Department of Computer Science

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Hamilton, New Zealand

Random Relational Rules

by

Grant Anderson

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

at the

University of Waikato

in the subject of

Computer Science

November 2008

Abstract

In the field of *machine learning*, methods for learning from single-table data have received much more attention than those for learning from multi-table, or *relational* data, which are generally more computationally complex. However, a significant amount of the world's data is relational. This indicates a need for algorithms that can operate efficiently on relational data and exploit the larger body of work produced in the area of single-table techniques.

This thesis presents algorithms for learning from relational data that mitigate, to some extent, the complexity normally associated with such learning. All algorithms in this thesis are based on the generation of random relational rules. The assumption is that random rules enable efficient and effective relational learning, and this thesis presents evidence that this is indeed the case. To this end, a system for generating random relational rules is described, and algorithms using these rules are evaluated. These algorithms include direct *classification*, classification by *propositionalisation*, *clustering*, *semi-supervised learning* and generating *random forests*.

The experimental results show that these algorithms perform competitively with previously published results for the datasets used, while often exhibiting lower runtime than other tested systems. This demonstrates that sufficient information for classification and clustering is retained in the rule generation process and that learning with random rules is efficient.

Further applications of random rules are investigated. Propositionalisation allows single-table algorithms for classification and clustering to be applied to the resulting data, reducing the amount of relational processing required. Further results show that techniques for utilising additional unlabeled training data improve accuracy of classification in the semi-supervised setting. The thesis also develops a novel algorithm for building random forests by making efficient use of random rules to generate trees and leaves in parallel.

Acknowledgements

Though this thesis lists only my name as the author, others have made substantial contributions, and I welcome the opportunity to give them the thanks and acknowledgement they deserve.

Bernhard Pfahringer, my primary supervisor, has given unstintingly of his knowledge, experience and time. Without his enthusiasm, advice, intuition, patience and occasional reassurance, this thesis could not have been completed. He was always able to find improvements that could be made, and motivate me to push on a bit further. I have also enjoyed our conversations on politics and the world in general. He has been both a great supervisor and a good friend.

Geoff Holmes and Eibe Frank formed the rest of my supervisory committee. I am grateful for their support and comprehensive feedback, particularly during the writing of this thesis, and for their willingness to take the time to discuss details with me.

I would also like to thank my colleagues in the Machine Learning Lab - some have graduated and moved on; some are still around as I write this. Richard, Gabi, Jimmy, Stefan, Peter, Reuben, Claudia, Jesse, Robert and several others were and are excellent sources of advice and conversation.

My family have supported me in many ways during my studies - including tactfully refraining from asking me how my thesis was going as the deadline approached! Especial thanks to my parents, whose support and encouragement throughout my life has enabled me to reach this point.

The friends who made the time away from work so enjoyable also deserve appreciation thanks to Tom, Ron, Arthi, Preeti, Topper, Laura, Glen and Scott, I was often able to relax and put aside my research for a while. Thanks also to Justen, Marie, Kathy and Pat for the great time we had in Canada.

Finally, my wife Elizabeth has been tremendously supportive throughout this endeavour. Her love, encouragement, companionship and confidence during these often trying times have been much appreciated.

Contents

List of Tables

List of Figures

List of Algorithms

Department of Computer Science

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Hamilton, New Zealand

Random Relational Rules

by

Grant Anderson

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

at the

University of Waikato

in the subject of

Computer Science

November 2008

Abstract

In the field of *machine learning*, methods for learning from single-table data have received much more attention than those for learning from multi-table, or *relational* data, which are generally more computationally complex. However, a significant amount of the world's data is relational. This indicates a need for algorithms that can operate efficiently on relational data and exploit the larger body of work produced in the area of single-table techniques.

This thesis presents algorithms for learning from relational data that mitigate, to some extent, the complexity normally associated with such learning. All algorithms in this thesis are based on the generation of random relational rules. The assumption is that random rules enable efficient and effective relational learning, and this thesis presents evidence that this is indeed the case. To this end, a system for generating random relational rules is described, and algorithms using these rules are evaluated. These algorithms include direct *classification*, classification by *propositionalisation*, *clustering*, *semi-supervised learning* and generating *random forests*.

The experimental results show that these algorithms perform competitively with previously published results for the datasets used, while often exhibiting lower runtime than other tested systems. This demonstrates that sufficient information for classification and clustering is retained in the rule generation process and that learning with random rules is efficient.

Further applications of random rules are investigated. Propositionalisation allows single-table algorithms for classification and clustering to be applied to the resulting data, reducing the amount of relational processing required. Further results show that techniques for utilising additional unlabeled training data improve accuracy of classification in the semi-supervised setting. The thesis also develops a novel algorithm for building random forests by making efficient use of random rules to generate trees and leaves in parallel.

Acknowledgements

Though this thesis lists only my name as the author, others have made substantial contributions, and I welcome the opportunity to give them the thanks and acknowledgement they deserve.

Bernhard Pfahringer, my primary supervisor, has given unstintingly of his knowledge, experience and time. Without his enthusiasm, advice, intuition, patience and occasional reassurance, this thesis could not have been completed. He was always able to find improvements that could be made, and motivate me to push on a bit further. I have also enjoyed our conversations on politics and the world in general. He has been both a great supervisor and a good friend.

Geoff Holmes and Eibe Frank formed the rest of my supervisory committee. I am grateful for their support and comprehensive feedback, particularly during the writing of this thesis, and for their willingness to take the time to discuss details with me.

I would also like to thank my colleagues in the Machine Learning Lab - some have graduated and moved on; some are still around as I write this. Richard, Gabi, Jimmy, Stefan, Peter, Reuben, Claudia, Jesse, Robert and several others were and are excellent sources of advice and conversation.

My family have supported me in many ways during my studies - including tactfully refraining from asking me how my thesis was going as the deadline approached! Especial thanks to my parents, whose support and encouragement throughout my life has enabled me to reach this point.

The friends who made the time away from work so enjoyable also deserve appreciation thanks to Tom, Ron, Arthi, Preeti, Topper, Laura, Glen and Scott, I was often able to relax and put aside my research for a while. Thanks also to Justen, Marie, Kathy and Pat for the great time we had in Canada.

Finally, my wife Elizabeth has been tremendously supportive throughout this endeavour. Her love, encouragement, companionship and confidence during these often trying times have been much appreciated.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In 1950, Alan Turing published a paper asking the question "Can machines think?" [79], referring several times to 'learning machines'. Machine learning, a term coined by Arthur Samuel later in the 1950s [71], has come to refer to the study and development of algorithms that can learn new knowledge from supervised and unsupervised data [29]. Generally, machine learning algorithms learn new knowledge from *single* tables of data. The International Conference on Machine Learning (ICML) was first held in 1982, and machine learning is a very active field of research.

Data mining is the automated extraction of knowledge and patterns from databases, and is closely related to machine learning, making use of techniques developed in that area. Both are concerned with the discovery of patterns and knowledge in data, with one of the main differences between the two fields being the quantity of data analysed – data mining is especially focused on large or complex databases [29]. Like machine learning algorithms, data mining algorithms learn knowledge from *single* tables of data. Due to the scale of data being analysed, algorithmic complexity is more important in data mining than it is in machine learning. According to [81], the first book on data mining was published in 1991, collecting papers from the first 'Knowledge Discovery in Databases' workshop, held in 1989. Like machine learning, data mining is currently a highly active field.

Inductive logic programming (ILP) is also related to machine learning – it is a research area "at the intersection of logic programming and machine learning" [53]. It is concerned with learning from examples, within the framework of clausal logic [56]. ILP is concerned with algorithms that can learn from relational data and employ background knowledge. ILP-based algorithms learn knowledge from *multiple* tables of data. The first ILP conference was held in

1991, and ILP is also an area of ongoing research.

Relational data mining (RDM) is the extraction of knowledge and patterns from relational databases in particular [22]. By definition, therefore, RDM algorithms must learn knowledge from *multiple* tables of data, and RDM makes use of techniques from ILP just as techniques from machine learning are applied to single-table data mining. Similarly to data mining, due to the nature of the data involved, the complexity of the algorithms used for relational data mining is more important than for ILP. Currently there are no conferences devoted to relational data mining, although general data mining conferences include work on RDM and there have been workshops in this area since 2001.

This thesis presents algorithms for learning from relational data that alleviate, to some degree, the complexity normally present in relational learning, and thus contributes to the field of relational data mining. All of these algorithms are based around the generation of random relational rules. This thesis presents evidence to confirm the hypothesis that these rules enable efficient and accurate relational learning. The algorithm used to generate random relational rules is described, and a number of algorithms using the rules are evaluated.

The remainder of this chapter gives background information applicable to subsequent chapters of the thesis. Section 1.1 provides the motivation for this thesis. Sections 1.2-1.3 define and provide examples for classification and evaluation. Section 1.4 briefly discusses two attribute-value algorithms that are used in Chapters 3 and 5, while Section 1.5 discusses techniques for producing ensembles of models, as ensembles are employed in Chapters 2 and 6. Section 1.6 defines relational data and gives examples of algorithms that use relational data. Section 1.7 discusses two families of methods for learning from unlabeled data that are applied in Chapters 4 and 5. Section 1.8 describes the content of the remaining chapters of the thesis.

1.1 Motivation

Substantial amounts of the world's data are stored in relational databases. According to the International Data Corporation (IDC), relational databases were a multi-billion dollar industry in 2006 [57]. Operations on relational data can have high computational complexity [65]. Nevertheless, relational approaches are often preferred to single-table methods for use with structured data. The latter must necessarily ignore the structure of the data, and thus cannot make

use of any information contained therein. Given this heavy reliance on relational data, there is a need for learning methods that operate efficiently on this data – that is, relational data mining algorithms. This need provides the motivation for this thesis – to provide algorithms to extract information from relational data, while taking measures to alleviate the computational complexity arising from processing such structured data. In addition, the body of work in flat-file learning is currently much larger than that in relational learning. Therefore, a secondary goal of this research is to determine methods for allowing relational learning techniques to make use of sophisticated flat-file approaches.

1.2 Classification

One common data mining task is *classification*. This task is performed by several algorithms that are described in later chapters, and is thus described in this section. Usually data for this task will be in the form of a set of examples (also called *instances*), each labeled with a *class*. An example dataset of this form is shown in Table 1.1 – this dataset (from [63]) is based on deciding if weather conditions are suitable for playing golf, so the class is 'play' and the label for each instance is either 'yes' or 'no'. A classification algorithm will seek to build a *model*, or classifier, based on patterns in the dataset that relates the qualities of the instances to the class labels of those instances. The instance qualities, such as 'outlook' and 'humidity', are also called *attributes*.

A model based on this dataset could be:

```
if (Outlook = overcast) Play = yes
else if (Outlook = sunny AND Humidity = high) Play = no
else if (Outlook = rainy AND Windy = true) Play = no
else Play = yes
```

This model gives the correct yes/no choice for Play for each instance in the dataset. A 'model' that described each attribute of each instance would also give the correct yes/no choices, as shown here:

```
if (Outlook = sunny AND Temperature = hot
  AND Humidity = high AND Windy = false) Play = no
if (Outlook = sunny AND Temperature = hot
  AND Humidity = high AND Windy = true) Play = no
```

Table 1.1: Example Dataset – Weather

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

```
if (Outlook = overcast AND Temperature = hot
  AND Humidity = high AND Windy = false) Play = yes
...
```

This model still gives the correct choices, but does not generalise based on the patterns in the dataset, which could make it less accurate on new instances that have properties that differ from those already seen. When a model is accurate on the data used to construct it, at the expense of the ability to generalise to new data, it is said to be *overfitting*.

1.3 Evaluation

This section discusses the evaluation of a model and two methods that are used to evaluate the performance of algorithms in subsequent chapters of this thesis.

Evaluating a model on the data used to train it can result in overly optimistic estimates of the quality of that model. To avoid this, models are generally evaluated by testing their ability to generalise to new data. This is accomplished by dividing the dataset into training and test sets – the model is constructed on the training set, and then used to predict the class of each instance in the test set. This allows the proportion of correct predictions to be used as an estimate of the quality of the model. The proportion of correct

predictions is also called *accuracy* (also often represented as a percentage), while the proportion of incorrect predictions is also known as the *error rate*.

For example, if the first seven instances from the weather data above were used for training, and the remaining seven set aside as test instances, the following model could be produced:

```
if (Outlook = sunny) Play = no
else if (Outlook = rainy AND Windy = true) Play = no
else Play = yes
```

This model produces correct results on the first seven instances. However, if this model is applied to the test instances, it predicts 5 of the 7 instances correctly (incorrectly predicting 'no' for the two Sunny outlooks that in fact are of class 'yes'), giving an accuracy of $\frac{5}{7}$ and an error rate of $\frac{2}{7}$.

Cross-validation

One standard method for evaluating classification algorithms is *stratified ten-fold cross-validation*. This method allows for efficient use of a dataset by ensuring that each instance is used both for training and testing, where a simple train-test split would use the training instances solely for training and the test instances only for testing. It randomly divides the dataset into ten parts, or 'folds'. The folds are stratified, meaning that the classes of data are represented in each fold in approximately the same distribution as in the full dataset. Each fold is used once as a test set, while the remaining nine folds form the training set for the classification algorithm. The combination of the ten results is regarded as providing a reasonable estimate of the quality of a classification algorithm, and the average result of repeated ten-fold cross-validation can provide an even more accurate estimate.

AUC

Another measure of classifier quality is the Area Under the 'Receiver Operating Characteristic' (ROC) Curve, or *AUC* [8]. In a setting with two classes, an ROC curve depicts the relationship between the proportion of *true positives* (correctly classified instances of the positive class) and the proportion of *false positives* (incorrectly classified instances of the negative class) as the threshold for predicting 'positive' moves from one to zero. The predictions are sorted according to a measure of confidence in their prediction of the positive class,

and the predictions are included in order from highest confidence to least. Thus the most desirable point on such a curve is (0.0, 1.0), indicating that all the positive instances are included, without any negative instances (false positives). This formulation for AUC only applies to two-class problems. In this thesis all but one of the classification problems are such two-class problems, and therefore AUC can be computed and used for comparison in almost all cases.

The area under the ROC curve is equivalent to the probability that, if one positive and one negative instance are selected at random, the confidence in positive class prediction will be greater for the positive instance than for the negative instance. Thus, higher AUC values represent better predictions. As each axis on the ROC curve ranges from zero to one, the area under the ROC curve can also range from zero to one. An AUC value of greater than 0.5 indicates the classifier performs better than randomly assigning classes to instances, with higher values indicating increasingly accurate classification results. An AUC value of 0.5 is a result equivalent to that which would be expected from a classifier that did not discriminate between the classes at all, with performance no better than random. An AUC of less than 0.5 indicates that the classifier would actually be improved if its predictions were inverted (positive predictions becoming negative and negative predictions becoming positive).

An example ROC curve based on an arbitrary assignment of confidence ordering to a dataset of 50 instances, split evenly into 25 positive and 25 negative instances, is shown in Figure 1.1. The curve is always above the line of no discrimination, and the area under the curve is 0.7264, indicating that the hypothetical classifier discriminates between classes to a reasonable extent.

1.4 Attribute-value Learning Algorithms

This section briefly discusses two attribute-value algorithms that are used in subsequent chapters, due to their accuracy and efficiency on propositionalised data.

1.4.1 SMO

A linear Support Vector Machine (SVM) is a hyperplane that separates a set of positive instances from a set of negative instances with the maximum margin

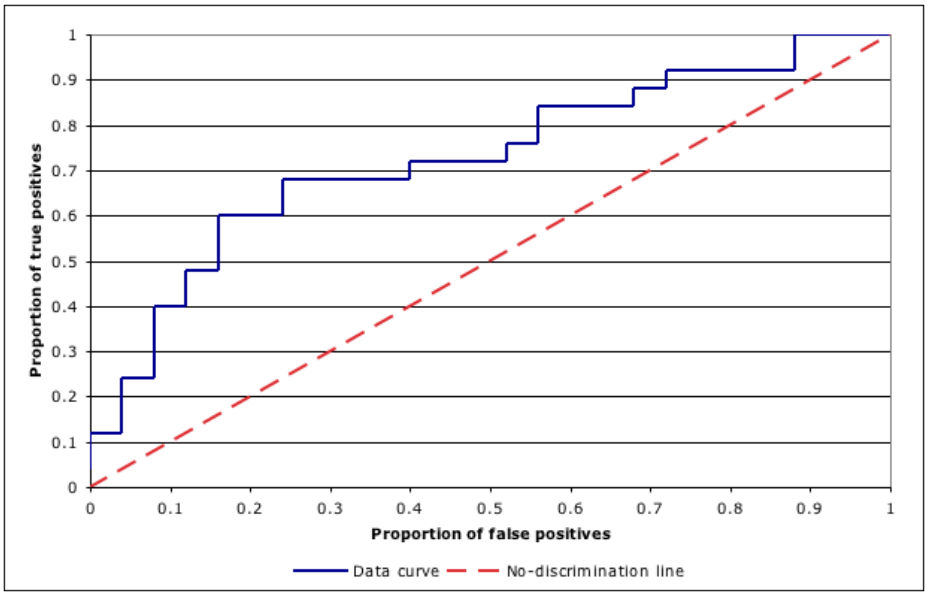


Figure 1.1: An example ROC curve

[81], assuming that the data is linearly separable. The calculation required to train an SVM is a complex Quadratic Programming problem. Sequential Minimal Optimisation (SMO) [60] is an SVM algorithm that decomposes the calculation into a series of smaller computations, solving the smallest possible optimisation problem at each step. This decomposition results in a substantial speed improvement and the ability to run in a relatively small amount of memory.

Test instances are classified by determining the side of the hyperplane on which they lie and assigning the class that matches the training instances on that side.

As SMO is based on the computation of dot products, it has linear complexity with regard to the number of attributes in the data. SMO for training a linear SVM has a complexity of approximately $\mathcal{O}(MN^2)$ where M is the number of attributes and N is the number of instances in the dataset [78].

1.4.2 Logistic Regression

Logistic regression produces an equation to describe training data, assigning regression coefficients (numeric weights) to each attribute in the data with the aim of maximising the accuracy of probability estimates for classes.

A logistic regression algorithm classifies new instances by applying the regression equation to their attributes to obtain a probability distribution across

the possible classes. In a nominal classification case, this results in the class with the highest probability being predicted.

In addition, ridge estimators can be used to improve the models generated by logistic regression in situations which produce unstable parameter estimates – in particular, with large numbers of attributes or highly correlated attributes [47].

A simple logistic regression algorithm has a complexity of $\mathcal{O}(M^2N + M^3)$, where M is the number of attributes and N is the number of instances in the dataset [39].

1.5 Ensemble Methods

This section discusses, and gives examples of, methods for ensemble learning. Ensemble methods can combine classifiers to produce a model with greater accuracy than its components, and are thus of interest in the context of random relational rules.

The output from multiple classifiers can be combined, with the aim of producing more accurate results than can be obtained by the individual classifiers. Some ensemble methods, such as Bagging and Boosting, are generalised procedures that can be applied to arbitrary classification algorithms. Random Forests are a specialised case of bagging, with a randomised tree used as the classification algorithm. Random forests have been shown to be accurate and efficient [12]. A random forest algorithm based on random relational rules is described in Chapter 6.

An important quality of an ensemble of classifiers is that its component classifiers be *diverse* – that is, they make different errors on test data [19]. The complexity of ensemble methods is dependent on the complexity of the individual models being generated, and generally linear with regard to the number of those models (although some ensemble methods can run in parallel).

1.5.1 Bagging

Bagging, or *b*ootstrap *agg*regat*ing*, takes a classification algorithm and a dataset and produces a specified number of classifiers. This is accomplished by sampling the training data with replacement (generally a number of times equal to the number of instances in the training data) for each classifier to be produced and applying the algorithm to be bagged to each sampled set [10],

as shown in Algorithm 1.

Algorithm 1 Pseudocode for the Bagging algorithm

To generate c classifiers:
for i = 1 to c **do**
  Sample the training data, D, with replacement to produce D_i
  Apply classification algorithm A to D_i to produce the classifier M_i
end for
To classify test instances:
for Each test instance T **do**
  for i = 1 to c **do**
    Predict the class of T using M_i
  end for
  Combine the predictions for an overall prediction for T
end for

Bagging is most beneficial when the classification algorithm is sensitive to small changes in the training data. For algorithms where small perturbations in the training data do not affect classifier construction to any great extent, Bagging will produce classifiers that make very similar predictions to each other and whose combined result will be rather similar to that produced by a single classifier.

1.5.2 Boosting

Boosting produces an ensemble of classifiers consecutively, with each new classifier being influenced by those previously built [27]. Each instance in the training set is assigned a weight – initially all instances have equal weight. As each classifier is added to the ensemble, the weights of instances correctly classified by that classifier are decreased, and the weights of those incorrectly classified are increased, as shown in Algorithm 2.

The generated classifiers are also weighted for prediction – their weights are determined by their errors on the weighted training data. This gives higher weightings to those classifiers that perform well on highly-weighted (frequently misclassified) instances.

1.5.3 Random Forests

A *random forest* is an ensemble of *decision trees*. In this section, decision trees are described, followed by *information gain* (a measure used in decision tree contruction), and then random forests themselves are detailed.

