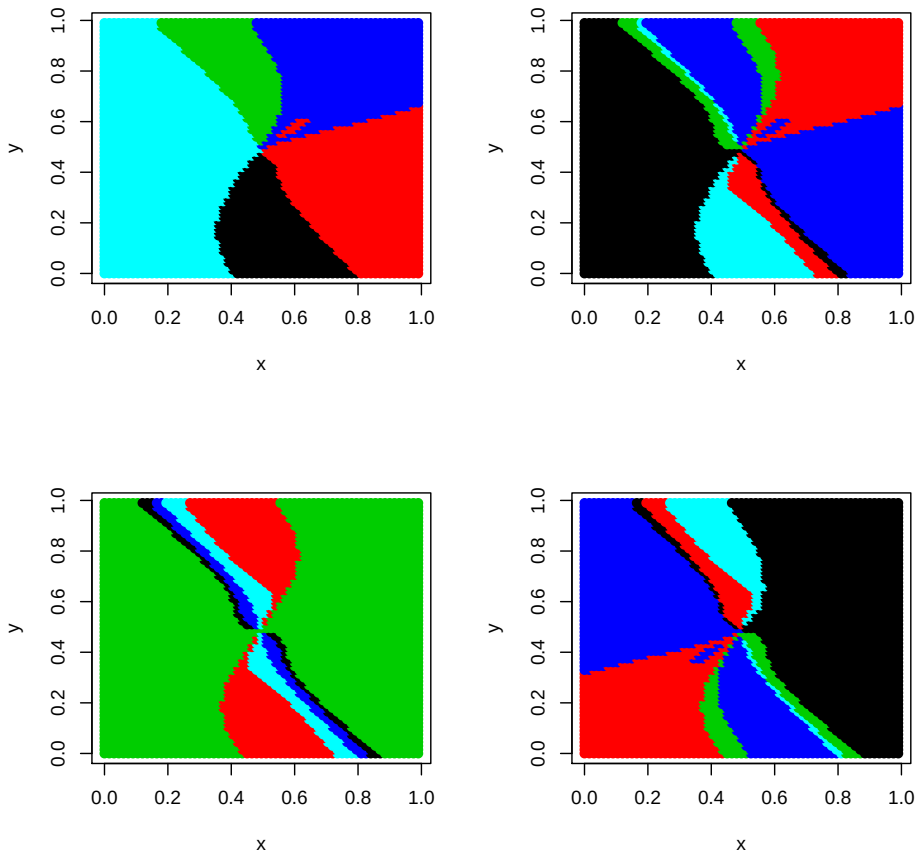

Figure 13: Sample mapping functions produced by using the node with the highest activation (top left), second highest (top right), third highest (bottom left) and fourth highest activation (bottom right).

Real-life data rarely fills the entire dimensional space. In order to limit the distribution of the synthesized training data, one can set up certain classes called (not available) and any training samples that would be mapped to NA would be suppressed.

The random multilayer perceptron is a useful technique for creating new test models. Higher dimensional spaces can be formed by increasing the number of input nodes. The input nodes can either be binary or numeric features. The output of the mlp can be used to create additional features – either numeric or ordinal.

8 Appendix C: Complexity in 2 Dimensions

The effect of complexity on the classifier can be investigated in 2 dimensional space. A two-dimensional grid consisting of 2591 points was created for a square that is 1 by 1. The points on the grid were assigned to 4 classes using the following procedure. All the points were initially assigned to class 2. n smaller squares were randomly placed over the 1 by 1 square and were used to define various regions in the space which were assigned different class numbers. To make things simple, we allowed the squares to overlap. These 2591 points defined the parent distribution from which training and test samples were derived. Training and test samples were created by randomly sampling with replacement the parent distribution.

Figure 14 shows the training sets of various sizes for a parent distribution created using 20 squares.