Algorithm 2 Pseudocode for the boosting algorithm

```
To generate c classifiers:
Initialise all instance weights to be equal
for i = 1 to c do
    Apply classification algorithm A to the weighted training data to produce
    the classifier M_i
    Increase weights of instances incorrectly classified by M_i
    Decrease weights of instances correctly classified by M_i
    Assign a weight to M_i based on its performance on the weighted training
    data
end for
To classify test instances:
for Each test instance T do
    Initialise all class weights to be 0
    for i = 1 to c do
        Predict the class of T using M_i
        Add the weight of M_i to the weight of the class it predicts for T
    end for
    Predict the class with the highest weight for T
end for
```

Decision Trees

A decision tree is a classifier with a tree structure – the internal nodes represent features in the data and the leaf nodes represent classifications [61]. To build the tree, a root node is initialised to contain all instances of the training data. Then the root node is 'split' into leaves according to the possible values of a feature in the data, and the instances are apportioned to leaves depending on their values for that feature. In the simplest case, this process is repeated for each leaf until all leaves are class-pure. When building a decision tree, the 'best' feature to split each internal node on is deterministically selected. For example, for the decision tree learner ID3 the best feature for node splitting is determined using the *information gain* metric. Pseudocode for ID3 is given in Algorithm 3.

Once the tree is constructed, each test instance is then classified by beginning at the root of the tree and following a path to a leaf. The path taken by the instance is based on the attributes of the instance and the results of the tests at each internal node. The test instance is classified as being of the class of the leaf it is assigned to. In more sophisticated tree construction algorithms than the one given in Algorithm 3 leaf nodes may not be class-pure, in which case the class that makes up the majority of the training instances at the leaf

Algorithm 3 Pseudocode for the ID3 algorithm

BuildTree(N):
if N contains instances of only one class **then**
  return
else
  Select the feature F with the highest information gain to split on
  Create f child nodes of N, $N_1...N_f$, where F has f possible values $(F_1...F_f)$
  for $i = 1$ to f **do**
    Set the contents of N_i to D_i, where D_i is all instances in N that match F_i
    Call BuildTree(N_i)
  end for
end if

will be assigned to test instances. A decision tree for the Weather dataset (from [81]) is given in Figure 1.2. C4.5 [63] is an upgrade of ID3 that includes improvements such as pruning and the ability to handle continuous attributes and missing values.

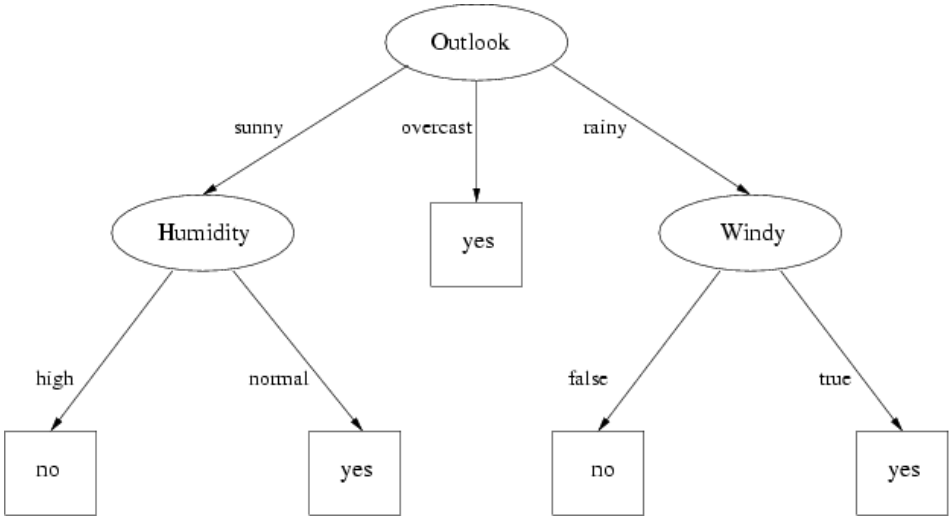


Figure 1.2: Decision tree for Weather data

Information Gain

The *information* of a node in a decision tree measures the number of bits required to specify the class of a new instance arriving at that node, given the classes of the current instances at the node, and is calculated as shown in Equation 1.1 for a node containing instances from two classes (A and B).

$$\text{info(node}(x,y)) = -\frac{x}{x+y}log_2(\frac{x}{x+y}) - \frac{y}{x+y}log_2(\frac{y}{x+y}) \quad (1.1)$$

Where:
x = number of instances of class A
y = number of instances of class B

The information of a split is then determined by the weighted sum of the information of the newly created nodes, as shown in Equation 1.2 for a node T being split by a test X into n different leaf nodes.

$$info_X(T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \times info(T_i) \quad (1.2)$$

The *information gain* is then the pre-split information less the post-split information. If the information required to specify the class of a new instance is less after the split than before it (which is the desirable outcome), then the information gain will be a positive number – the decrease in information required.

An example of this calculation for the Weather dataset from Section 1.2 follows, splitting the full dataset on the 'outlook' attribute:

Table 1.2: Class values splitting on the 'outlook' attribute

Outlook	Class values
Sunny	yes, yes, no, no, no
Overcast	yes, yes, yes, yes
Rainy	yes, yes, yes, no, no

The information of the full dataset (9 'yes', 5 'no') is given by:

$$info(9,5) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.940 \text{ bits} \quad (1.3)$$

The information of the split on 'outlook' is given by:

$$\begin{aligned} info(outlook) &= \frac{5}{14}info(sunny) + \frac{4}{14}info(overcast) + \frac{5}{14}info(rainy) \\ &= \frac{5}{14}(-\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5}) + \\ &\quad \frac{4}{14}(-\frac{4}{4}log_2\frac{4}{4} - \frac{0}{4}log_2\frac{0}{4}) + \\ &\quad \frac{5}{14}(-\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5}) \\ &= 0.693 \text{ bits} \end{aligned} \tag{1.4}$$

(where $0log_20$ is defined to be 0)
And the information gain of the split is given by:

$$\begin{aligned} \text{gain} &= information(dataset) - information(outlook) \\ &= 0.940 - 0.693 \\ &= 0.247 \text{ bits} \end{aligned} \tag{1.5}$$

The complexity of building a C4.5 decision tree is $\mathcal{O}(MNlogN)$ for tree construction, with an additional $\mathcal{O}(N(logN)^2)$ for pruning, where M and N refer to the attributes and instances in the data [81]. This cost assumes that the tree depth is $\mathcal{O}(logN)$, and that sorting of numeric attributes need only occur once.

Random Forests

A random forest [11] is an ensemble of decision trees. Random forests use Bagging to produce a randomly sampled set of training data for each of the trees built. They also select splitting features semi-randomly – a random subset of a given size is produced from the space of possible splitting features, and the best feature deterministically selected from that subset. An example of random forest construction is shown in Algorithm 4.

To classify a test instance, a random forest classifies the instance using each of the trees in the forest, then combines the results. The method used to combine the results can be as simple as predicting the class predicted by the greatest number of trees, or a more sophisticated method can be employed

Algorithm 4 Pseudocode for the random forest algorithm

To generate c classifiers:
for $i = 1$ to c **do**
  Randomly sample the training data D with replacement to produce D_i
  Create a root node, N_i containing D_i
  Call BuildTree(N_i)
end for

BuildTree(N):
if N contains instances of only one class **then**
  return
else
  Randomly select x% of the possible splitting features in N
  Select the feature F with the highest information gain to split on
  Create f child nodes of N, $N_1...N_f$, where F has f possible values ($F_1...F_f$)
  for $i = 1$ to f **do**
    Set the contents of N_i to D_i, where D_i is all instances in N that match F_i
    Call BuildTree(N_i)
  end for
end if

– for example, taking into account the relative proportions of classes at the leaves reached by the instance, if the trees produce mixed-class leaf nodes, to assign confidence values to individual tree predictions.

1.6 Relational Data Mining

As random relational rules are the focus of this thesis, this section describes the representation of relational data, as opposed to propositional data. Algorithms that learn from relational data are described and the computational complexity of relational data mining is discussed. Propositionalisation – the process of transforming relational data into a propositional representation – is also described.

1.6.1 Representing Relational Data

The example data in Section 1.2 (Table 1.1) is *propositional*, or 'flat-file'. It consists of a single table of data, in which each instance of data is assigned a value for each attribute, allowing a representation of an instance as a series of

Table 1.3: Example instances – East-West (trains)

Train	Direction
train1	eastbound
train2	eastbound
...	...
train19	westbound
train20	westbound

attribute-value pairs. For example, the first instance from Table 1.1 could be described as follows:

```
Outlook = sunny
Temperature = hot
Humidity = high
Windy = false
Play? = no
```

Relational data consists of multiple tables of data. An example of this (from the East-West trains dataset [46]) is given in tabular form in Tables 1.3-1.5 and illustrated in Figure 1.3.

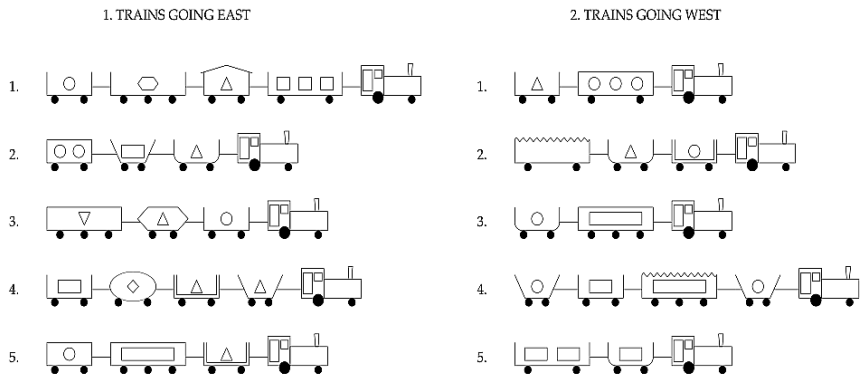


Figure 1.3: East-West Trains data

The trains in this dataset are labeled as eastbound or westbound, and each train is composed of a number of cars – most with some load, but some with no load – that differs from train to train. Representing this dataset as attribute-value pairs is not straightforward, as each train is composed of multiple cars, each with their own attributes.

First-order logic allows relational datasets to be represented in a manner that preserves the relationships between the tables of data. Tables correspond

Table 1.4: Example instances – East-West (cars)

Train	Car	Pos^n	Shape	Length	Sides	Roof	Wheels
train1	car1	1	rectangle	short	not_double	none	2
train1	car2	2	rectangle	long	not_double	none	3
train1	car3	3	rectangle	short	not_double	peaked	2
train1	car4	4	rectangle	long	not_double	none	2
train2	car5	1	rectangle	short	not_double	flat	2
train2	car6	2	bucket	short	not_double	none	2
train2	car7	3	u_shaped	short	not_double	none	2
...	...	...	...	...	...	...	...
train19	car58	1	rectangle	long	not_double	flat	3
train19	car59	2	rectangle	long	not_double	flat	2
train19	car60	3	rectangle	long	not_double	none	2
train19	car61	4	u_shaped	short	not_double	none	2
train20	car62	1	rectangle	long	not_double	flat	2
train20	car63	2	u_shaped	short	not_double	none	2

Table 1.5: Example instances – East-West (loads)

Car	Shape	Quantity
car1	circle	1
car2	hexagon	1
car3	triangle	1
car4	rectangle	3
car5	circle	2
car6	rectangle	1
car7	triangle	1
...	...	...
car58	rectangle	3
car59	rectangle	3
car61	triangle	1
car62	hexagon	1
car63	triangle	1

to *predicates*, and the columns of those tables correspond to *arguments* within the predicates. Each row of each table is represented by a *tuple* with a number of arguments (*arity*) equal to the number of columns in the table. An example of this representation for the first train, train1, is given below:

```
train(train1, eastbound).
car(train1, 1, rectangle, short, not_double, none, 2, load1).
car(train1, 2, rectangle, long, not_double, none, 3, load2).
car(train1, 3, rectangle, short, not_double, peaked, 2, load3).
car(train1, 4, rectangle, long, not_double, none, 2, load4).
load(load1, circle, 1).
load(load2, hexagon, 1).
load(load3, triangle, 1).
load(load4, rectangle, 3).
```

A example of a first-order rule that could be produced using this data (and covers train1) is:

```
train(TrainID, eastbound):-
  car(TrainID, Position, short, Sides, Roof, Wheels, LoadID),
  Wheels < 3,
  load(LoadID, circle, Quantity).
```

The rule is given in the syntax of the first-order-logic-based programming language Prolog [9]. To describe it in more detail, the rule has been split into its individual literals:

```
train(TrainID, eastbound):-
```

The portion of the rule preceding the ':-' is the 'head' of the rule. Rules can be regarded as "body implies head", in that the rule can be read as "*if* body *then* head". The head of this rule describes trains that are eastbound. By Prolog convention, variables within the tuples begin with uppercase letters and constants begin with lowercase letters, so in this rule, 'short' and 'circle' are constants and 'TrainID', 'Position', 'Sides', 'Roof', 'Wheels', 'LoadID' and 'Quantity' are variables.

```
car(TrainID, Position, short, Sides, Roof, Wheels, LoadID),
```

The portion of the rule following the ':-' is a series of comma-separated literals that make up the 'body' of the rule. When the above literal is being evaluated, the TrainID variable will already have been *bound* to the value for a particular train (at least for the purposes of the system described in this thesis), so at this point the rule covers all trains containing at least one short car.

```
Wheels < 3,
```

When the above literal is being evaluated, the Wheels variable will already have been bound by the previous literal. Thus, this literal restricts the rule to only covering trains that contain a short car with less than three wheels.

```
load(LoadID, circle, Quantity).
```

When this final literal is being evaluated, the LoadID variable will already have been bound by the first literal of the rule body. Thus, this literal restricts the rule to only covering trains that contain one or more short cars with less than three wheels that have circular loads. The full rule can be read as "if a train contains a short car and that car has less than three wheels and a circular load, then the train is eastbound". Rules are terminated with a period. For clarity, a variable that is never used after its introduction in a rule can be replaced with an underscore, resulting in the following for the example rule, as 'Position', 'Sides', 'Roof' and 'Quantity' are never used after their introduction:

```
train(TrainID, eastbound):-
  car(TrainID, _, short, _, _, Wheels, LoadID),
  Wheels < 3,
  load(LoadID, circle, _).
```

1.6.2 Inductive Logic Programming

Inductive Logic Programming (ILP) systems learn patterns from data expressed in logical representations such as first-order logic – that is, relational data. This section describes ILP systems used for comparison in subsequent chapters.

FOIL

The FOIL algorithm [62] (First Order Inductive Learner) is a greedy covering algorithm. It begins with an empty theory and an empty rule, then iteratively adds the 'best' literal(s) (determined by greedy search using an *information gain* metric, although 'gainless' literals can also be added if they introduce new variables) to the rule until the rule only covers instances of a single class. Once this point has been reached, the rule is added to the theory, the instances covered by the rule are removed from the training data and the process is repeated on the new training set. This results in a 'decision list' of rules for classifying new data – the rules are applied in order to each test instance until one matches, at which point the test instance is assigned the class of the matching rule.

FOIL is capable of learning recursive concepts [64] and employs a sophisticated scheme to prevent infinite recursion. FOIL also employs 'checkpoints' such that if a rule cannot be completed satisfactorily (which can occur if there is no literal that can be added or if adding another literal will make the rule too complex with regard to the training data), a useful part of the rule can be retained and the search restarted from that point. FOIL also employs pruning after rules have been generated to remove unnecessary literals and in some cases improve the coverage of the rules. As a search-based relational rule generator, FOIL is used for comparison with random relational rules.

TILDE

TILDE (Top-down Induction of Logical DEcision trees) is an algorithm for constructing decision trees from relational data [6]. TILDE is a relational upgrade of the C4.5 algorithm [63] for decision tree construction, using (by default) the same information gain heuristic to evaluate possible features for splitting nodes. The major difference between TILDE and C4.5 lies in the computation of those features.

C4.5 deals with propositional, attribute-value data, and so uses tests that compare an attribute to a value – for example, 'Outlook = Sunny' or 'Temperature > 16.3'. TILDE, dealing with relational data, must utilise tests that involve a more complex representation. TILDE describes features using first-order logic, with each feature being composed of one or more literals.

To illustrate the operation of TILDE, a small dataset and example tree, derived from those given in [5], are shown here in Tables 1.6-1.8 and Figure

Table 1.6: Example instances – Maintenance (machines)

Machine	Action
machine1	fix
machine2	sendback
machine3	sendback
machine4	ok

Table 1.7: Example instances – Maintenance (worn parts)

Machine	Worn Part
machine1	gear
machine1	chain
machine2	engine
machine2	chain
machine3	wheel

1.4.

The Maintenance dataset consists of four machines, some of which have worn parts. Some of the worn parts can be replaced by a local engineer, and some must be sent back to the manufacturer for replacement. If a machine contains worn parts the engineer cannot replace, it belongs to class 'sendback', while if all worn parts it contains can be replaced by the engineer, it belongs to class 'fix'. If a machine contains no worn parts it belongs to class 'ok'.

In producing the example tree in Figure 1.4, the algorithm initially considers possible literals to add, selecting the one that leads to the greatest information gain – $worn(Machine, Part)$. This literal succeeds on all machines with one or more worn parts, but does not succeed on machines without worn parts. This leads to a left subtree containing three instances of classes 'sendback' and 'fix' and a right subtree containing only the machine with no worn parts. The latter subtree is class-pure, and so becomes a leaf node. The algorithm then considers literals to split the remaining instances, selecting $replaceability(Part, not\text{-}replaceable)$, which creates two class-pure leaf nodes. The instances for which that literal succeeds have parts that the engineer can-

Table 1.8: Example instances – Maintenance (part replaceability)

Part	Replacable
gear	replaceable
chain	replaceable
engine	not-replaceable
wheel	not-replaceable

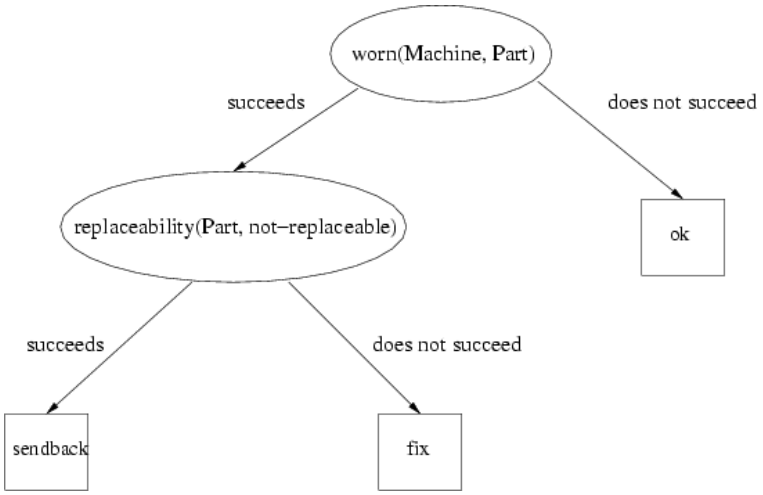


Figure 1.4: Decision tree for Maintenance dataset

not replace and thus must be sent back, and the instance for which it does not succeed contains only worn parts that can be replaced by the engineer.

Leaf nodes can be represented as a path from the root to the leaf – a conjunction of literals. For example, the 'sendback' leaf could be described by:

```
sendback(Machine):- worn(Machine, Part),
                    replaceability(Part, not-replaceable).
```

Paths that include nodes with literals that did not succeed are described using the negation of the conjunction up to that point, as an 'invented predicate' that does not share variables introduced at the unsuccessful node with the rest of the conjunction. The 'fix' leaf is an example of this, using the invented predicate $p1$:

```
fix(Machine):- worn(Machine, Part), not(p1(Machine)).
p1(Machine):- worn(Machine, Part2),
     replaceability(Part2, not-replaceable).
```

The above conjunction describes machines with one or more worn parts, but not machines with any parts that are both worn and non-replaceable. The same result cannot be achieved by simply negating the literal of the unsuccessful node. If this were to be done for 'fix', the result would be:

```
fix(Machine):- worn(Machine, Part),
               not(replaceability(Part, not-replaceable)).
```

This conjunction describes machines with one or more worn parts, one or more of which is replaceable (literally, 'not not-replaceable'), instead of the desired result, machines with one or more worn parts, all of which are replaceable.

For more complex problems, TILDE can use 'lookahead' to add literals that introduce variables but have zero information gain, in conjunction with gainful literals that make use of the introduced variables. Variables introduced in a given node can be only used by the literals in that node, the child of that node for which the test succeeds and the descendants of that child – other nodes are outside the scope of the introduction.

FORF

FORF (First Order Random Forests) [3] is an ensemble combination of the relational decision trees created by TILDE, and can also be regarded as a relational upgrade to random forests, just as TILDE is a relational upgrade of C4.5.

FORF creates an ensemble of decision trees using Bagging to initialise the root nodes. As with propositional random forests, a test at an internal node is chosen using a heuristic (such as information gain) from a set of tests randomly selected from the possible tests at that node. The size of that set relative to the number of possible tests (as a proportion) can be varied by the user.

As a relational random forest generation algorithm, FORF is used for comparison with the random forest algorithm based on random relational rules, described in Chapter 6.

1.6.3 Complexity of Relational Learning

Just as with attribute-value machine learning algorithms, computational complexity is an important quality for relational learning algorithms.

Two factors are considered here – the hypothesis space and the complexity of evaluating a rule in that space. These factors are described using De Raedt's terminology and results from [65].

In the attribute-value (propositional) case, there is only one table of data, and attributes cannot be compared to each other. The 'maximum arity' is thus the number of attributes in the dataset, and there is at most one 'predicate literal' in the rule. This results in a hypothesis space complexity that is exponential only in the number of attributes.

This can be illustrated by considering a simple propositional dataset that describes the state of traffic lights at an intersection. Each light (attribute) has three possible values (red, orange and green). If we consider only one light, it has three possible values. For two lights there are 3×3 or 9 possible combinations of values, and for three lights there are $3^3 = 27$ possible combinations. This progression in the hypothesis space is clearly exponential. In spite of this exponential growth in the hypothesis space, most propositional algorithms are polynomial in the number of instances and attributes. For example, evaluating a 'rule' in this hypothesis space would be linear with respect to the 'arity' of the dataset – there will be at most one comparison to each attribute, for each instance.

In the relational case, on the other hand, where learnt theories are represented as first-order rules, consisting of a conjunction of literals, De Raedt demonstrates that the hypothesis space searched, in addition to being exponential with respect to the maximum arity of a relation, is also exponential with respect to the number of pairs of literals that share common variables and the maximum number of predicate literals in a rule.

Testing whether an instance is covered by a rule is, like the hypothesis space, exponential with regard to the number of pairs of literals sharing common variables in a clause. This means that FOIL and TILDE (and thus FORF, being derived from TILDE) have exponential complexity with regard to this value in the worst case. These algorithms deal with the large hypothesis space using heuristic search techniques. They produce models by selecting one literal at a time according to a heuristic, rather than searching the entire hypothesis space for models, and thus limit the search at any given point to those models that can be formed by adding a single literal to the current rule.

To illustrate the exponential nature of relational data mining, consider a very simple relational dataset of chemical compounds, each containing ten atoms, using the following predicates:

```
compound(CompoundID, Class)
atom(CompoundID, AtomID, Element, Charge)
```

A rule consisting of a single *atom* predicate would require each of the ten possible atoms in each compound to be evaluated. A rule consisting of two such predicates would require each possible combination of two atoms to be evaluated, for a total of one hundred literal evaluations for each compound. A rule constructed from three *atom* predicates would need one thousand literal

evaluations per compound, and so on for more predicates, as shown in Table 1.9.

Table 1.9: Literal evaluations per compound for simple 'Compound' dataset

Atom predicates	Literal evaluations	Total evaluations
1	10	10
2	10×10	100
3	$10 \times 10 \times 10$	1000
...	...	...
x	$10 \times 10 \times 10$ ...	10^x

It can be seen that the number of evaluations required increases rapidly with respect to the number of predicates, even with only ten atoms per compound. For this reason, most ILP systems set a maximum on the number of literals allowed in the rules they generate (without such a limit, an exhaustive search would be searching an unboundedly large space). An increase in the branching factor of the data (in this case the number of atoms per compound) can also have a substantial effect on computational cost – for example, if the compounds consisted of 30 atoms each, this would increase the number of literal evaluations required in the 3-literal case by a factor of 27.

The number of instances in a dataset is not as significant as the previously mentioned factors with regard to complexity – TILDE, for example, has a complexity similar to C4.5 with respect to the number of instances [6].

1.6.4 Propositionalisation

As described in Section 1.6.3, the complexity of coverage testing and the hypothesis space in relational data leads to relational algorithms that are exponential in the worst case. Standard (attribute-value) machine learning algorithms, on the other hand, can be more computationally efficient, but cannot process the relationships and richness of information contained in relational data.

Propositionalisation is the process of transforming relational data into a propositional representation, with the aim of preserving the richness of information in the relational data, while producing a representation of the data that can be utilised by efficient standard machine learning algorithms [41], and is thus of interest with regard to relational data mining.

The most naïve form of propositionalisation would involve simply 'flattening' the data, converting all the information related to a given relational instance into a series of attributes. However, there are a number of difficulties with this approach – for example, placing explicit labels on created attributes that indicate relationships not actually present in the data. To use the Mutagenesis dataset as an example, each Compound instance, under this transformation, would contain attributes for the elements, charges and quanta types for each atom in the compound (with the added complication that every instance would need to contain as many attributes as the largest instance, and smaller instances would have to deal with missing values). This would result in a structure similar to:

```
Compound, Atom1El, Atom1Ch, Atom1Qu, Atom2El, Atom2Ch, Atom2Qu..
```

In this representation the ordering of the atoms within each compound will have an effect on the models produced by attribute-value machine learning algorithms, as correspondences are explicitly drawn between atoms in the same position in the ordering – an ordering not present in the original dataset.

RSD [80] (Relational Subgroup Discovery) takes a logic-based approach to propositionalisation. It computes all possible combinations of first-order predicates (within defined constraints) that could form useful features, then instantiates selected variables to produce features. The features produced in this way are then transformed into a set of Boolean values that denote, for each instance in the data, whether that instance is covered by that feature.

RELAGGS [43] uses relational database-oriented techniques, such as aggregation, to produce propositional representations that are not limited to Boolean values. For example, a set of relational attributes can be summarised by values such as minimum, maximum, mean, mode or frequency counts.

1.7 Learning with Unlabeled Data

The application of random relational rules to learning from data without explicit class labels is discussed in subsequent chapters of the thesis, so a brief overview of such learning is given here.

This section describes two methods for learning from data in which some or all of the instances are without explicit class labels – clustering, in which all of the data is unlabeled, and semi-supervised learning, in which a portion of the data is labeled and the remainder is not.

In *supervised* learning for classification, the learning algorithm is provided with a set of instances with class labels [13]. A model is derived from the labeled training data and used to classify test instances whose labels have been hidden. On the other hand, in *unsupervised* learning, the data has no class labels at all. Instead of classifying the data, unsupervised algorithms search for useful structure and groupings within the data.

1.7.1 Clustering

Clustering is a form of unsupervised learning, in which instances are divided into groups, generally according to some distance measure, such as the Euclidean distance [34]. Clustering is unlike the train-test procedure of supervised learning, where a model is produced on labeled training data and used to assign labels to test data, in that it takes a set of unlabeled instances and attempts to produce a meaningful grouping of instances within that set in the absence of class labels.

Instances within a cluster should be, according to the distance measure used, more similar to each other than they are to instances in other clusters. In fact, the 'Cluster assumption' states "If points are in the same cluster, they are likely to be of the same class" [13] (although the reverse does not necessarily hold), which suggests a method for assessing the quality of clustering if class labels are available for the data.

The k-means algorithm is used to cluster data in Chapter 4 and is thus described here, in Algorithm 5. *K-means* is a *partitioning* clustering algorithm, meaning that it produces a single set of partitions (as opposed to methods that produce a nested series of partitions). K-means begins with some number (k) of randomly assigned partition centres (*centroids*) and iteratively reassigns partition centres (and thus partitioning) until a convergence is reached. An example of this process, reproduced from a diagram previously published in [25], is shown in Figure 1.5. Initially two points are randomly selected as cluster centres and each of the remaining points assigned to whichever of the centre points they are closer to. Then the centroids of each cluster are calculated and the data points are assigned to their nearest centroids repeatedly until the clusters converge to the final stage shown.

It initially creates k cluster centres by selecting instances at random from the data, then assigns each remaining instance to the cluster with the nearest centre. Once this initialisation is complete, the k-means algorithm sets the

centre of each cluster to its centroid and again assigns each instance to the closest centre. This process is repeated until it has converged on a set of centroids that will not change with further iterations.

Algorithm 5 Pseudocode for the k-means algorithm

Randomly select k instances as initial reference points $R_1..R_k$
for each instance i in the data **do**
  Assign instance i to the closest of the k reference points
end for
Set new reference points $R'_1..R'_k$ to be the centroids of the instances assigned to each reference point
converged = *false*
while converged = *false* **do**
  for each instance i in the data **do**
    Assign instance i to the closest of the k reference points
  end for
  Set new reference points $R'_1..R'_k$ to be the centroids of the instances assigned to each reference point
  if $\forall$i $R_i = R'_i$ **then**
    converged = *true*
  end if
end while

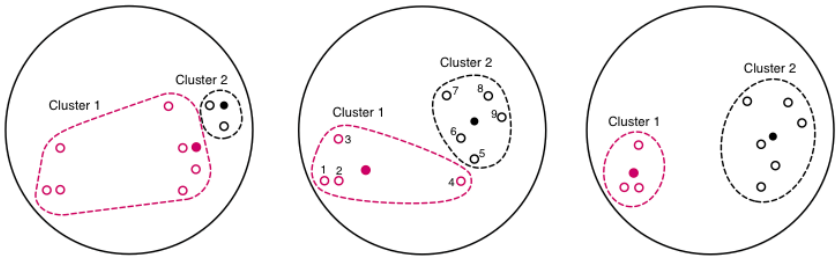


Figure 1.5: k-means clustering process

1.7.2 Semi-supervised Learning

Semi-supervised learning falls between supervised and unsupervised learning, in that semi-supervised algorithms utilise both labeled and unlabeled data. A common task in semi-supervised learning is, given a dataset in which some instances have class labels and some do not, to predict class labels for those instances without them.

An example of a basic Expectation-Maximisation (EM) [18] algorithm for semi-supervised learning is shown in Algorithm 6. This is a more general form of an algorithm discussed in [13],

Algorithm 6 EM algorithm for semi-supervised learning

```
Build a classifier from labeled instances only
while Classifier parameters improve do
    Use the current classifier to estimate a class for each unlabeled instance
    Re-estimate the classifier, given the estimated class membership of each
    instance.
end while
```

This form of learning is especially valuable in situations where there are large amounts of unlabeled data available, but it is expensive (in terms of time and/or money) to obtain labels for that data. Examples of such situations, also given in [13], include:

- Speech recognition – obtaining recorded speech is cheap, but transcribing (and thus labeling) it requires human effort.
- Webpage classification – vast numbers of webpages are freely available, but their classification requires human effort.
- Protein functions - large numbers of protein sequences are available, but classifying the function of a protein may take years of investigation.

1.8 Thesis Structure

In this chapter the background and motivation of this thesis were discussed, along with algorithms and concepts that will be elaborated upon in later chapters.

Chapter 2 introduces the RRR algorithm, discussing rule and ruleset construction and also giving experimental results.

Chapter 3 covers the use of RRR for propositionalisation, introducing the RRR-P algorithm, and compares the experimental results produced by RRR-P to those achieved by other learning algorithms.

Chapter 4 discusses the use of RRR-P for relational clustering via propositionalisation and compares the results to other relational clustering methods.

In Chapter 5 RRR-P is applied to the field of relational semi-supervised learning using propositionalisation. Two methods of making use of unlabeled data are compared to results produced by supervised RRR-P.

Chapter 6 is concerned with the application of RRR to the construction of random forests in a manner that allows trees and individual branches to be grown in parallel.

Finally, Chapter 7 summarises the thesis and its major contributions.

Chapter 2

Random Relational Rules

Exhaustive search in relational learning is generally infeasible, therefore some form of heuristic search is usually employed, such as in FOIL [62]. Randomly generated rules provide a method of searching the space of possible rules that can be arbitrarily scaled by varying the number of rules generated. However, such randomly generated rules are not individually powerful classifiers.

Stochastic discrimination [38] provides a framework for combining arbitrary numbers of weak classifiers in a way where accuracy improves with additional rules, even after maximal accuracy on the training data has been reached. The weak classifiers must have a slightly higher probability of covering instances of their target class than of other classes. As the rules are also independent and identically distributed, the Central Limit theorem applies and as the number of weak classifiers/rules grows, coverages for different classes resemble well-separated normal distributions. Stochastic discrimination is closely related to other ensemble methods like Bagging, Boosting, or Random Forests, all of which have been tried in relational learning [21, 31, 3].

This chapter describes an algorithm for randomly generating relational rules, and a framework for combining those rules to make predictions using stochastic discrimination. Although the rules are not individually powerful classifiers, when they are combined using stochastic discrimination, good predictive accuracy can be achieved.

Section 2.1 describes how the random rules are generated and Section 2.2 describes how these rules are combined into rulesets. Classification using the generated rulesets is described in Section 2.3 and experimental results are reported and compared to those produced by FOIL in Section 2.4, with a comparison to the runtime of FOIL in Section 2.5. Section 2.6 summarises the chapter.

2.1 Rule Generation

This section discusses the generation of random relational rules by the Random Relational Rules (RRR) algorithm. Section 2.1.1 gives the algorithm used by RRR for rule production and Section 2.1.2 discusses the complexity of rule evaluation, first with respect to a single literal and then for entire rules.

2.1.1 Rule Generation Algorithm

The RRR algorithm operates on two-class problems, and produces one set of first-order rules for each class. Unlike Bagging, it does not resample the training set. Rules are generated fully randomly by adding literals to a partial clause in a manner similar to the FOIL algorithm (pseudocode for which is shown in Algorithm 7).

Algorithm 7 Pseudocode for rule generation in the FOIL algorithm

```
Theory: empty
Remaining: all positive instances
while Remaining is not empty do
  Rule: empty
  while Rule covers negative instances do
    for each literal that could be added to the rule do
      Compute the information gain of the literal
    end for
    Add the best literal(s) to Rule
  end while
  Remove positive instances covered by Rule from Remaining
  Add Rule to Theory
end while
```

The algorithm for RRR is given in Algorithm 8. Where a random choice is made in the algorithm, each possible choice has equal probability. Predicate literals must have exactly one variable already bound. Test literals cannot introduce variables.

The stopping condition of FOIL – purity of rule coverage – differs from the rule length limitation of RRR, and RRR does not remove the instances covered by each generated rule from the training set (as the rules generated by RRR are much less likely to have class-pure coverage than those produced by FOIL), but literal-by-literal rule generation is common to both. FOIL also limits the length of its generated rules indirectly, by rejecting literals that would cause the bits required to encode the rule to exceed those required to indicate the

Algorithm 8 Pseudocode for rule generation in the RRR algorithm

```
while Number of literals in rule is less than maximum rule length do
  Randomly select whether to generate a predicate or test literal
  if Generating a predicate literal then
    Randomly select which predicate literal to add
    Add predicate literal, ensuring that exactly one variable is bound, and
    introducing new variables for each other argument
  else if Generating a test literal then
    Randomly select which variable to test
    Randomly select whether to test against a variable or constant
    if Testing against a variable then
      Randomly select a variable to test against
    else if Testing against a constant then
      Randomly select a constant to test against
    end if
    Randomly select an operator (from {=, ≠} or {=, ≠, <, ≤, >, ≥} as
    appropriate)
    Add test literal
  end if
end while
```

instances covered by the rule. RRR computes the coverage of literals once the rule is complete, while FOIL computes the coverage of possible literals before selecting which one to add.

2.1.2 Complexity

Because the structure of the rule production method of RRR has such a similarity to that of FOIL, the two are compared here.

The complexity of constructing and evaluating a rule in RRR is dominated by the cost of the evaluation, which is exponential with respect to the number of literals in the rule, and influenced by the 'branching factor' – for a predicate literal, the number of new bindings in the predicate compatible with the current ones and for a test, the proportion of the current bindings that satisfy the test. This is $\mathcal{O}(c^n)$, where the upper bound for c is the maximum branching factor for any single literal and n is the number of literals.

Evaluating a single literal

The cost to evaluate a single literal is the same for FOIL as it is for RRR – both are dependent on the branching factor. Pazzani and Kibler calculate bounds and estimates for FOIL's search [58], and their terminology will be used here.

For the branching factor, which applies to both RRR and FOIL, for a literal where no new variables are introduced, let the Density of that literal be the proportion of cases where that literal is true. For a literal that introduces new variables, let the Power of the predicate be the maximum number of solutions for that predicate with exactly one variable bound. Given those definitions and that L_i for i = 1 to k gives the literals in a rule of length k, Growth(L_i) is 1 if L_i does not introduce new variables and Power(L_i) if it does (see Equation 2.1).

$$\text{Growth}(L_i) = \begin{cases} 1 & L_i \text{ introduces no new variables} \\ \text{Power}(L_i) & \text{otherwise} \end{cases} \tag{2.1}$$

This leads to an upper bound for the branching factor of:

$$\text{BranchingFactor} \leq \prod_{i=1}^{k} \text{Growth}(L_i) \tag{2.2}$$

For an estimate of the branching factor, the AveragePower of a predicate can be defined as the average number of solutions for that predicate when exactly one variable is bound, and the AverageGrowth of a literal as its Density if no new variables are introduced or its AveragePower if new variables are introduced.

$$\text{AverageGrowth}(L_i) = \begin{cases} \text{Density}(L_i) & L_i \text{ introduces no new variables} \\ \text{AveragePower}(L_i) & \text{otherwise} \end{cases} \tag{2.3}$$

This allows the branching factor to be approximated with:

$$\text{BranchingFactor} \approx \prod_{i=1}^{k} \text{AverageGrowth}(L_i) \tag{2.4}$$

The branching factors given in Equations 2.2 and 2.4 show that both RRR and FOIL have branching factors that grow exponentially with the number of possible solutions to the predicates. Later literals in a rule thus often have a higher branching factor than earlier ones and have a correspondingly greater cost to evaluate.

Table 2.1: Literals considered in rule construction

Literal added	Predicates searched	Tests searched	Total searched
atom(A,B,C,D,E)	2	0	2
atom(A,F,C,G,H)	38	106	144
E > -0.121	124	216	340
H <= 0.011	124	216	340
H > -0.084	124	216	340
E <= -0.112	124	216	340
Total	536	970	1,506

The number of literals evaluated

FOIL faces a higher cost than RRR in rule construction, where, when determining a literal to add, FOIL evaluates all possible literals and RRR randomly selects one. The number of literals FOIL investigates grows exponentially with the arity of the predicates and the number of variables currently in the rule. Thus, as the size of the rule increases, the number of literals FOIL evaluates increases – and as the branching factor usually also increases with the number of literals, the cost to evaluate those literals also increases.

For example, the Mutagenesis dataset contains three predicates –

- compound(CompoundID)
- atom(CompoundID, AtomID, Element, Quanta, Charge)
- bond(CompoundID, AtomID, AtomID, BondType).

Table 2.1 gives the number of literals considered for a rule generated by FOIL during one of the experiments:

```
active(A):-
  atom(A, B, C, D, E),
  atom(A, F, C, G, H),
  E > -0.121,
  H <= 0.011,
  H > -0.084,
  E <= -0.112.
```

Only predicate literals change the number of variables in the rule and thus affect the search space.

In constructing the same rule, RRR would evaluate two predicate literals and five test literals, for a total of seven (RRR would use an additional test to

encode the equality of the *element* field for the two atoms). Because the cost of evaluating literals changes as the rule grows, it cannot be said that RRR could produce 1506 / 8 ≈ 188 rules with roughly the same cost as FOIL produces one, but it can be observed that RRR can randomly generate a substantial number of rules without exceeding FOIL's computational cost. The advantage of RRR shows more clearly on high-arity predicates such as *conformation/168* in the Musk_1 dataset. After one *conformation* predicate is added to the rule, the number of *conformation* predicates to be examined escalates. For each of the 168 arguments in a *conformation* literal, FOIL must examine a *conformation* predicate using an existing variable for that argument or introducing a new one. This gives 2^{167} or roughly 1.87×10^{50} predicate literals for FOIL to evaluate. RRR has to evaluate only the predicate it randomly selects and its execution time is thus unaffected by this explosive increase in the number of possible literals.

2.2 Ruleset Production

A single randomly generated rule is not particularly useful for predicting the classes of unseen instances. RRR-SD (Random Relational Rules – Stochastic Dsicrimination) generates a number of random rules, then combines them to produce predictions using stochastic discrimination [38].

Section 2.2.1 describes how stochastic discrimination operates and how it affects rule generation, Section 2.2.2 examines the proportion of generated rules that are useful for stochastic discrimination and Section 2.2.3 discusses some implementation details that increase the efficiency of rule evaluation.

2.2.1 Stochastic Discrimination

Stochastic discrimination is a methodology for combining weak classifiers (in this case, random rules) to produce a complex classifier that can generalise to new data. Kleinberg's algorithm for stochastic discrimination requires that the weak classifiers be 'enriched', all covering a greater proportion of the target class than the other class, to differentiate between classes, and also that the set of classifiers be 'uniform', covering the training data as evenly as possible. These requirements are discussed in more detail below.

As the weak classifiers all cover a greater proportion of the target class (due to enrichment) than they do of the other class, instances of the target class

will, on average, have a higher coverage than those of the other class. Due to uniformity, instances that the randomly generated weak classifiers would tend to cover less frequently will instead receive coverage closer to the average. The result of this process as the number of weak classifiers increases is thus that the coverages for instances of both the target and non-target classes tend to approach normal distributions, with the mean for the target class greater than that for the non-target class, and decreasing variances for both means.

The threshold for class prediction for test instances is therefore the midpoint between the mean coverages for the two classes. More specifically, test instances are classified according to Equations 2.5-2.6 (for a two-class problem, classes P (the target class, for which the set is enriched) and N). The coverage on the test instance is compared to the midpoint between the overall mean coverages for each class, and if it is greater than that midpoint, it is classified as being of the target class. The mean coverage on the target class is expected to be higher than that on the other class, due to the enrichment requirement.

$$\text{Predict P if prop(Instance)} > \frac{\text{meanCoverage(P)} + \text{ meanCoverage(N)}}{2} \quad (2.5)$$

Where:
Instance = the test instance to be classified
prop(Instance) = the proportion of classifiers in the set that cover *Instance*
meanCoverage(Class) = the mean coverage of training instances of that class by the ruleset

$$\text{meanCoverage(P)} = \frac{\sum_{i=1}^{\text{size(P)}} \text{prop}(\text{P}_i)}{\text{size(P)}} \quad (2.6)$$

Where:
size(P) = the number of instances of class P
P_i = the i^{th} instance of class P

Enrichment

Enrichment is a rule-level quality: a rule is enriched for a particular class, if it covers a greater proportion of the instances of that class than it does of the instances of the other class.

$$\text{A rule is enriched if } \frac{\#\text{covered}_P}{\#\text{total}_P} > \frac{\#covered_N}{\#total_N} \tag{2.7}$$

For example, given a simple dataset containing 10 instances of class A and 20 instances of class B and a rule that covered 6 instances of class A and 11 instances of class B, the calculation would be as follows:

$$\text{The rule is enriched for class A if } \frac{6}{10} > \frac{11}{20} \tag{2.8}$$

As 0.6 is greater than 0.55, the rule is enriched for class A. In addition, to avoid overly specific rules, rules are required to cover more than one instance of the class for which they are enriched.