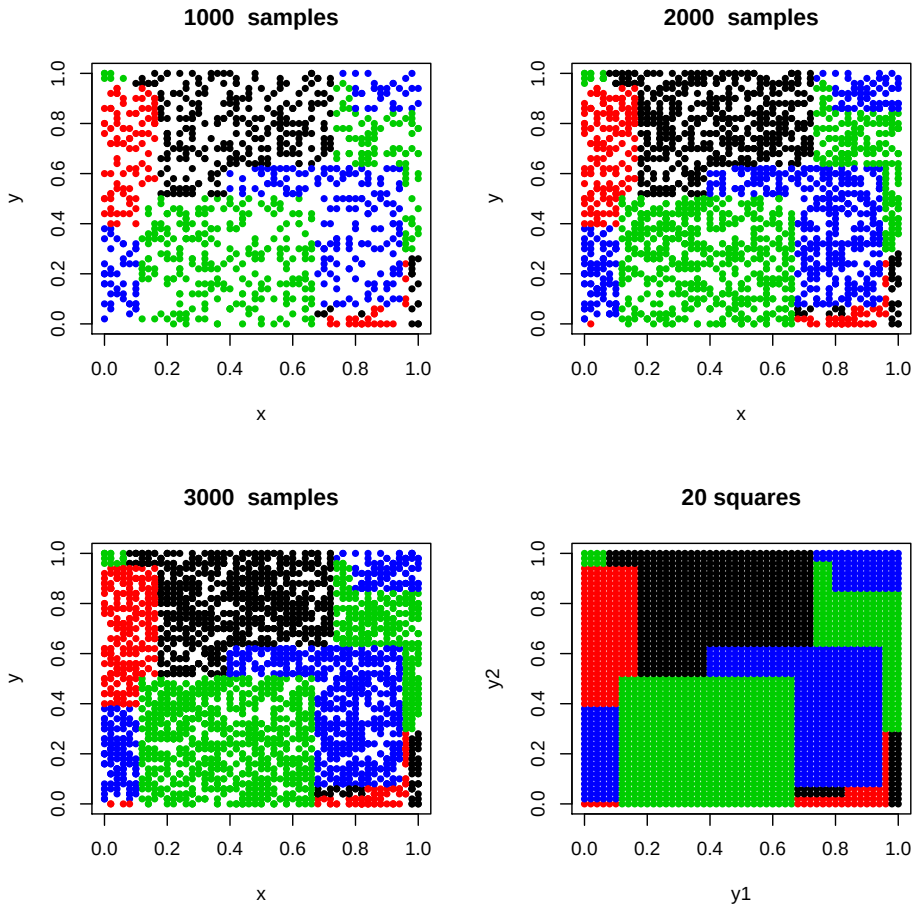

Figure 14: Two-dimensional training samples extracted from a two-dimensional grid shown in the bottom right.

A more complex case was created by overlaying as many as 200 squares is shown in Figure 15. We allowed the size of the squares to gradually shrink to zero.