Uniformity

Uniformity is a ruleset-level quality - a uniform ruleset covers the training instances of a given class as evenly as possible. The current coverage of a ruleset affects the selection of new rules to be added, in a similar fashion to Boosting. RRR-SD defines uniformity as the standard deviation of the coverages of each instance of the target class in the training set, so that the best theoretically possible uniformity is 0, at which point each instance of the target class would be covered by exactly the same number of rules.

While determining if a particular rule is enriched is a simple mathematical calculation, determining whether adding that rule will satisfy the uniformity constraint is less straightforward. Several approaches to this problem were investigated before a satisfactory solution was found for RRR-SD.

- Fixed threshold – Setting a fixed threshold for the standard deviation, and rejecting rules that would bring the standard deviation over this threshold, either has very little effect on the uniformity (if the threshold is too high) or rejects a high proportion of rules, requiring large numbers of rules to be generated, and can result in non-termination, with no possible rule that can keep the uniformity under the threshold (if the threshold is too low).

- Annealing – An 'annealing' approach, where the threshold is initially set high and then decreased as rules are added, initially has the drawback of a high threshold, and later displays the high rejection rate (and concomitant requirement for many rules to be generated) and potential non-termination of a low threshold.

- Decreasing – Requiring each rule to keep the uniformity value the same or decrease it in order to be added has a similar drawback to a low threshold, in that the number of rules that will satisfy such a constraint grows smaller as more rules are added, and eventually non-termination results.

- Increasing Threshold – The reverse of the 'annealing' method, starting with a low threshold and raising the threshold gradually as rules fail, resetting it to the low value when a rule is added, avoids the non-termination problem, but is strongly affected by the size of increments and frequency of threshold raising. Similarly to the simple threshold method, if the threshold goes up quickly, uniformity is hardly affected, but if it goes up slowly, although the uniformity is improved, the number of rules generated to add a single rule is prohibitive.

All of the above methods for ensuring uniformity share the property that the number of rules that will be generated and discarded before one is added to the ruleset is unknown. Therefore, RRR-SD uses an alternative approach to ensure uniformity that guarantees progress within a fixed number of rules. A "maximum batch size" is selected, and as enriched rules are generated, they are added to the batch. When the batch reaches its maximum size, a subset of the rules in the batch will be added to the ruleset. This subset is determined by evaluating the resulting uniformity for adding each possible non-zero subset of rules, and selecting the subset resulting in the best value for uniformity. The remaining rules in the batch are discarded. This ensures that at least one rule in every batch will be added to the ruleset, while also maintaining an acceptable level of uniformity.

2.2.2 The number of rules evaluated

The number of rules RRR must examine to produce a ruleset of size N depends on the batch size and the training data. For any given dataset, there are a certain number of possible rules that can be generated. A certain proportion, p, of the generated rules will be enriched. If batches of size b are being used, then in the worst case one rule from each batch will be added to the ruleset. In this case bN enriched rules must be generated to complete the ruleset, and thus the upper bound on overall number of rules to be generated can be computed as $\frac{bN}{p}$. More than one rule per batch can be selected, so for most datasets less

than $\frac{bN}{p}$ random rules are generated to form a set of size N of enriched and reasonably uniform rules.

2.2.3 Implementation

This section discusses some implementation details which are essential for more efficient computation.

Algorithm 9 is a pseudo-code description of the RRR-SD algorithm, prior to the optimisations presented below. As the basic stochastic discrimination algorithm requires arbitrary selection of a target class (which could exclude usefully discriminatory rules that are enriched for the class not selected) RRR-SD, while similarly operating only on two-class problems, generates two sets of relational rules. Each of these rulesets will be enriched for one of the classes in the dataset, and contain at least a user-specified number of rules – due to the batch mechanic for uniformity, slightly more than the minimum number of rules may be produced.

Algorithm 9 Pseudocode for the RRR-SD algorithm

```
for Each class do
  while Number of rules for current class is less than the minimum do
    while Number of rules in batch is less than the minimum do
      Generate a rule
      if Rule is enriched for current class then
        Add rule to rule batch
      end if
    end while
    Calculate the most uniformity-preserving non-zero subset of rules in the
    rule batch
    Add those rules to the ruleset for the current class
  end while
end for
```

The following optimisations were implemented:

- Existence tests: Predicates that introduce variables that are never used in subsequent literals are 'existence tests' which either succeed or fail, without the need to enumerate all possible solutions. Consequently, such predicates will be treated like tests, and are cheap to evaluate.

  Example:

  ```
  rule(CompoundID):-
  ```

```
  atom(CompoundID, AtomID1, Element1, Quanta1, Charge1),
  Charge1 > 0.1,
  atom(CompoundID, AtomID2, Element2, Quanta2, Charge2).
```

In the example, the second *atom* literal introduces four new variables, none of which are used for tests, so the existence of an *atom* in the Compound is sufficient for this literal to succeed.

- Re-ordering and Separation: Literals in rules are re-ordered to minimise the branching factor encountered in evaluation of the rule. Because of the random rule generation process, re-ordering is both more useful and cheaper to compute than in systems like FOIL, as it has to be done only once, after the final literal has been included, and it has the potential to significantly speed up coverage computations.

  As predicates can introduce variables, they have the potential to increase branching, while tests have only two possible outcomes – success or failure. As the impact of the branching caused by a predicate literal, or the decrease in branches caused by a test literal, is greater the earlier in the rule it appears, the test literals should optimally appear as early in the rule as possible. The earliest the tests can appear in the rule is immediately after the predicate literal that introduces the variable or variables being tested, so tests are moved as close as possible to the predicates introducing their variables. Predicates are moved as far to the right as possible.

  For example, this rule:

```
rule(CompoundID):-
  atom(CompoundID, _, _, _, Charge1),
  atom(CompoundID, _, Element2, _, _),
  atom(CompoundID, _, _, Quanta3, _),
  Element2 = 'h',
  Quanta3 = '3',
  Charge1 > 0.1.
```

  would become:

```
rule(CompoundID):-
  atom(CompoundID, _, _, _, Charge1),
```

```
  Charge1 > 0.1,
  atom(CompoundID, _, Element2, _, _),
  Element2 = 'h',
  atom(CompoundID, _, _, Quanta3, _),
  Quanta3 = '3'.
```

For clarity, the Prolog syntax that describes unused variables as underscores is used here. The tests on Charge1 and Element2 have been moved to be adjacent to the predicates that introduced those variables.

To reduce the branching factor even further, some rules can be split into independent subrules, such that no subrule depends on variables introduced in another subrule. The rule is true for an instance (a particular binding of CompoundID, in the example below) if all of its subrules are true for that instance.

Example:

```
Subrule 1: rule(CompoundID):-
  atom(CompoundID, _, _, _, Charge1),
  Charge1 > 0.1.
Subrule 2: rule(CompoundID):-
  atom(CompoundID, _, Element2, _, _),
  Element2 = 'h'
Subrule 3: rule(CompoundID):-
  atom(CompoundID, _, _, Quanta3, _),
  Quanta3 = '3'.
```

- Enrichment: Each rule belongs to exactly one of three disjoint sets - a rule is enriched for one class, a rule is enriched for the other class, or a rule is enriched for neither class (usually because it covers either no instances or all instances).

  Rather than generating rules until the ruleset enriched for one class is complete, then repeating the process for the other class, RRR-SD generates rules and adds each enriched rule to its appropriate ruleset. By interleaving the rule generation process in this way, no enriched rule will be wasted.

- Negation: If enough rules have already been generated for one class, additional enriched rules for that class are irrelevant. However, inverting

the coverage results for a rule enriched for one class will yield the coverage for a rule enriched for the other class in a binary class setting, and therefore such rules do not have to be discarded. This inversion is accomplished by treating any instances not covered by the rule as being covered, and any instances covered by the rule as being not covered, with the result that if the rule was enriched for one of the two classes, the negation must now be enriched for the other.

For most datasets it was found that the distribution of enriched random rules was slightly skewed with a larger number of enriched rules being generated for one class than the other. Thus, negation helped to reduce redundant rule creation and therefore also to reduce computing times.

- Prefixes: Every prefix of a random rule is another random rule – some may be enriched, some may not. As, in the course of evaluating the full rule, all the prefixes are also evaluated, there is very little computational cost in making use of this additional information. RRR-SD selects the 'most enriched' prefix for each rule – the prefix for which the ratio between the proportions covered of instances of each class by the rule is the greatest. This also has the advantage that, even if the full-length rule is not enriched, one of its prefixes may be, increasing the proportion of possible useful rules.

- ID Elements: When generating predicate literals, for appropriate datasets, the single variable that the predicates must already have bound can be required to be their 'ID element', with the remaining variables in the predicate being newly introduced. The 'ID element' is the argument in the predicate that identifies which instance it belongs to. For example, in the Mutagenesis predicate $atom(CompoundID, AtomID, Element, Quanta, Charge)$, the CompoundID argument is the 'ID element'. This reduces the amount of evaluation required, as when determining the mutagenicity of a particular compound, only atoms of that compound will be considered when determining the mutagenicity of the compound. For datasets where the instances are not interdependent, it does not make sense to generate rules that predict the class of one instance based on properties of another instance. (If the 'ID element' were not required to be bound, rules could be generated containing Atom and Bond predicates that could be instantiated to come from different compounds).

Algorithm 10 is a pseudo-code description of the optimised RRR-SD algorithm.

Algorithm 10 Pseudocode for the optimised RRR-SD algorithm

```
while Number of rules for either class is less than the minimum do
  while Number of rules in batch for either class is less than the minimum
  do
    Generate a rule
    Select an enriched prefix of that rule (including the full-length rule)
    if Rule is enriched for class A then
      if Rule batch for class A is not yet full then
        Add rule to rule batch for class A
      else
        Negate rule and add it to rule batch for class B
      end if
    else if Rule is enriched for class B then
      if Rule batch for class B is not yet full then
        Add rule to rule batch for class B
      else
        Negate rule and add it to rule batch for class A
      end if
    end if
  end while
  Calculate the most uniformity-preserving non-zero subset of rules in each
  rule batch
  Add those rules to their corresponding rulesets
end while
```

2.3 Ruleset Evaluation

RRR-SD produces two sets of rules, one enriched for each class, as the rules enriched for one class may be quite different from those enriched for the other. Because the rulesets are constructed independently, there is no guarantee that the coverage distribution of one ruleset will be mirrored in the other. This means that the raw proportions of rules in each ruleset that cover a test instance cannot be directly compared to determine a prediction. Therefore a transformation must be applied to produce compatible predictors for both rulesets. Two ratios which make the proportions comparable have shown reasonable performance – each ruleset's average coverage across all training instances (AC) and the per-ruleset mean of each class's average coverage on the training instances (MAC).

Table 2.2: Example dataset

Class	Number of Instances	Mean coverage by R_A	Mean coverage by R_B
A	90	0.3	0.4
B	10	0.1	0.6
Overall	100	0.28	0.42

$$\text{AC}_{ruleset}(\text{Instance}) = \frac{p(\text{Instance})}{\text{mean coverage of training instances}} \tag{2.9}$$

$$\text{MAC}_{ruleset}(\text{Instance}) = \frac{p(\text{Instance})}{0.5(\text{mean\_coverage}_{class\ A} + \text{mean\_coverage}_{class\ B})} \tag{2.10}$$

Where:
p(Instance) = proportion of rules in ruleset that cover Instance

Both for AC and MAC, the final classification decision is made in favour of the class predicted by the ruleset with the maximal value for a given test example.

To demonstrate AC and MAC, consider a hypothetical dataset with 100 instances, 90 being of class A and 10 being of class B, and two rulesets R_A (enriched for class A) and R_B (enriched for class B), summarised in Table 2.2. R_A has a mean coverage on class A instances of 0.3 and a mean coverage on class B instances of 0.1, for an overall average coverage of 0.28. R_B has a mean coverage on class A of 0.4 and a mean coverage on class B of 0.6, for an overall average coverage of 0.42. The average of the mean coverages is $((0.1 + 0.3) \div 2 =)$ 0.2 for R_A and $((0.4 + 0.6) \div 2 =)$ 0.5 for R_B. For the purposes of evaluating a test instance, $Test$, this gives:

$$AC_{R_A} = \frac{coverage_{R_A}(Test)}{0.28} \text{ and } AC_{R_B} = \frac{coverage_{R_B}(Test)}{0.42} \tag{2.11}$$

$$MAC_{R_A} = \frac{coverage_{R_A}(Test)}{0.2} \text{ and } MAC_{R_B} = \frac{coverage_{R_B}(Test)}{0.5} \tag{2.12}$$

If $AC_{R_A}(Test) > AC_{R_B}(Test)$ then the AC method classifies Test as being

Table 2.3: Classification decisions

Condition	AC	MAC	Raw
$c \geq 1$	A	A	A
$1 > c \geq \frac{2}{3}$	A	A	B
$\frac{2}{3} > c \geq \frac{2}{5}$	B	A	B
$\frac{2}{5} > c$	B	B	B

Table 2.4: Example classifications by RRR-SD on hypothetical data

			Classification		
Coverage_{R_A}	Coverage_{R_B}	c	AC	MAC	Raw
0.1	0.1	1.0	A	A	=
0.1	0.3	0.33	B	B	B
0.2	0.1	2.0	A	A	A
0.2	0.4	0.5	B	A	B
0.3	0.4	0.75	A	A	B
1.0	1.0	1.0	A	A	=

of class A, and otherwise as being of class B. Similarly, if $MAC_{R_A}(Test) > MAC_{R_B}(Test)$ then the MAC method classifies Test as being of class A, and otherwise as being of class B.

These formulae allow the results in Table 2.3 to be calculated for the AC, the MAC and the unadjusted coverage (Raw, which is retained to demonstrate that the raw proportions are not useful when directly compared). If we let $c(Test) = \frac{Coverage_{R_A}(Test)}{Coverage_{R_B}(Test)}$ then rearranging Equations 2.11 and 2.12 gives the decision boundaries in Table 2.3 which are illustrated with examples in Table 2.4.

For this data, when a test instance has equal coverage by both rulesets, it is classified as an A. Because R_A's average coverage is lower than R_B's, numerically equal coverage is regarded by the algorithm as R_A having unusually high coverage. This leads to a prediction of A, the class for which the rules in R_A are enriched. AC and MAC disagree on the classification for an instance when c is between the quotient of the average coverages ($\frac{0.28}{0.42} = \frac{2}{3}$) and the ratio of the average of the mean coverages ($\frac{0.2}{0.5} = \frac{2}{5}$) – thus if the classes were present in equal proportions, the AC and MAC would be equal. The AC is affected by the proportions of the classes in the dataset – as the difference in the class proportions increases, so does the range in which AC and MAC produce differing classifications, as the decision point for AC moves closer to the majority class. This narrows the range in which the majority class (A in this example) is predicted. In the fifth row of Table 2.4, the hypothetical instance had the

average coverage for a class A instance from both rulesets (0.3, 0.4), but the raw coverage predicted class B – a possibility that confirms the unsuitability of unmodified coverage for comparisons.

2.4 Experiments and results

An evaluation of RRR on several datasets has been conducted, using ten stratified ten-fold cross-validation runs, on the following standard ILP datasets: Mutagenesis (with and without regression-unfriendly instances) [76], Musk1 [20], Carcinogenesis [75], and Diterpenes [23]. Mutagenesis and Carcinogenesis were limited to low-level structural information as represented by *atoms* and *bonds*; additional propositional information such as global properties *lumo* or $logP$, or predefined functional groups, was deliberately excluded: they are known to improve classification accuracy significantly, thereby potentially masking the relational performance of the investigated algorithms. More detailed information on these datasets is given in Chapter 3.

The current implementation of RRR-SD is limited to two classes, so the Diterpenes dataset was transformed into three two-class versions by using all pairwise combinations of the three largest classes called 3, 52 and 54 – $\text{Diterpenes}_{54,3}$, $\text{Diterpenes}_{52,3}$ and $\text{Diterpenes}_{52,54}$.

The minimum ruleset size was set to 500 (for an overall total of at least 1,000 rules in each run). The maximum number of literals per rule for RRR was set to six. (This is comparable to other clause contruction systems – RSD [80] sets a maximum length of five for the Mutagenesis problem, while ALEPH [73] defaults to four.) Greater maximum rule lengths were examined for RRR, but caused increases in runtime without a significant increase in accuracy, so for these and later experiments, the maximum rule length for RRR is six.

RRR's results are compared to FOIL 6.4 on these datasets using the default options. FOIL 6.4 fails to produce rules on Musk1, but Ray and Craven [66] report results gained from a version of FOIL modified to run on that dataset, and their results (marked by *) are used for comparison (no standard deviations were given, so the corresponding result in Figure 2.2 has no error bar).

The results of this evaluation are shown in Tables 2.5 and 2.6 and graphically in Figures 2.1 and 2.2. The confidence of predictions (for each instance) used for computing AUC for RRR-SD was calculated by taking the ratio between the comparison method value for each ruleset, as shown in Equation

2.13.

$$\text{Confidence (T}_i\text{, MAC)} = \frac{MAC_0(T_i)}{MAC_1(T_i)} \tag{2.13}$$

Where:
T_i = test instance
MAC_0(i) = MAC for instance i from ruleset enriched for class 0
MAC_1(i) = MAC for instance i from ruleset enriched for class 1

Although AC and MAC produce different accuracies on a given dataset, they produce the same AUC, because the AC and MAC values are, for any given ruleset, related by a particular ratio, as shown in Equation 2.14. The *meancov* values are calculated from the training data and are the same for all test instances, so that while c' and c'' are constant (although usually different) for any given ruleset, the value dependent on the test instance coverage, r, is common to both the AC and the MAC. Thus, while they may produce different predictions, the ordering of the confidence of those predictions is the same.

$$\begin{aligned}
&\frac{MAC_0(i)}{MAC_1(i)} && \frac{AC_0(i)}{AC_1(i)} \\
&= \frac{\frac{2\times cov_0(i)}{meancov_{0,0}+meancov_{0,1}}}{\frac{2\times cov_1(i)}{meancov_{1,0}+meancov_{1,1}}} && = \frac{\frac{cov_0(i)}{meancov_0}}{\frac{cov_1(i)}{meancov_1}} \\
&= \frac{2cov_0(i) \times (meancov_{1,0} + meancov_{1,1})}{2cov_1(i) \times (meancov_{0,0} + meancov_{0,1})} && = \frac{cov_0(i) \times meancov_1}{cov_1(i) \times meancov_0} \\
&= \frac{cov_0(i)}{cov_1(i)} \times \frac{(meancov_{1,0} + meancov_{1,1})}{(meancov_{0,0} + meancov_{0,1})} && = \frac{cov_0(i)}{cov_1(i)} \times \frac{meancov_1}{meancov_0} \\
&= r \times c'' && = r \times c'
\end{aligned} \tag{2.14}$$

Where:
$MAC_x(i)$ = the MAC value from the ruleset enriched for class x for instance i
$AC_x(i)$ = the AC value from the ruleset enriched for class x for instance i
$cov_x(i)$ = the coverage by the ruleset enriched for class x for instance i
$meancov_x$ = the mean coverage by the ruleset enriched for class x across all training instances
$meancov_{x,y}$ = the mean coverage by the ruleset enriched for class x across training instances of class y
$r = \frac{cov_0(i)}{cov_1(i)}$

Table 2.5: Accuracy for RRR-SD and FOIL

Dataset	FOIL	RRR-SD AC	RRR-SD MAC	
Carcinogenesis	46.8±7.4	58.0±7.8	58.5±8.1	↗
$\text{Diterpenes}_{52,3}$	93.6±3.7	88.4±4.4	90.2±4.1	
$\text{Diterpenes}_{52,54}$	87.0±4.7	84.0±4.0	85.6±3.2	
$\text{Diterpenes}_{54,3}$	94.9±2.8	93.6±4.1	93.6±4.1	
Musk_1	*	83.9±13.1	83.6±13.2	
Mutagenesis_{All}	69.2±10.3	75.8±8.6	76.6±8.5	
Mutagenesis_{RF}	73.8±10.1	77.5±9.3	80.2±9.7	

Table 2.6: AUC for FOIL and RRR-SD

Dataset	FOIL	RRR-SD AC/MAC	
Carcinogenesis	0.496±0.075	0.654±0.088	↗
$\text{Diterpenes}_{52,3}$	0.947±0.034	0.965±0.023	
$\text{Diterpenes}_{52,54}$	0.891±0.046	0.943±0.023	↗
$\text{Diterpenes}_{54,3}$	0.962±0.025	0.979±0.019	
Musk_1	0.719*	0.898±0.122	↗
Mutagenesis_{All}	0.725±0.097	0.779±0.101	
Mutagenesis_{RF}	0.775±0.094	0.842±0.108	

Here and in subsequent chapters, the measures of variance indicated by ± in tables and error bars in figures are standard deviations across all folds. RRR-SD can be seen to produce more accurate results than FOIL on the Mutagenesis_{RF}, Mutagenesis_{All} and Carcinogenesis datasets, while performing slightly worse on the Diterpenes datasets. This difference is significant at the 95% level (by corrected unpaired t-test, as different folds were used by FOIL and RRR-SD) for the Carcinogenesis dataset. The corrected t-test (as described in [55]) is used for significance testing as the standard t-test has been shown to produce inflated Type I error. However, when AUC is examined, RRR-SD performs better than FOIL on Carcinogenesis and $\text{Diterpenes}_{52,54}$ (again by corrected t-test, with 95% significance). Although [66] does not give a standard deviation for their Musk_1 AUC result, for any standard deviation up to 0.22 (substantially larger than the standard deviation produced by RRR-SD) RRR-SD performs better with 95% significance. Significance is indicated in Tables 2.5 and 2.6 by ↗ where RRR-SD's result is significantly higher than FOIL. One factor contribut-

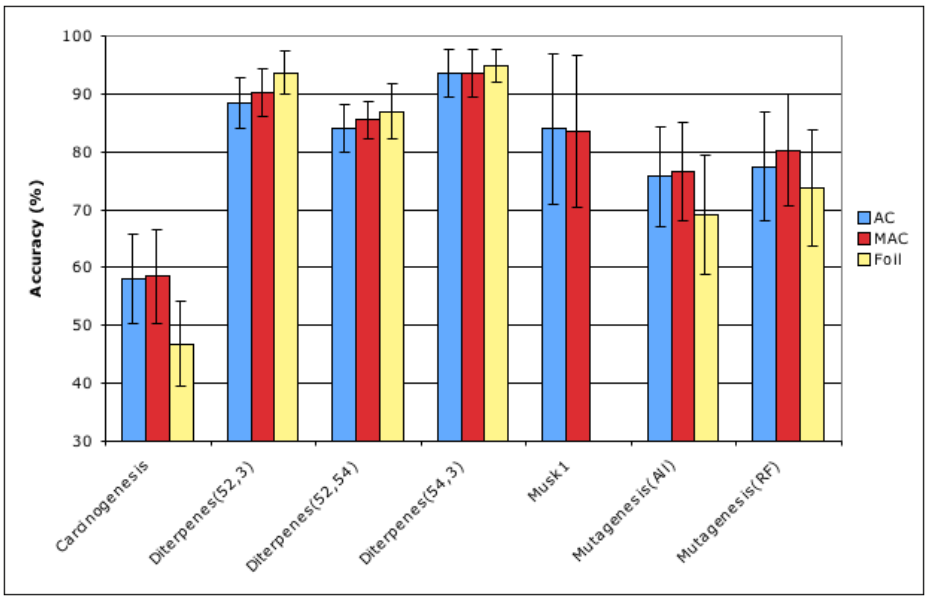


Figure 2.1: Accuracy for RRR-SD and FOIL

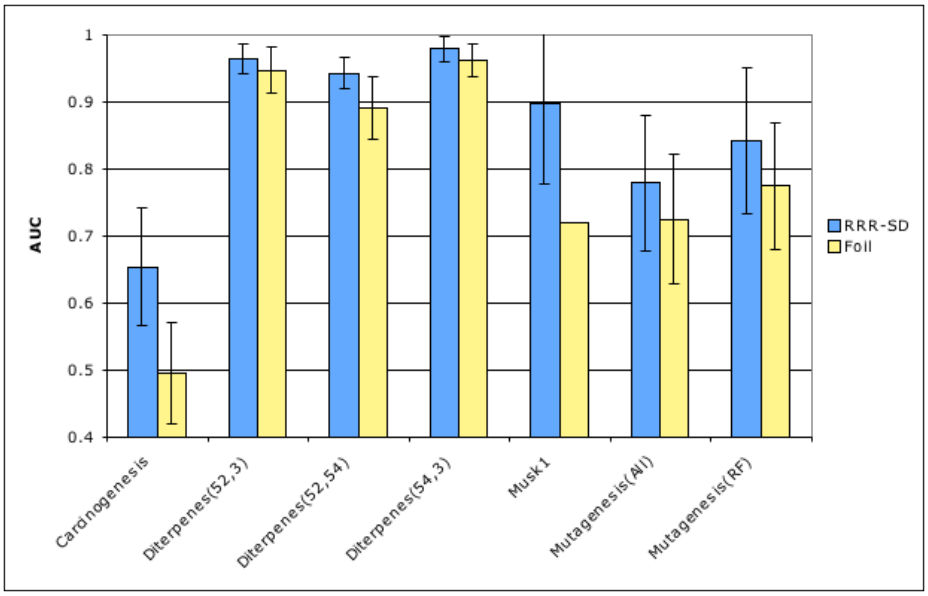


Figure 2.2: AUC for RRR-SD and FOIL

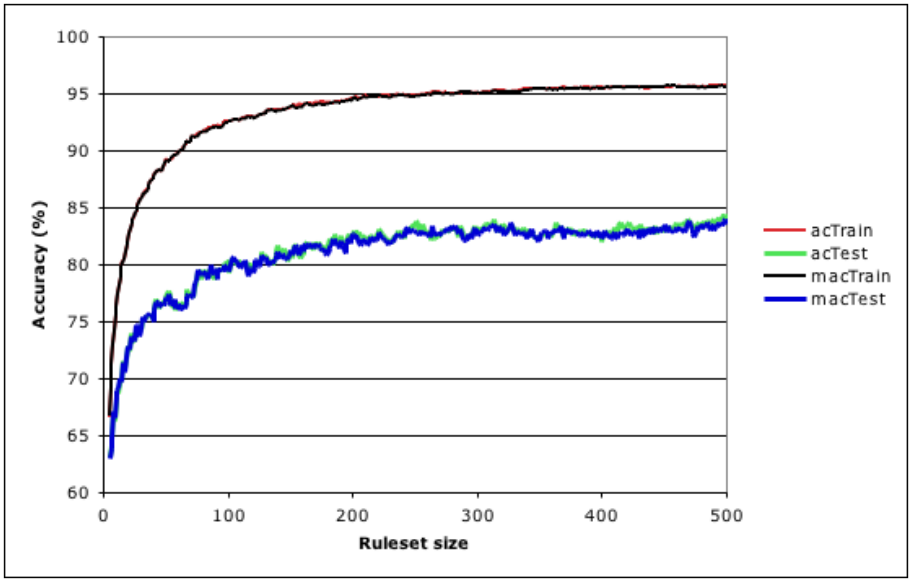


Figure 2.3: Test and Training Accuracy for RRR-SD on Musk_1

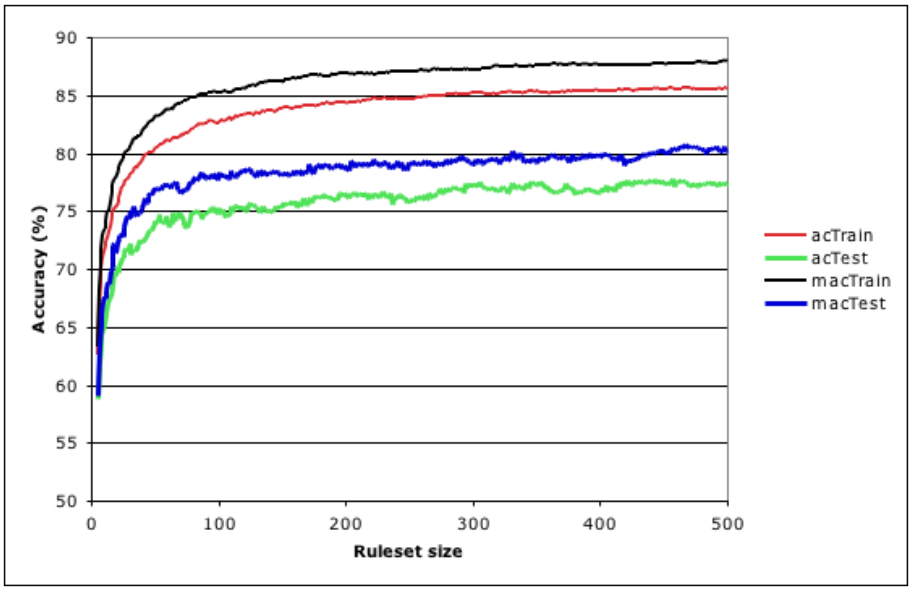


Figure 2.4: Test and Training Accuracy for RRR-SD on Mutagenesis_{RF}

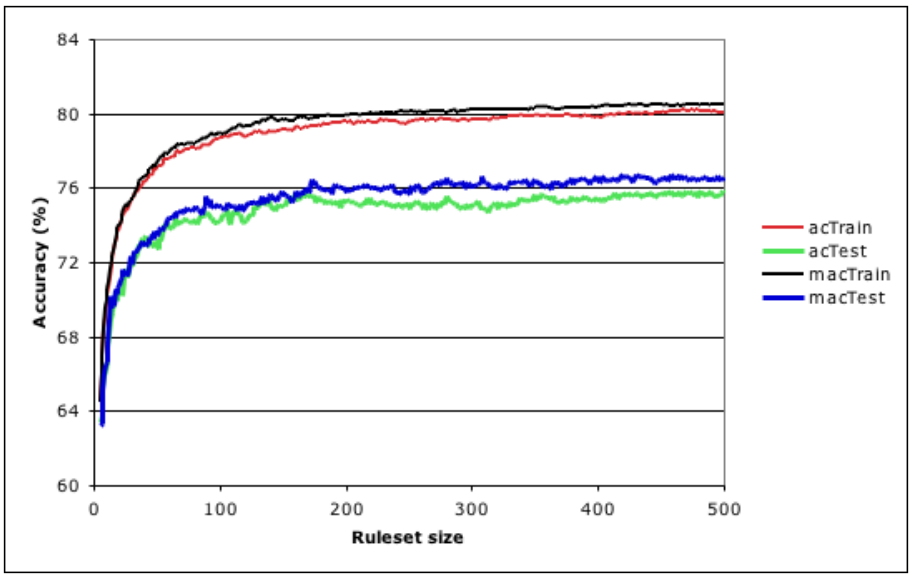


Figure 2.5: Test and Training Accuracy for RRR-SD on Mutagenesis$_{All}$

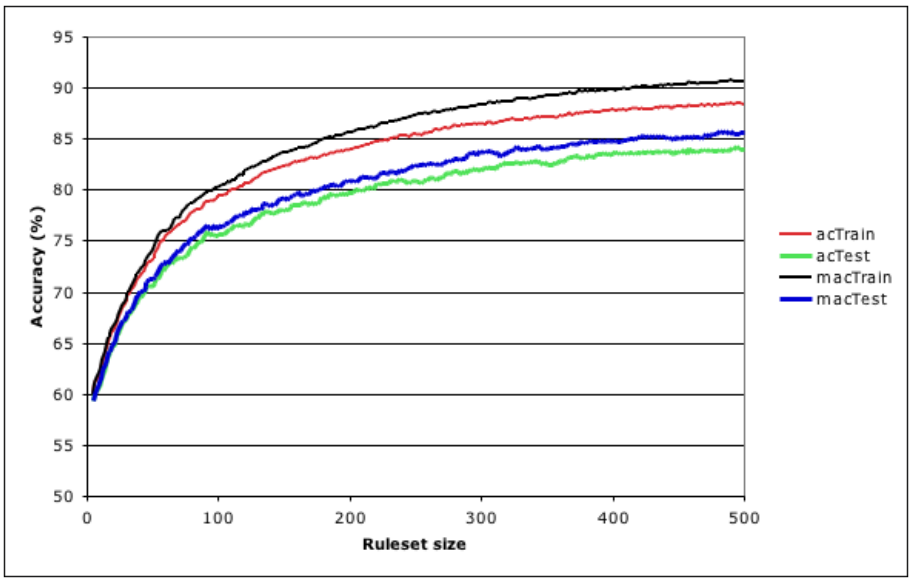


Figure 2.6: Test and Training Accuracy for RRR-SD on Diterpenes$_{52,54}$

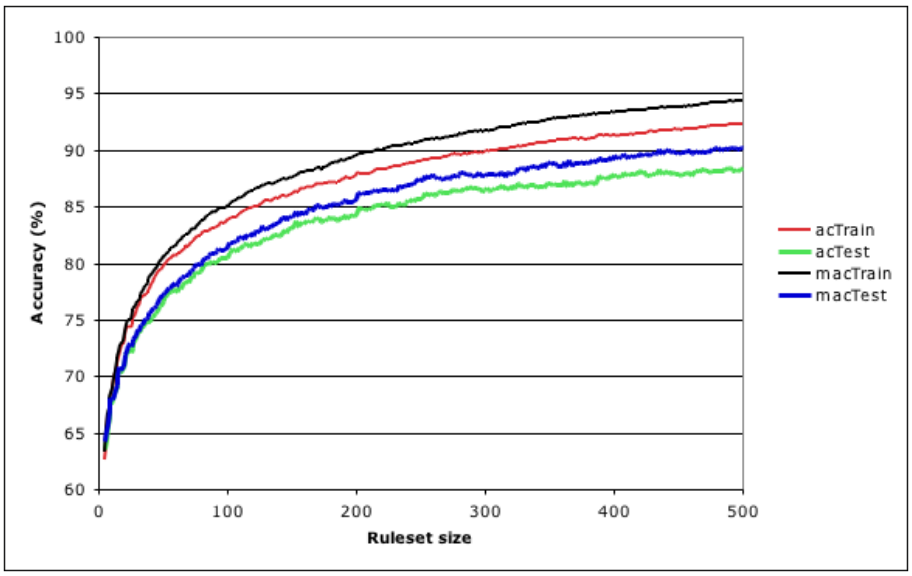


Figure 2.7: Test and Training Accuracy for RRR-SD on $\text{Diterpenes}_{52,3}$

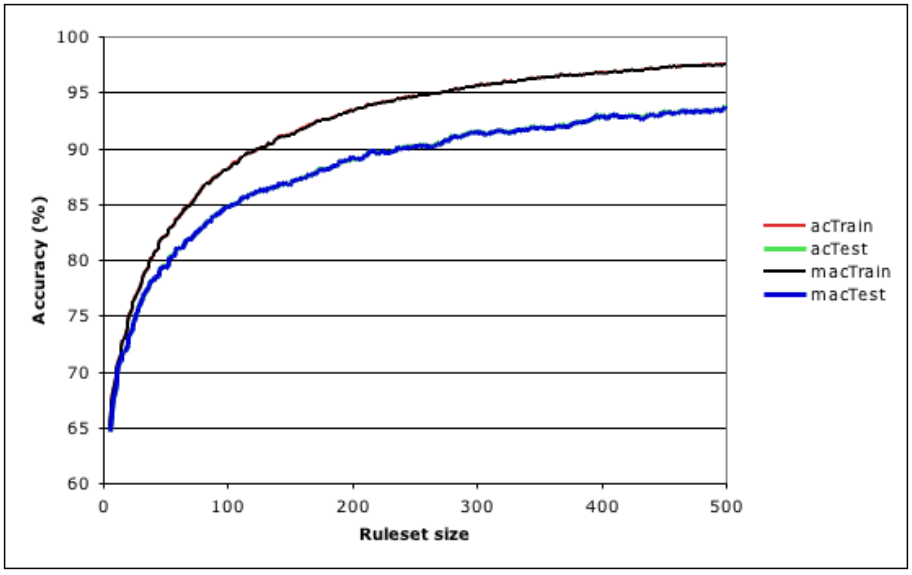


Figure 2.8: Test and Training Accuracy for RRR-SD on $\text{Diterpenes}_{54,3}$
Note that the acTrain-macTrain and acTest-macTest pairs virtually overlap, as the classes are present in almost equal proportions causing AC and MAC to be very similar

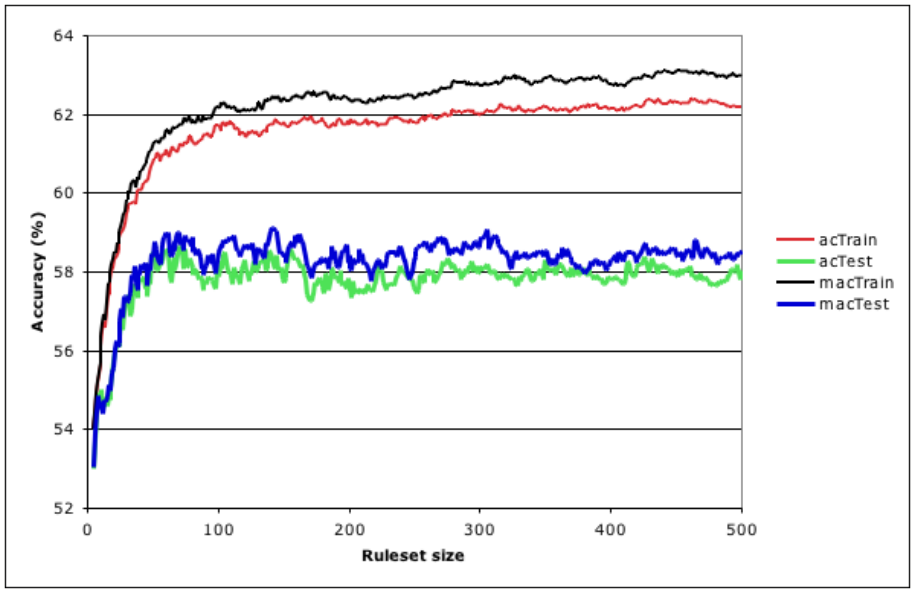


Figure 2.9: Test and Training Accuracy for RRR-SD on Carcinogenesis

ing to this may be that a 'negative' prediction for FOIL occurs when none of the rules in its ruleset are used, meaning that all negative predictions are made with equal confidence.

Experiments were also conducted that investigated varying the ruleset size. Results for all seven datasets are shown in Figures 2.3 through 2.9. When the two classes contain equal numbers of training instances, AC and MAC are equal, so for datasets where this very nearly holds (Musk_1, $\text{Diterpenes}_{54,3}$) the results are extremely similar. On datasets where the disparity in class sizes is larger (Mutagenesis_{RF} and Mutagenesis_{All}, for example, where the ratio between the classes is approximately 2:1) MAC can be seen to outperform AC – as could be expected, given that the calculation of MAC takes ruleset coverage on classes into account, while AC does not. The results from the larger ruleset sizes demonstrate that, on all the datasets except Carcinogenesis, accuracy eventually levels out, but does not deteriorate or overfit when generating more than enough rules.

2.5 Timing

The time taken for RRR-SD (using the same parameters as for the experiments in Section 2.4) and FOIL to perform ten ten-fold cross-validation runs was also measured, and the mean times taken for ten-fold cross-validation runs are given in Table 2.7.

Table 2.7: Average time taken for ten-fold cross-validation for RRR-SD and FOIL

	time taken (minutes)	
Dataset	RRR-SD	FOIL
Carcinogenesis	318.3	2374.7
$\text{Diterpenes}_{52,3}$	115.9	322.2
$\text{Diterpenes}_{52,54}$	105.9	586.3
$\text{Diterpenes}_{54,3}$	103.6	27.1
Musk_1	30.1	-
Mutagenesis_{All}	90.7	1239.4
Mutagenesis_{RF}	78.8	715.0

On most datasets RRR-SD is shown to be substantially faster. The exception is $\text{Diterpenes}_{54,3}$, on which FOIL's greedy search is quick to find several high-coverage rules.

2.6 Summary

This chapter has introduced RRR, an algorithm for generating random relational rules, and RRR-SD, an algorithm for classification using RRR based on the framework of stochastic discrimination. Two different methods – AC and MAC – for aggregating rule predictions have been described. Both the AC and MAC method produce good predictive performance results, with MAC consistently being either equal to or better than AC on all the datasets tested. The difference is explained by MAC's better ability for coping with imbalanced classes.

RRR-SD demonstrates that it is possible for ensembles of randomly generated weak rules to be competitive with those produced by FOIL's heuristic search, while also being generated substantially more quickly in most cases.

Chapter 3

Propositionalisation

In this chapter, RRR-P (Randomised Relational Rules – Propositionalisation), a system that propositionalises data using the rules generated by RRR and applies propositional learning algorithms to the results, is described. The results of applying propositional learning algorithms to the propositionalised datasets are reported, and compared to other relational learning methods. Section 3.1 discusses propositionalisation in general and Section 3.2 describes how the output of the RRR algorithm is used to propositionalise data. Section 3.3 gives the results obtained by applying standard machine learning algorithms to the propositionalised data and compares the results with those obtained by other methods, and Section 3.4 contains a discussion of those results.

3.1 Propositionalisation

Propositional learning algorithms represent instances as single objects with values for a given set of attributes, which can make it difficult to represent relationships between objects. Relational learning algorithms employ more sophisticated concept descriptions to overcome this limitation and allow relationships to be represented explicitly so that they can be used in learning. However, this increased expressivity also results in greater computational complexity.

Propositionalisation is the application of a transformation that converts relational data into propositional data. This can be advantageous as, assuming the propositional representation preserves sufficient information from the original relational data, efficient propositional classification algorithms can be applied. The goal of propositionalisation of relational data is to achieve the

efficiency of standard machine learning algorithms, while preserving the relational information encoded in the data.

To give some context for the propositionalisation algorithm introduced in this chapter, three previous approaches to propositionalisation are briefly discussed in Sections 3.1.1-3.1.3.

3.1.1 RSD

The Relational Subgroup Discovery (RSD) algorithm [80] takes a three-step approach to propositionalisation. RSD first computes all possible conjunctions of first-order literals that could form admissible features. In this step no variables are instantiated – the set of conjunctions of literals that is produced is constant-free, describing the structure of possible features. It also obeys a connectivity requirement – features that can be decomposed into two or more separate features are not admissible.

In the second step, selected variables in the features constructed in the first step are bound. The user can specify which variables should be instantiated in this way. For each initial feature, a number of features are generated, each with a different combination of the possible bindings of the variables to be instantiated in that feature. Of these generated features, those true for at least a certain (user-specified) minimum number of instances are retained, and the rest discarded. Additionally, no feature may have the same Boolean value across all instances (i.e. be always true or always false), and no two features may have identical Boolean values across all instances (one is arbitrarily chosen to represent the equivalent features).