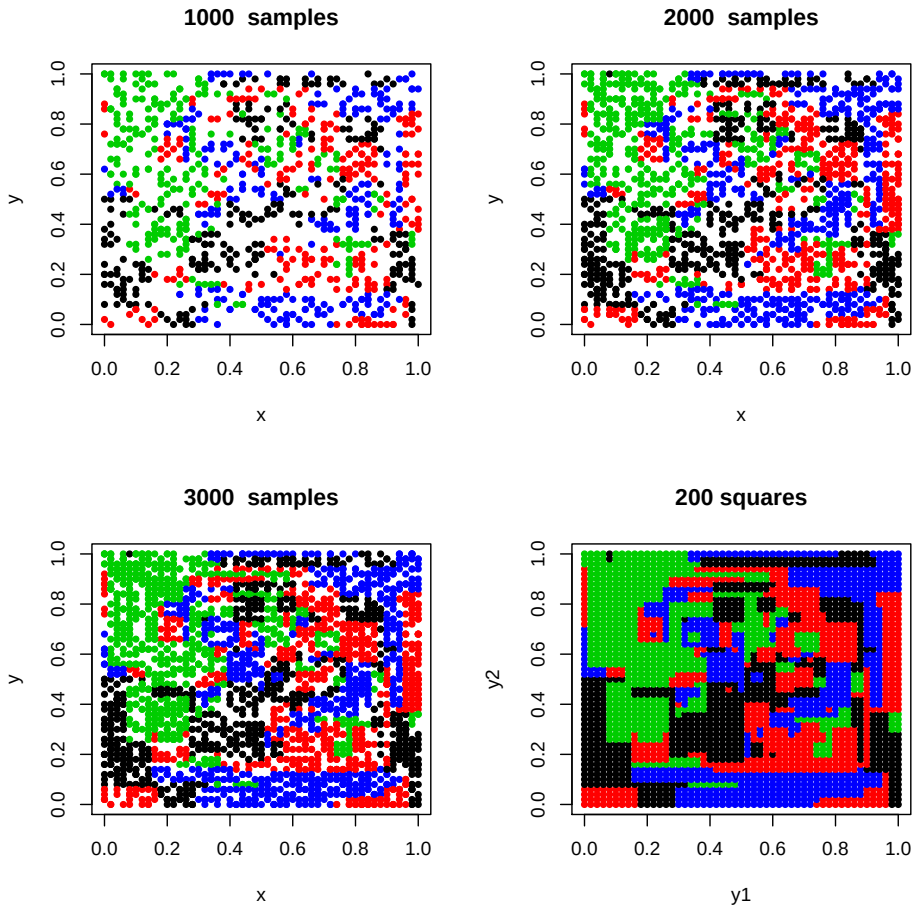

Figure 15: Two-dimensional training samples extracted from a two-dimensional grid shown in the bottom right.

The performance of the randomForest package on this data is shown in Figure 16.

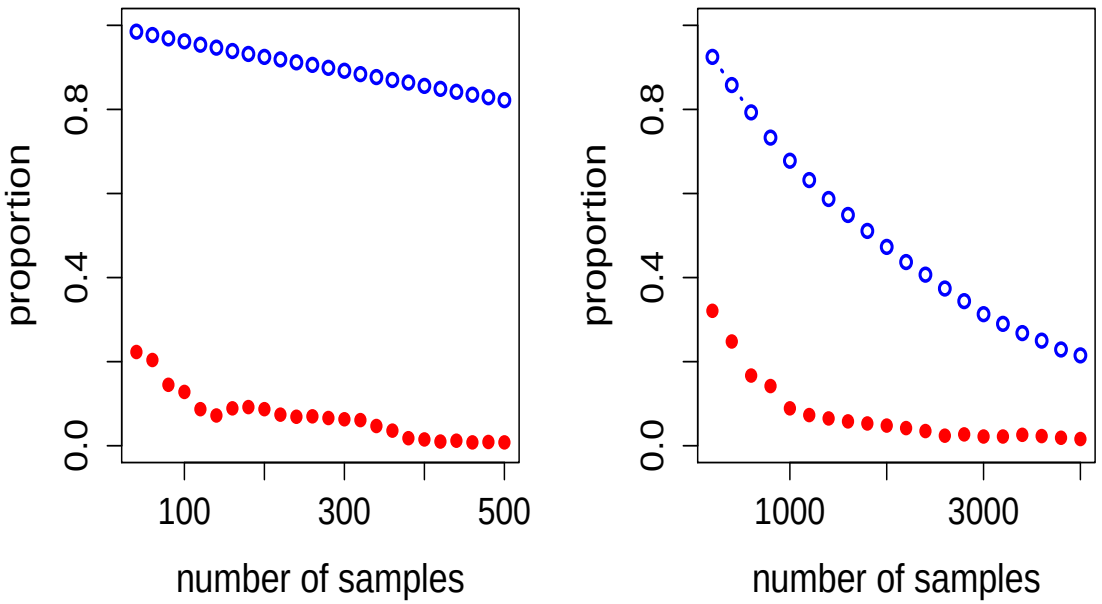

Figure 16: Test error rate and proportion of missing vectors in training set (blue) versus training set size. Left: for 20 overlapping squares. Right: for 200 overlapping squares.

Note that it was not necessary for the parent distribution to lie on a 2-dimensional grid. The training could be generated using a uniform random generator. The disadvantage is that we would not be able to determine the degree of representation of the training set.