In the third step, each resulting feature is converted into a Boolean attribute, based on the truth value of the feature as applied to each instance in the data.

3.1.2 SINUS

SINUS [41] constructs features left-to-right, beginning with a single literal describing an individual. For each new literal, structural (introducing new variables) and property (tests some property of an existing variable) predicates are considered. The maximum number of literals to produce and the maximum number of variables in a feature can be specified by the user.

Variable reuse is an option with a significant impact on the generated features. If no reuse is allowed (only one property predicate is allowed for each

introduced variable) the resulting feature set is smaller, but contains only simple features. If reuse is allowed without constraint, redundant and/or contradictory features may be generated, and the feature set is much larger. A compromise allows only structural predicates to reuse variables. This allows for the construction of more complex features, but with less redundant features than allowing full reuse would produce.

Once the features are produced, the feature set can be filtered by applying tests for feature quality, and removing irrelevant features. The feature set can then be converted into a Boolean representation, just as with RSD. However, SINUS is also able to translate models produced by some propositional learners back into Prolog form.

3.1.3 RELAGGS

RELAGGS [43] (*Rel*ational *agg*regation*s*) makes use of the relational database technique of aggregation to summarise information from the non-target relations with respect to each instance in the target relation. The identifiers of the instances are propagated to the non-target relations via foreign key relationships using database joins, and the information belonging to each instance in those relations is then combined into a single row using aggregates.

Numeric attributes can be described with information such as minimum, maximum, average and sum, and even more sophisticated derived information such as standard deviations, ranges and quantiles. Nominal attributes can be described by their cardinality. One advantage of this approach is that the propositionalisations thus generated are not limited to Boolean attributes.

3.2 Propositionalisation using RRR

The RRR algorithm can be used as a tool for propositionalisation. Each relational rule it produces can be transformed into Boolean features for each instance, where the feature is 'true' if the rule covers the instance or 'false' if it does not (just as for RSD, for example). A very simple example of this is shown in Table 3.1, for a dataset consisting of the integers from one to ten, in which it is assumed that each rule corresponds to a column, and begins with the literal $number(X)$, followed by the literal heading the column. The instance '3', for example, would have the propositional representation "t, t, f".

Table 3.1: A simple example of propositionalisation

Instance	X = 3	X < 9	X >= 7
1	f	t	f
2	f	t	f
3	t	t	f
4	f	t	f
5	f	t	f
6	f	t	f
7	f	t	t
8	f	t	t
9	f	f	t
10	f	f	t

Pseudocode for the 'Random Relational Rules – Propositionalised' algorithm (hereafter RRR-P) is given in Algorithm 11.

Algorithm 11 Pseudocode for the RRR-P algorithm

```
while Number of rules in ruleset for either class is less than the minimum
do
  while Number of rules in batch for either class is less than the minimum
  do
    Generate a Rule
    if Rule is acceptable with regard to coverage constraints (enrichment)
    then
      Add Rule to appropriate rule batch
    end if
  end while
  Calculate the most uniformity-preserving non-empty subset of rules in
  each rule batch
  Add those rules to the ruleset
end while
Use ruleset to generate Boolean-valued propositional dataset
Apply any propositional classification algorithm
```

3.3 Experiments

For each of the experiments using RRR-SD in Chapter 2, a propositionalisation was generated from the rules produced. For each fold in each of the ten ten-fold cross-validation runs, the rules generated on the training data were used to propositionalise that data, and then also used to propositionalise the

test set. The resulting train-test sets were then evaluated using two standard machine learning algorithms from Weka [81] - Smo [60], a support vector machine (the SVMs used in this and later chapters are all linear) and Logistic [47], a logistic regression algorithm. As these algorithms are sensitive to their Complexity and Ridge parameters, respectively, five values were used for each, and if the accuracy was highest at either the highest or lowest value, the range was extended. The mean accuracy for each parameter value was calculated across the ten ten-fold cross-validation runs, and the highest of these results is compared to previous results from the literature. This could be viewed as optimistic – however, reported results in the literature are also generally the result of parameter tuning and therefore similarly optimistic. In addition to this, where the reported results do vary according to parameter settings, only the highest result is taken for comparison to the results for Rrr. Standard deviations are included in figures where the applicable values were available in the literature.

3.3.1 Mutagenesis

The Mutagenesis dataset [76] is a set of 230 nitroaromatic compounds. These compounds occur in automobile exhaust and also in the production of many industrial compounds. Nitroaromatic compounds that are extremely mutagenic have been found to be carcinogenic and damage DNA. The ability to determine mutagenicity from molecular structure is thus of interest to various industries, including the pharmaceutical industry – in producing less hazardous compounds, or for situations where standard mutagenicity tests are inapplicable.

The dataset consists of three relations:

- Compound: compound(CompoundID, Class)
- Atom: atom(CompoundID, AtomID, Element, QuantaType, Charge)
- Bond: bond(CompoundID, AtomID, AtomID, QuantaType)

Where:
CompoundID is the unique identifier for the compound
Class is either *active* or *inactive*
AtomID is the unique identifier for the atom
Element is the chemical element of the atom

Table 3.2: Distribution of Mutagenesis instances across the regression-friendly and -unfriendly subsets

Compounds	Active	Inactive	Total	Majority
Regression-friendly	125	63	188	0.665
Regression-unfriendly	13	29	42	0.690
Total	138	92	230	0.6

QuantaType is the type of the atom or bond, as assigned by the molecular modelling package QUANTA
Charge is the partial charge of the atom

The Mutagenesis dataset has been split into 'regression-friendly' and 'regression-unfriendly' subsets [17]. The split is based on a regression equation, derived from four propositional attributes determined by expert inspection, that correctly classifies a high proportion of the 'regression-friendly' instances. The distribution of active and inactive instances between the two subsets is shown in Table 3.2.

Increasing levels of background information for the Mutagenesis dataset have been described [74] – some of these include the propositional attributes already mentioned. For experiments with RRR, the 'B0' level of background knowledge (as described in [74]) is used – only the descriptions of atoms and bonds, and numeric inequalities are included. The increased levels of background information were not used as they are known to improve classification accuracy significantly, thereby potentially masking the relational performance of the algorithm.

Mutagenesis$_{RF}$

Figures 3.1 and 3.2 show the accuracy obtained by RRR-P on the 188-instance regression-friendly subset of the Mutagenesis data (hereafter Mutagenesis$_{RF}$). Both standard machine learning algorithms show clear peaks – Complexity 0.1 for SMO and Ridge 10 for Logistic – with the result for SMO being slightly higher than that for Logistic.

The comparison between the result for RRR-P and other algorithms is shown in Figure 3.3. The results for FOIL and RRR-SD from Chapter 2 have been included. Published results have also been included for Relational Kernels [82], the rule learner RIPPERMI [14], the decision tree inducer TILDE [6],

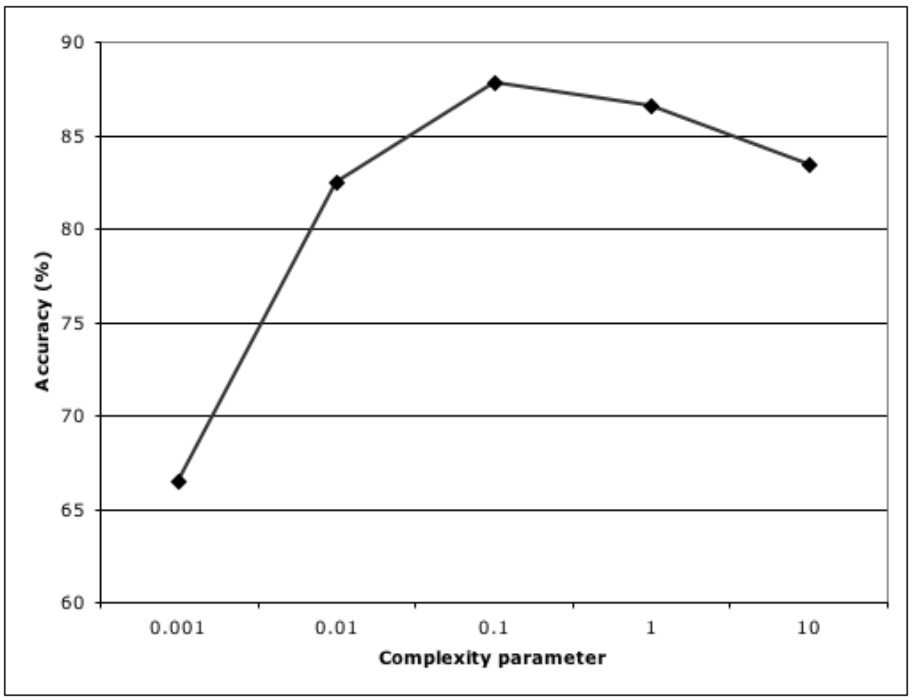


Figure 3.1: Accuracy for RRR-P on Mutagenesis$_{RF}$, using SMO

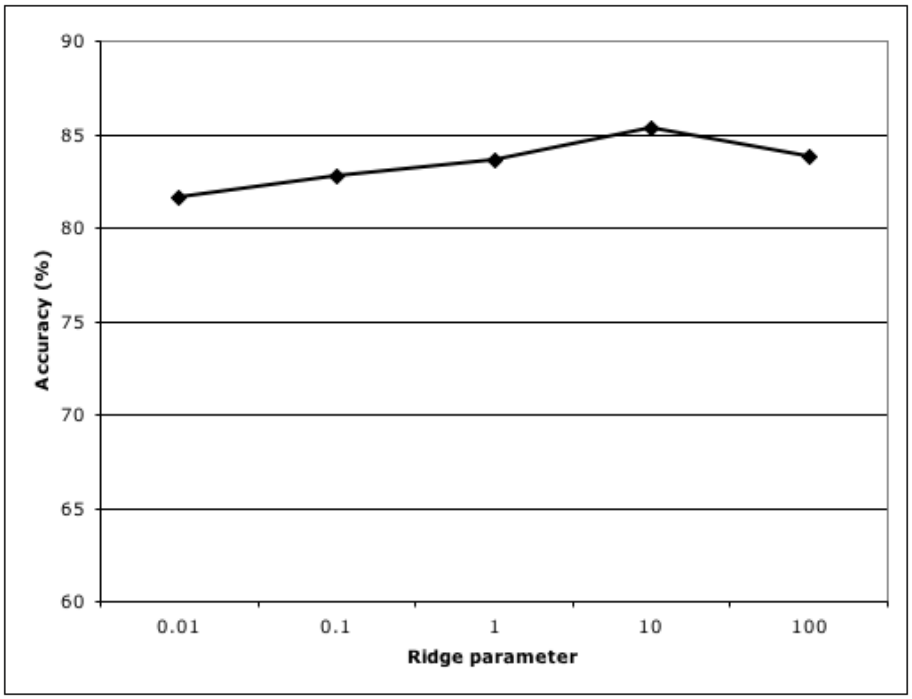


Figure 3.2: Accuracy for RRR-P on Mutagenesis$_{RF}$, using Logistic

the ILP systems PROGOL [54] and ALEPH [73], the ALEPH-based algorithm Random Seeds [49], NFOIL and TFOIL [44] (adding naive Bayes methods to FOIL), KFOIL [45] (adding kernel methods to FOIL) and 1BC2 [26], a first-order upgrade to naive Bayes. Results for RIPPERMI and TILDE were obtained from [15], for ALEPH and KFOIL from [45], for NFOIL and TFOIL from [44], and for PROGOL from [74]. Of the algorithms compared, only Random Seeds performs better than RRR-P(SMO). RRR-P(Log) performs slightly worse than RRR-P(SMO), with similar accuracy to Relational Kernels. Both RRR-P methods show substantial increases in accuracy over RRR-SD.

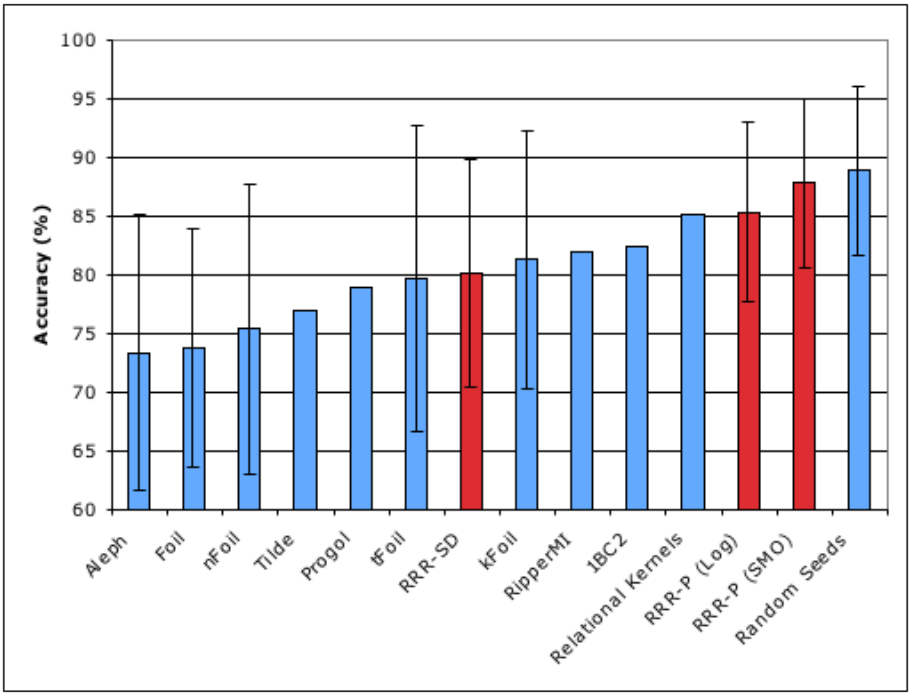


Figure 3.3: Accuracy for various algorithms on Mutagenesis$_{RF}$

Mutagenesis$_{All}$

Figure 3.4 shows the accuracy obtained by RRR-P on the complete 230-instance Mutagenesis dataset (hereafter Mutagenesis$_{All}$), compared to other algorithms. The algorithms RRR-P is compared to here are several variations on FORF [3] (First Order Random Forests), which used out-of-bag estimation rather than ten-fold cross-validation, and TILDE, once again also including the RRR-SD and FOIL results from Chapter 2. The FORF variants differ in the use they make of aggregates (FORF-NA uses no aggregates).

As FORF used out-of-bag estimation rather than ten-fold cross-validation,

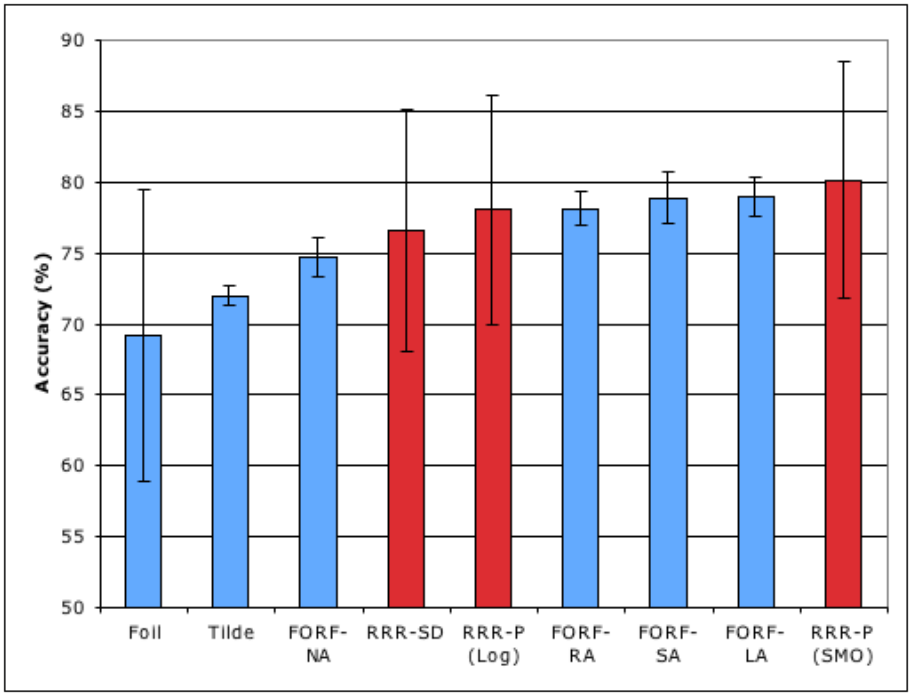


Figure 3.4: Accuracy for various algorithms on Mutagenesis$_{All}$

less weight should be placed on this comparison, but RRR-P(SMO) produces the highest accuracy by a small margin. RRR-P(Log) performs almost as well as the FORF methods that make use of aggregates. Again, both RRR-P results improve markedly on RRR-SD.

The standard machine learning algorithms used for RRR-P again showed clear peaks with regard to the parameters of the algorithms – Complexity 0.1 for SMO and Ridge 10 for Logistic – with the result for SMO once again being slightly higher than that for Logistic.

3.3.2 Musk$_1$

The Musk$_1$ dataset [20] is a set of 92 chemical compounds, some of which are classified (by expert human judges) as musk molecules. The dataset includes only compounds for which all published results agreed on their classification. Each compound can exist in a number of different conformations, depending on the rotation of its internal bonds. If at least one of the conformations for a molecule is determined to be a musk molecule, the molecule is classified as musk, otherwise it is classified as nonmusk.

The dataset consists of two relations:

- Compound: compound(CompoundID, Class)

Table 3.3: Distribution of Musk_1 instances

Positive	Negative	Total	Majority
47	45	92	0.511

- Conformation: conformation(CompoundID, ConformationID, F1, F2, ... , F166)

Where:
CompoundID is the unique identifier for the compound
Class is either *Musk* or *Nonmusk*
ConformationID is the unique identifier for the Conformation
F1 through F162 describe distances from the origin to the molecule's surface along 162 different vectors
F163 through F166 describe the position of the single oxygen atom in the Conformation

The distribution of the dataset is shown in Table 3.3.

Results are given in Figure 3.5 for RRP-P and RRR-SD. Published results are also reported for EM-DD [84], Relational Kernels [82], Iterated Axis-Parallel Rectangles [20], mi-SVM [1] (a support vector machine extension for multiple-instance data), KeS [28] (a support vector machine using a kernel for structured data), RIPPERMI [15] (a multiple-instance extension to the rule learner RIPPER) and TILDE. Results for EM-DD, KeS and IAPR were obtained from [28], and results for TILDE and RIPPERMI were obtained from [15].

It has been previously noted in [1] that the IAPR algorithm is optimised for the Musk classification task, which accounts for the gap between it and the other algorithms shown. RRR-P(SMO) performs similarly to the other non-IAPR algorithms, with RRR-P(Log) slightly worse – both RRR-P methods still improve on RRR-SD, however.

The highest accuracy was achieved for SMO with Complexity 0.1 and greater. At Complexity 0.1, the training data is classified perfectly (possibly due to the combination of a small number of instances and a large number of attributes), and so higher Complexity values produce identical results, as shown in Figure 3.6. This occurs because the SMO algorithm terminates when full separation of training instances is achieved. The best accuracy was achieved at Ridge parameter 0.1 for Logistic, as shown in Figure 3.7.

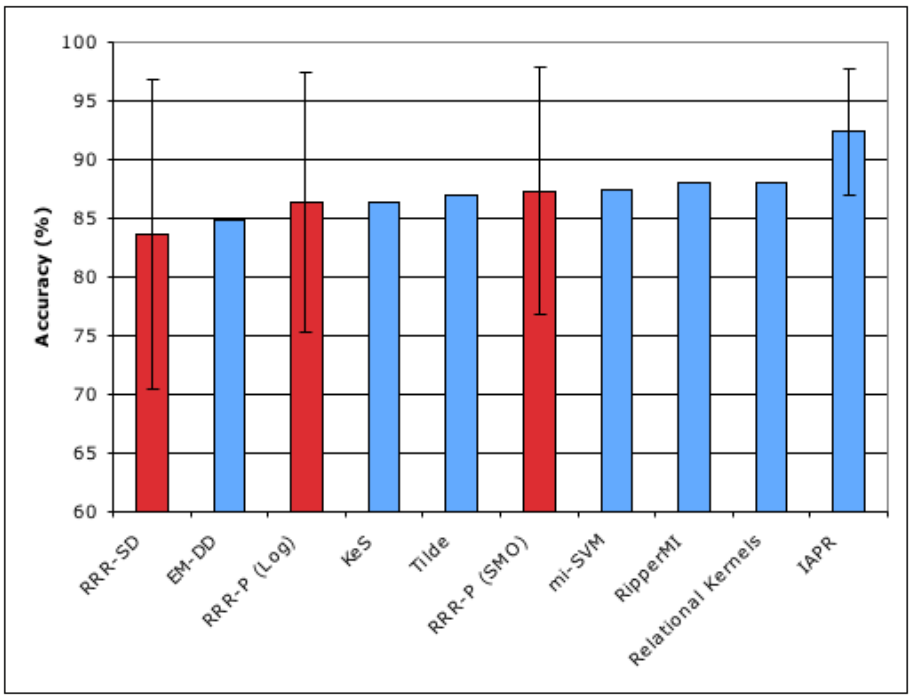


Figure 3.5: Accuracy for various algorithms on Musk_1

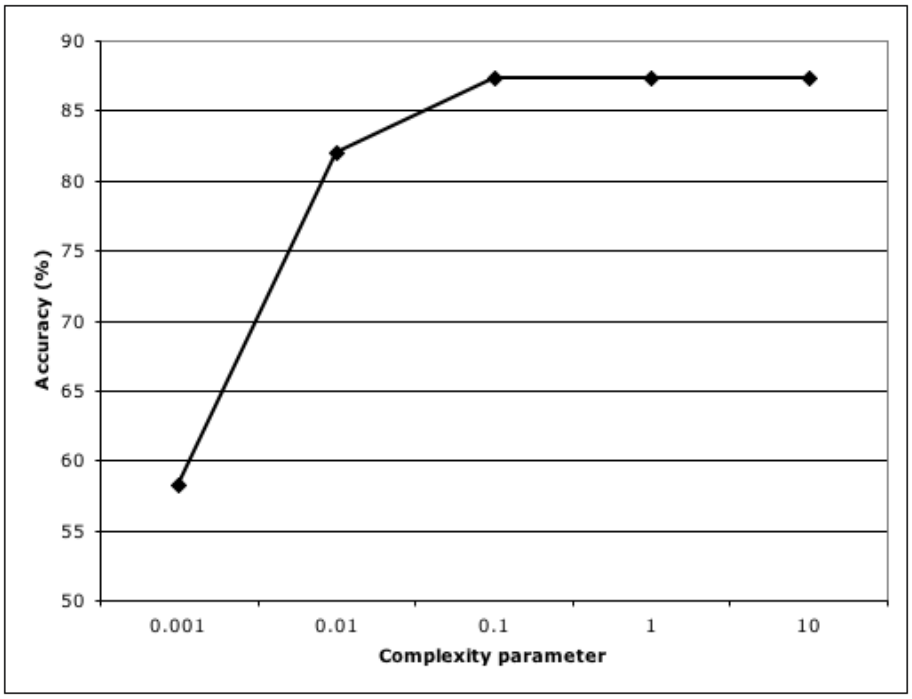


Figure 3.6: Accuracy for RRR-P on Musk_1, using SMO

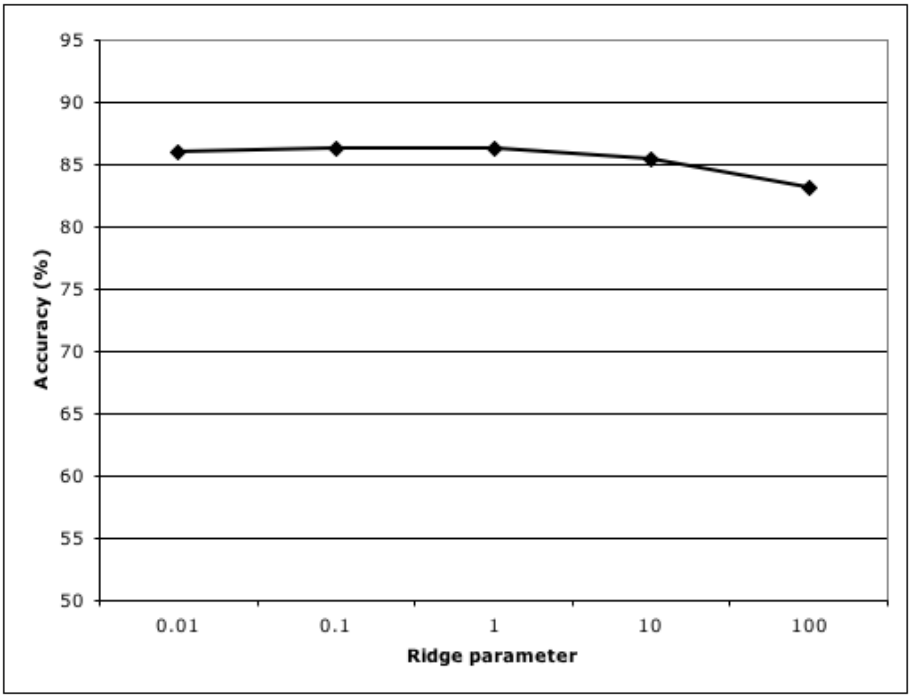


Figure 3.7: Accuracy for RRR-P on Musk_1, using Logistic

3.3.3 Carcinogenesis

The Carcinogenesis dataset [36] is a set of 330 diverse organic compounds. The goal is to predict which of the compounds are carcinogenic. Obtaining information on the carcinogenicity of compounds by empirical experimentation is slow and also requires experiments on animals, so a reliable machine learning model for carcinogenicity detection would be of great use.

The distribution of the dataset is shown in Table 3.4. Again, for experiments with RRR, only the Atom and Bond information in the dataset is used – the extra structural information is not used.

The dataset consists of three relations:

- Compound: compound(CompoundID, Class)
- Atom: atom(CompoundID, AtomID, Element, QuantaType, Charge)
- Bond: bond(CompoundID, AtomID, AtomID, QuantaType)

Where:
CompoundID is the unique identifier for the compound
Class is either *active* or *inactive*
AtomID is the unique identifier for the atom

Table 3.4: Distribution of Carcinogenesis instances

Positive	Negative	Total	Majority
182	148	330	0.552

Element is the chemical element of the atom
QuantaType is the type of the atom or bond, as assigned by the molecular modelling package QUANTA
Charge is the partial charge of the atom

Results are given in Figure 3.8 for RRR-P, RRR-SD and FOIL. Published results are also given for PROGOL [36] and ensemble methods applied to ALEPH [21] (in particular Different Seeds), although it should be noted that these results are obtained using five-fold cross-validation, while ten-fold cross-validation was used for RRR-P and FOIL.

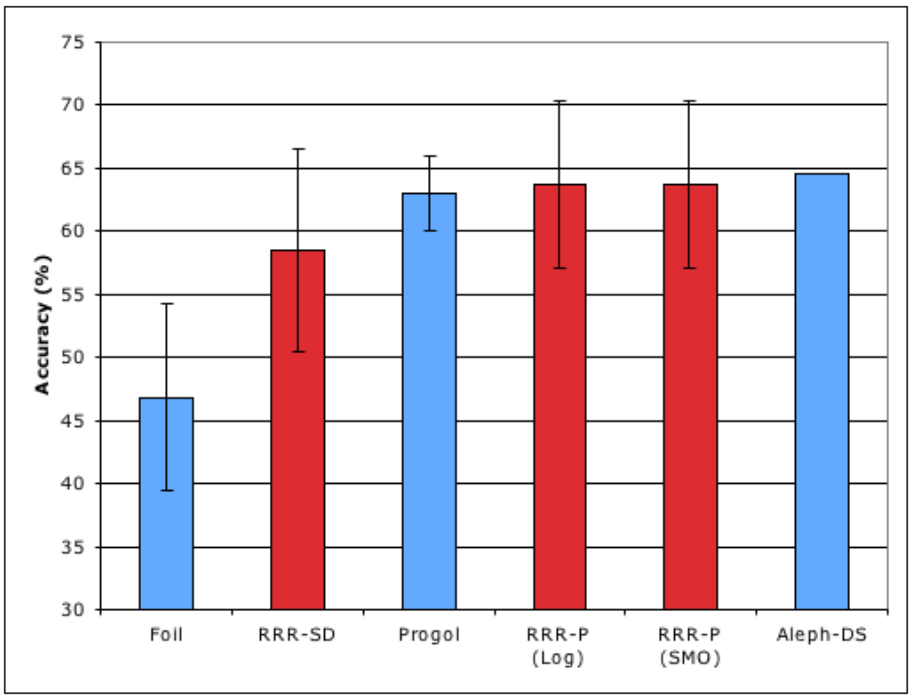


Figure 3.8: Accuracy for various algorithms on Carcinogenesis

Both propositional algorithms achieved results very similar to that of PROGOL on this dataset. As with the previous datasets, both forms of RRR-P improve markedly on RRR-SD.

The standard machine learning algorithms used for RRR-P showed clear peaks, with the peak accuracy for SMO achieved with Complexity 0.01. The

peak for Logistic was with a Ridge parameter of 1000, and SMO was the better-performing of the two attribute-value algorithms on this dataset.

3.3.4 Diterpenes

Diterpenes [23] are a class of organic compounds with about 5000 members known. They are of interest due to their use as lead compounds in searching for new pharmaceutical effectors. The skeleton of every diterpene contains twenty carbon atoms. Most diterpenes belong to one of twenty common skeleton types. The problem posed by the Diterpenes dataset is to identify the skeletons of diterpenes given their ^{13}C-NMR-Spectra (Nuclear Magnetic Resonance). The ^{13}C-NMR-Spectra include frequencies and multiplicities for each atom in the skeleton, and are obtained by analysing the spectrums emitted by nuclei excited by radio pulses.

If each carbon atom in each of the compounds is assigned an atom number, based on its place in the skeleton, the problem becomes a propositional one, with very good results achieved by propositional learners. However, the assignment of atom numbers is a difficult process itself, and thus the relational representation of the dataset, without assigned atom numbers, is of interest.

The Diterpenes dataset contains 1503 instances, from 23 classes. Their distribution is given in Table 3.5 – names are not given for the seven single-instance classes in [23].

As RRR-SD is limited to two-class problems, the datasets used by RRR-P were constructed from the three largest classes in the Diterpenes dataset – Labdan, Clerodan and Kauran. Three two-class datasets were created – one for each combination of the three classes. They are identified (using the class codes rather than the class names) as Diterpenes$_{52,54}$, Diterpenes$_{52,3}$ and Diterpenes$_{54,3}$, and their distribution is given in Table 3.6. In Chapters 4 and 5 algorithms are described that are not limited to two classes, and these algorithms are tested on the full Diterpenes dataset (Diterpenes$_{All}$), so the distribution for that dataset is also given in Table 3.6.

The dataset consists of two relations:

- Compound: compound(CompoundID, Class)
- Spectrum: spectrum(CompoundID, Multiplicity, Frequency)

Where:
CompoundID is the unique identifier for the compound

Table 3.5: Distribution of Diterpenes instances

Class Name	Class Code	Quantity
Labdan	c52	448
Clerodan	c54	356
Kauran	c3	353
Pimaran	c22	155
Beyeran	c4	72
Atisiran	c5	33
Gibban	c18	13
Cassan	c47	12
Spongian	c46	10
Trachyloban	c2	9
6,7-seco-Kauran	c28	9
Erythoxilan	c33	9
8,9-seco-Labdan	c80	6
Portulan	c71	5
5,10-seco-Clerodan	c79	4
Ericacan	c15	2
	c8	1
	c10	1
	c31	1
	c32	1
	c46	1
	c60	1
	c64	1

Table 3.6: Distribution of Diterpenes instances and three two-class subsets

Dataset	Total Size	Majority
Diterpenes_{All}	1503	0.298
$\text{Diterpenes}_{52,54}$	804	0.557
$\text{Diterpenes}_{52,3}$	801	0.559
$\text{Diterpenes}_{54,3}$	709	0.502

Class is the Diterpene skeleton code (c52, c54, c3, etc.)
Multiplicity describes the number of protons bound to the carbon atom emitting the spectrum - s, d, t and q for 0, 1, 2 and 3 respectively
Frequency is the resonance frequency of the carbon atom emitting the spectrum

As the two-class Diterpenes datasets have no other published results, results are only given in Figure 3.9 for RRR-SD, RRR-P and FOIL. The ordering of the four algorithms is consistent across all three datasets, with RRR-P(SMO) producing higher accuracy than RRR-P(Log), and both RRR-P methods outperforming FOIL, which in turn is more accurate than RRR-SD, as previously seen in Chapter 2.

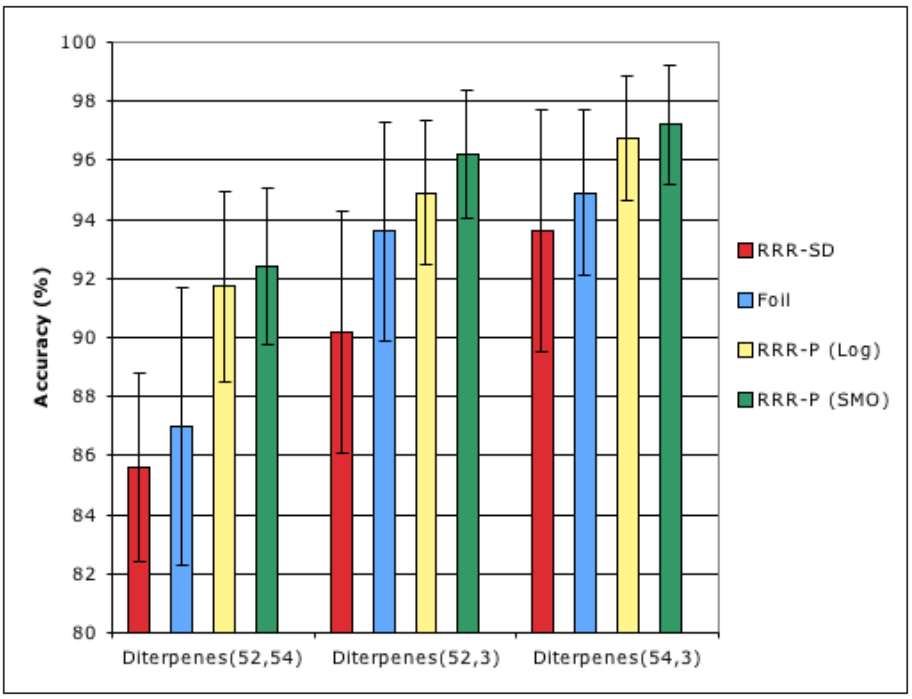


Figure 3.9: Accuracy for various algorithms on $\text{Diterpenes}_{52,54}$, $\text{Diterpenes}_{52,3}$ and $\text{Diterpenes}_{54,3}$

All three of the two-class Diterpenes subsets showed peaks in accuracy at Complexity 0.1 for SMO. However, when varying the Ridge parameter of the Logistic algorithm, the accuracy obtained was highest at the highest of the five tested Ridge values for $\text{Diterpenes}_{52,54}$. When the range of Ridge parameters was extended upwards, the highest accuracies for this dataset occurred at Ridge 100, as illustrated in Figure 3.10. The highest accuracy for $\text{Diterpenes}_{54,3}$ and $\text{Diterpenes}_{52,3}$ occurred at Ridge 10.

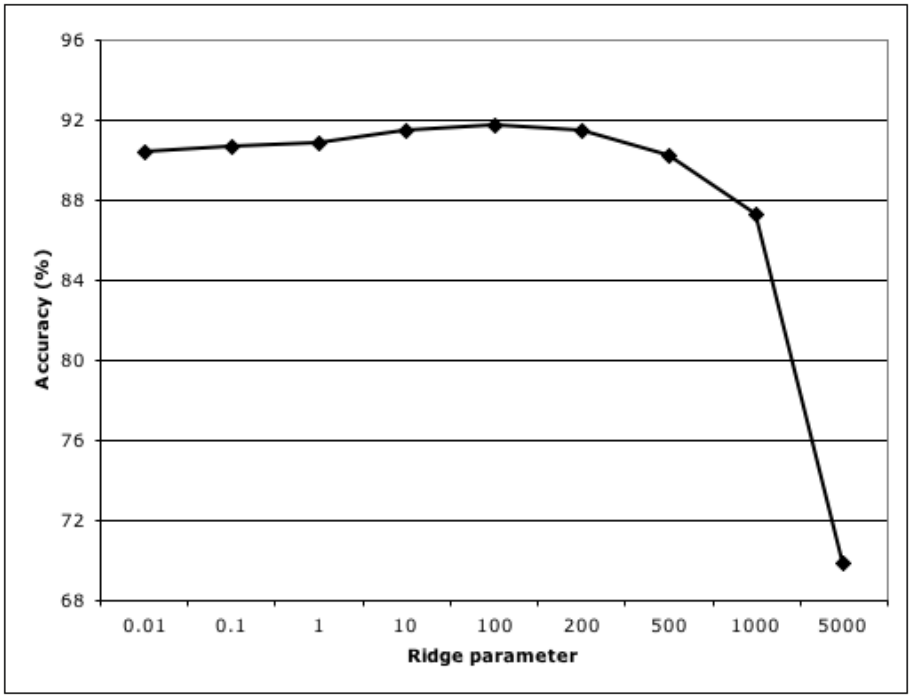


Figure 3.10: Accuracy for RRR-P on $\text{Diterpenes}_{52,54}$, using Logistic

3.4 Summary

From the results in Section 3.3, it can be seen that RRR-P (with either propositional algorithm) achieves higher accuracy than RRR-SD across all of the datasets that the algorithms were tested against. In all of these cases, the highest accuracy for RRR-P obtained with SMO as the propositional algorithm was superior to that obtained with Logistic. The results achieved by RRR-P are sensitive to the parameters of the propositional algorithms used, to varying degrees – on the Diterpenes datasets, the Ridge parameter of Logistic was varied widely without great impact on accuracy, but on Mutagenesis_{RF} SMO dropped sharply when the Complexity was varied from 0.1. The accuracy of RRR-P is competitive with the reported results of several other relational learning algorithms.

Chapter 4

Relational Clustering

4.1 Introduction

Clustering is a process by which instances are divided into groups, where appropriate groupings are determined by some distance measure. Relational clustering applies this process to relational data. Such distance measures are more complex to determine for relational data than for propositional data, as relational data cannot easily be fitted to a Euclidean framework. RDBC [37], for example, uses the distance measure of RIBL [24, 32], which recursively compares the relational elements of the data until features can be propositionally compared. A metric for terms and clauses is described in [33], and relational distance measures can also be derived from relational kernels [28, 82].

Clustering of relational data has so far received substantially less attention than classification of such data. One approach, based on a relational clustering tree as a variant of the relational tree learner TILDE, is described in [7]. This chapter describes RRR-C, a two-tiered approach to relational clustering that obviates the need for a relational distance measure, allowing standard propositional clustering algorithms to be applied to multi-relational data. In the first step the relational data is propositionalised [40] using randomly generated first-order rules (similar to the relational association rules generated by WARMR [69]), which are then converted into Boolean features, based on their coverage. The generation process restricts the rules to be within certain coverage minima and maxima to avoid overly specific or general rules, respectively. The rules are also generated in a manner that encourages even coverage across the data. In the second step, the resulting propositional dataset is clustered using a standard propositional clusterer such as k-means [50].

Section 4.2 details the algorithm, Section 4.3 reports on experiments, and Section 4.4 provides a summary of the chapter.

4.2 Randomised Relational Clustering

The RRR-C (Randomised Relational Rules – Clustering) algorithm comprises two tiers: a first level which generates random rules aiming to cover all examples as uniformly as possible, and a second level which turns these rules into Boolean features for a propositional representation. This acts as input for any propositional clustering algorithm.

The experiments using RRR-C reported below employed standard k-means using standard Euclidean distance. Random rules are generated using RRR, but with one modification, as clustering operates on data without class labels. As Enrichment makes use of class information, the Enrichment requirement for the individual rules is replaced with a requirement that the coverage of the rule on the training data be between user-defined minima and maxima. This prevents against both very specific and also against very general rules; worst cases would be universally true rules or rules covering just a single example. With the removal of the Enrichment requirement, rulesets are no longer biased towards a particular class, so only one ruleset needs to be generated by the algorithm. The Uniformity requirement is retained, with small batches of rules being generated and the most uniformity-preserving non-zero subset of each batch being added to the ruleset.

The basic algorithm for RRR-C is given in Algorithm 12. The complexity of RRR-C is the sum of the complexity of both stages. Usually, when using propositionalisation in ILP, the propositionalisation stage dominates the total complexity, and this is true for RRR-C as well. Even though generating a random rule is extremely fast, its coverage still has to be determined both for checking the coverage constraints and uniformity of coverage, as well as to generate the propositional data-set. In the worst case this coverage computation can be exponential, even for a single rule. The complexity of rule evaluation is discussed in Chapter 2. The complexity of propositional clustering algorithms on the contrary is often linear or quadratic at worst (k-means, for example, is linear with regard to both the number of instances and the number of attributes in the data [16]). The time required to create a propositionalisation of a dataset can be up to two orders of magnitude greater than that required for a clustering run using a specified cluster number and random seed, but the

propositionalisation thus created can be used for more than one clustering run.

Algorithm 12 Pseudocode for the RRR-C algorithm

```
while Number of rules in ruleset is less than the minimum do
  while Number of rules in batch is less than the minimum do
    Generate a Rule
    if Rule is within coverage constraints (minimum-maximum coverage)
    then
      Add Rule to rule batch
    end if
  end while
  Calculate the most uniformity-preserving non-zero subset of rules in the
  current rule batch
  Add those rules to the ruleset
end while
use ruleset to generate Boolean-valued propositional dataset
apply any propositional clustering algorithm
```

Indeed, in some cases the total time required for a full clustering run across multiple random seeds and cluster numbers (as described in Section 4.3.1) for a particular propositionalisation was less than the time required to generate the propositionalisation itself.

4.3 Experiments

This section describes the experiments performed using RRR-C and compares the results to other relational clustering systems. Section 4.3.1 details the experimental setup, Sections 4.3.2 and 4.3.3 discuss the results using two different measures of clustering quality, and Section 4.3.4 gives an example of a particular clustering run.

4.3.1 Experimental Setup

An evaluation of RRR-C on several datasets has been conducted, always generating random rules for the full dataset, and then clustering the resulting propositional data with the k-means algorithm [50], using Euclidean distance. RRR-C was run with five different coverage ranges – 5%-50%, 10%-50%, 25%-50%, 25%-75% and 'Wide' (which covered at least two instances and at most one less instance than the dataset size) – generating 1,000 rules (and thus 1,000 propositional attributes) on each run. The following datasets were

used: Mutagenesis$_{RF}$, Mutagenesis$_{All}$, Musk$_1$, Carcinogenesis, Diterpenes$_{54,3}$, Diterpenes$_{52,3}$, Diterpenes$_{52,54}$ and Diterpenes$_{All}$. As noted in Chapter 3, for Mutagenesis and Carcinogenesis only low-level structural information is used, as represented by atoms and bonds – neither global properties (e.g. *lumo* or *logP*) nor predefined functional groups are included.

The propositionalisation process described in Algorithm 12 is class-blind. Generated rules are added if the proportion of training instances they cover falls within a defined range, independent of the class labels of those training instances (unlike Enrichment, which is calculated using class labels). This allows RRR-C to be applied to datasets with more than two classes, and thus it was possible to use Diterpenes$_{All}$ as well as the three two-class subsets thereof.

To study the influence of the number of clusters that number was varied from 2 up to 50. Ten propositionalisations were generated for each of Musk$_1$, Mutagenesis$_{RF}$, Mutagenesis$_{All}$ and Carcinogenesis, and each propositionalisation was clustered with ten different random seeds for the k-means algorithm. For time reasons (the execution time of k-means in particular – although a single run of k-means is generally much faster than a propositionalisation run, the number of k-means runs required for the varying cluster numbers and random seeds was substantial), only three propositionalisations were generated for each of the larger datasets (Diterpenes$_{All}$ and its subsets), and each was clustered with three different random seeds. Multiple runs and seeds were used to ensure stable results, as the combination of the randomness of RRR-C's propositionalisation and the sensitivity of the k-means algorithm to its random seed could lead to highly variable results.

RRR-C is compared to two other relational clustering approaches. RSD [80], like RRR-C, can generate Boolean-valued propositional datasets which can then be clustered by standard k-means. Contrary to RRR-C's random heuristic approach, RSD generates rules via systematic search. The minimum coverage for RSD's features was set to four different values – as RSD does not have a setting for maximum coverage, minimum coverages were selected to match those used by RRR-C - two instances ('Wide') and 5%, 10% and 25% of the number of instances in the dataset. RSD usually produces a smaller number of rules than RRR-C (which is set to generate 1,000, as noted above), which can be attributed to RSD's non-duplication and connectivity requirements. However, RSD produces more than 1,000 rules for all datasets except Mutagenesis$_{RF}$ and Mutagenesis$_{All}$ when the minimum coverage is set to two instances. It also produces more than 1,000 rules for all coverage settings on Musk$_1$, due to

Table 4.1: Number of rules generated by RSD

Minimum coverage Dataset	25%	10%	5%	2 instances
Musk_1	11,187	14,154	17,845	23,853
Mutagenesis_{RF}	72	108	151	359
Mutagenesis_{All}	87	124	171	445
Carcinogenesis	364	436	469	1,549
$\text{Diterpenes}_{54,3}$	391	483	558	3,452
$\text{Diterpenes}_{52,3}$	467	562	659	3,556
$\text{Diterpenes}_{52,54}$	379	481	558	3,637
Diterpenes_{All}	675	813	915	5,158

the high-arity Conformation predicate of that dataset. The number of rules generated by RSD for each dataset is shown in Table 4.1. The maximum rule length for RSD was set to 5 literals for all datasets except Musk_1 (which was set to 4), as greater rule lengths resulted in impractically long runtimes.

The second system RRR-C is compared to is the Relational K-Means (RKM) algorithm of RelWeka [83], which implements the RIBL [32] distance measure, a proper distance for relational data. The RIBL distance measure makes use of relative "edit distances" between instances. Whereas RSD and RRR-C are very similar, RKM is a rather different approach based on more direct relational clustering, which does not rely on propositionalisation. While RRR-C and RSD both generate several different representations of the datasets for clustering, using different coverage ranges, RKM uses the datasets directly.

4.3.2 Penalised Error Rate

There exists no single universally agreed upon measure for clustering quality. As true class labels are available for all datasets, which are not used during clustering, one possible measure of cluster quality is the agreement of clusters with classes. Clearly one would expect better accuracies with more clusters, as it should be easier to find smaller class-pure clusters than larger ones. One caveat here is that when taking the majority class of each cluster as its "label", clusters with only one example will automatically be correct – the degenerate case of this being a clustering where each cluster contains only one instance. Such a clustering would be treated as perfect. For this reason a Penalised Error Rate was used that treats instances in single-instance clusters as errors.

The trends visible for penalised error rates follow reasonable expectations

– in general, higher number of clusters lead to smaller penalised error rates. The exception to this is the Musk_1 dataset, where the comparatively small number of instances leads to higher numbers of single-instance clusters as the number of clusters generated increases. Indeed, the penalised error rate begins to increase at around 30 clusters for Musk_1, as shown in Figure 4.1. On the Musk_1 datatset, RRR-C and RSD performed very similarly, with RRR-C(Wide) performing slightly worse than the other algorithm-coverage pairs.

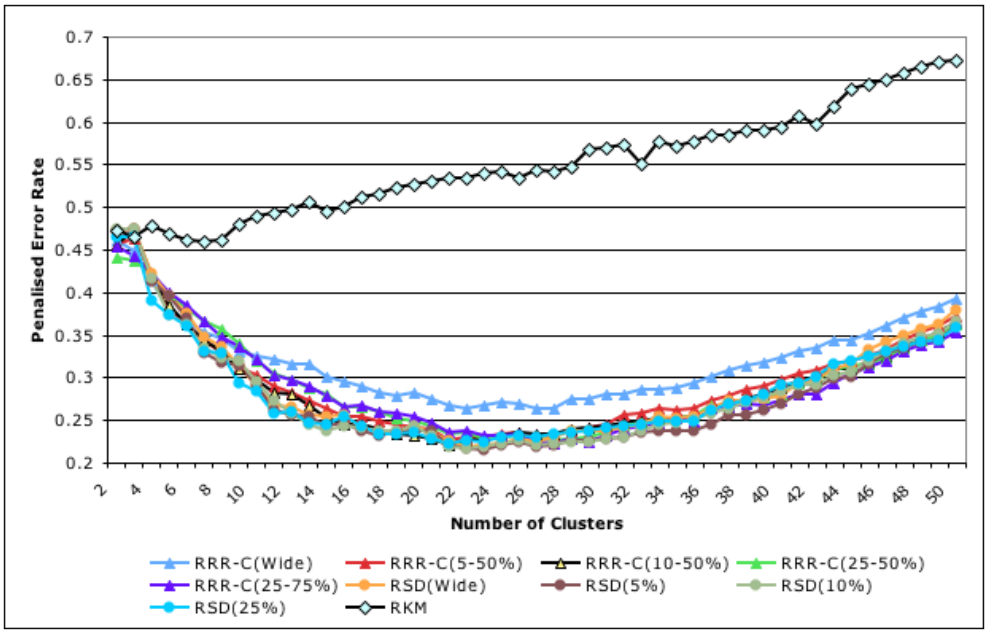


Figure 4.1: Penalised error rates on Musk_1

For Mutagenesis_{RF} and Mutagenesis_{All} (see Figures 4.2 and 4.3), the penalised error rates for RRR-C and RSD are also very similar, across all coverage ranges. On both datasets, RRR-C performs slightly better than RSD for smaller numbers of clusters, but as the number of clusters is increased the difference between the penalised error rates decreases.

On Carcinogenesis, RRR-C and RSD again perform similarly. For smaller numbers of clusters, RRR-C(25%-75%) performs slightly worse than the others.

On all four of the above datasets, RKM performed worst of the three systems. This is at least partially due to the tendency of RKM to produce both larger clusters (less likely to be class-pure) and more single-instance clusters (automatic errors) than either of the other two algorithms, which increase the penalised error rate. An example of this for a 10-cluster run on the Musk_1 dataset is shown in Table 4.2.

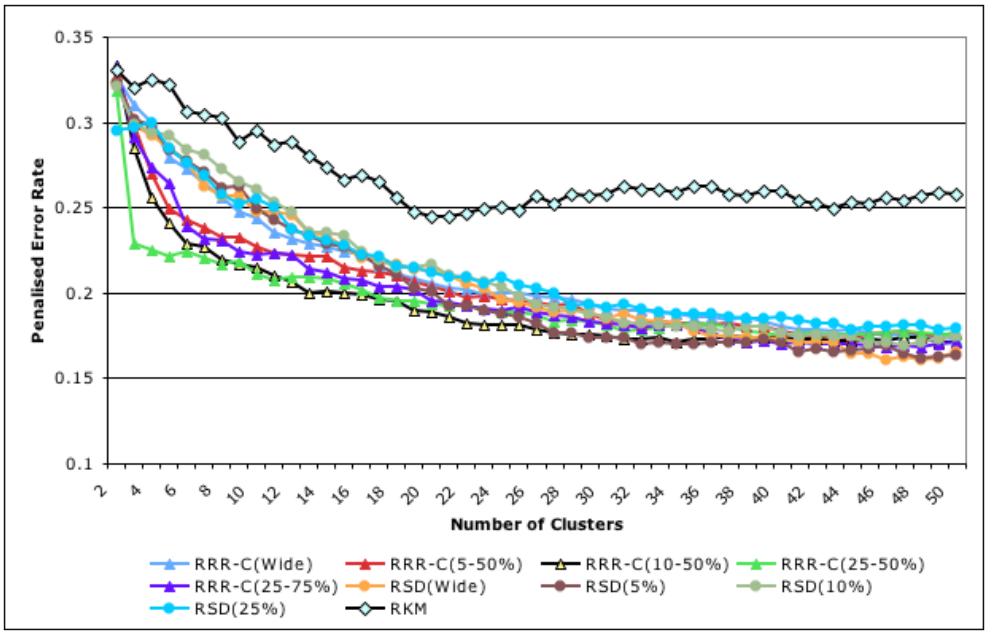


Figure 4.2: Penalised error rates on Mutagenesis$_{RF}$

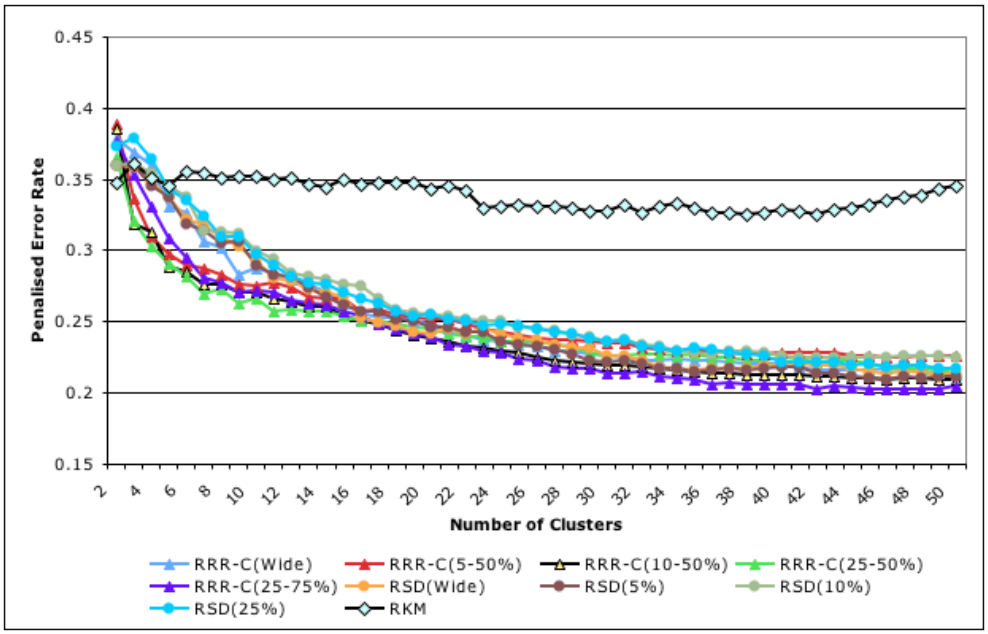


Figure 4.3: Penalised error rates on Mutagenesis$_{All}$

Table 4.2: Size of clusters generated by RKM on $Musk_1$

Cluster Size	Quantity
80	1
4	1
1	8

On the Diterpenes datasets, the trends displayed are somewhat different. For these datasets, RKM and RRR-C generally perform better than RSD, although RRR-C(Wide) consistently produced a higher penalised error rate than the other RRR-C experiments. The four coverage ranges for RSD produce penalised error rates that are very similar to each other, with only the minor exception that RSD(25%) performs slightly worse than the other coverage ranges on $Diterpenes_{54,3}$.

RKM produces a much smaller number of single-instance clusters on these datasets, which may contribute to its improvement in penalised error rate relative to the other algorithms, when compared to the non-Diterpenes datasets.

The difference between the penalised error rates for RRR-C and RSD may be due to RSD's restriction on rule generation – RSD will not accept rules that can be decomposed into two or more distinct rules.

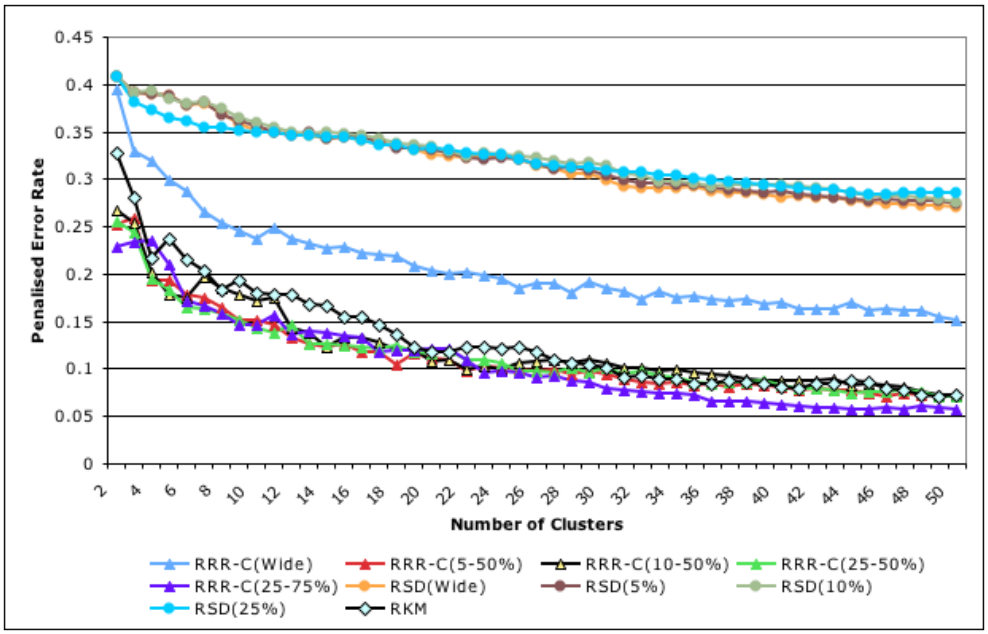


Figure 4.4: Penalised error rates on $Diterpenes_{52,54}$

While the penalised error rates for RRR-C on the Diterpenes subsets are

very low (as shown in Figure 4.4 – the penalised error rates for the other two subsets follow a similar pattern), the penalised error rate for Diterpenes$_{All}$ is substantially higher, as shown in Figure 4.5. This occurs because Diterpenes$_{All}$ is a 23-class dataset, with a skewed class distribution such that three classes make up over 75% of the dataset – most of the generated clusters are dominated by one of the three major classes.

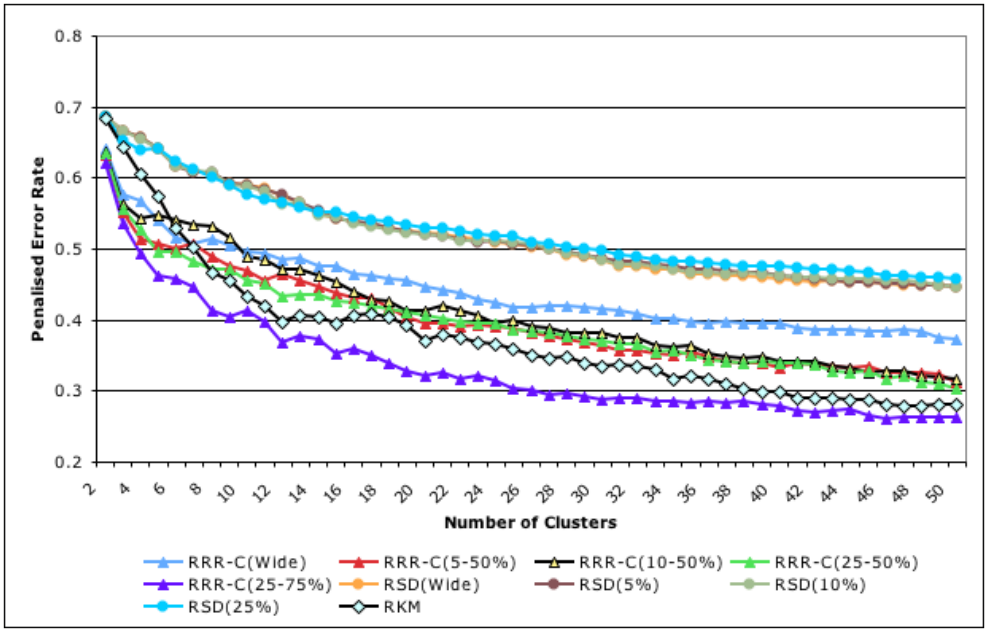


Figure 4.5: Penalised error rates on Diterpenes$_{All}$

The worse performance of the Wide coverage for RRR-C, compared to the other coverage ranges, may be related to the fact that when this coverage range is used, a high proportion of the generated rules have coverage in the range (two instances – 5% of instances), as shown in Table 4.3.

As Carcinogenesis has somewhat similar proportions of low-coverage attributes to Diterpenes under RRR-C, but does not display this behaviour, it may be that the number of instances in the dataset is also a factor, given that the Diterpenes datasets are 2-4 times larger than the Carcinogenesis dataset. The Wide coverage range is the only one bounded by an absolute number of instances, rather than a proportion, and two instances is a much smaller proportion of the Diterpenes datasets than of the smaller datasets, resulting in rules with correspondingly low coverage. It may even be the case that this behaviour is the result of some unknown property of the Diterpenes data.

The penalised error rate is substantially higher for some datasets when

Table 4.3: Proportion of rules generated by RRR-C that cover (2 instances – 5% of instances)

Dataset	Proportion
$\text{Diterpenes}_{52,3}$	0.7204
Carcinogenesis	0.7180
$\text{Diterpenes}_{52,54}$	0.7104
$\text{Diterpenes}_{54,3}$	0.7039
Diterpenes_{All}	0.6961
Mutagenesis_{RF}	0.6372
Mutagenesis_{All}	0.6353
Musk_1	0.2946

rules are generated using the Wide coverage range – rulesets generated with a higher minimum coverage for rules appear to perform better for clustering.

The Normalised Mutual Information (NMI) measure [51] for cluster evaluation was also investigated, but the relative performance of the algorithms was very similar to that observed using the Penalised Error Rate.

4.3.3 Silhouette Width

Another measure used to compare clusterings is the average silhouette width [70]. The silhouette value s_i for an instance i is calculated according to the following formula:

$$s_i = \frac{b_i \text{ - } a_i}{\max(a_i,b_i)} \tag{4.1}$$

Where:
$a_i = md(i,c_i)$ (c_i = the cluster containing i)
$b_i = \min(md(i,c_{j\neq i}))$ for all clusters j not containing instance i
$md(i,c)$ = the mean distance from instance i to all instances in cluster c

The silhouette value is thus a measure of clustering quality that is independent of the class labels of the data, instead using the distance measure to determine whether an instance has been optimally clustered. It compares the average distance from a given instance i to each other instance in its cluster to the average distance from i to each instance in the closest cluster (the closest cluster being that with the smallest average distance to i across all instances it contains). Higher silhouette values therefore arise from tighter clusters (smaller

intra-cluster distances) and more separated clusters (larger inter-cluster distances). Silhouette values lie between -1 and +1, with lower values indicating an increasing likelihood that the instance could have been better placed in the cluster represented by b. A silhouette value of zero indicates that the instance could be equally well clustered in the cluster represented by b as in its current cluster.

In the case where a cluster contains only one instance, the silhouette value of that instance is defined to be zero, again to avoid overly positive evaluation of single-instance clusters. Under propositionalisation, it is possible for two or more instances to have identical attribute values. This occurs when these instances produce the same Boolean values for each of the rules generated by RRR-C or RSD. When a cluster is composed of instances with identical attribute values, the silhouette is calculated as in Equation 4.1, but with a value of 0 for a_i (because there is no distance between the instances), which gives a result of 1 for each instance in the cluster. This is shown in Equation 4.2 (which assumes that b_i is positive – this holds except in the pathological case that all instances in the dataset are identical). The effects of this property are further discussed later in this section.

$$s_i = \frac{b_i \text{ - } a_i}{\max(a_i,b_i)} = \frac{b_i \text{ - } 0}{\max(0,b_i)} = \frac{b_i}{b_i} = 1 \tag{4.2}$$

Where:
$a_i = 0$, as all instances in the cluster have identical attributes
$b_i = \min(md(i,c_{j \neq i}))$ for all clusters j not containing instance i
$md(i,c)$ = the mean distance from instance i to all instances in cluster c

To evaluate the quality of a clustering, the average silhouette width is used, which is the average of the silhouette values for all instances in a dataset (shown in Equation 4.3).

$$\text{Average silhouette width} = \frac{\sum_{i=1}^{n} s_i}{n} \tag{4.3}$$

Where:
s_i = the silhouette value for the i^{th} instance in the dataset
n = the number of instances in the dataset

A subjective interpretation of the average silhouette width is given in [70],

Table 4.4: Interpretation of silhouette width

Average Sil. Width	Interpretation
0.71-1.00	Strong structure
0.51-0.70	Reasonable structure
0.26-0.50	Weak structure
up to 0.25	No substantial structure

and described in Table 4.4.

The average silhouette width follows similar trends for the Mutagenesis$_{RF}$ and Mutagenesis$_{All}$ datasets (shown in Figures 4.6 and 4.7) for both RRR-C and RSD – a slow increase as the number of clusters increases, although with an initial peak for a very small number of clusters, followed by a drop, for Mutagenesis$_{RF}$.

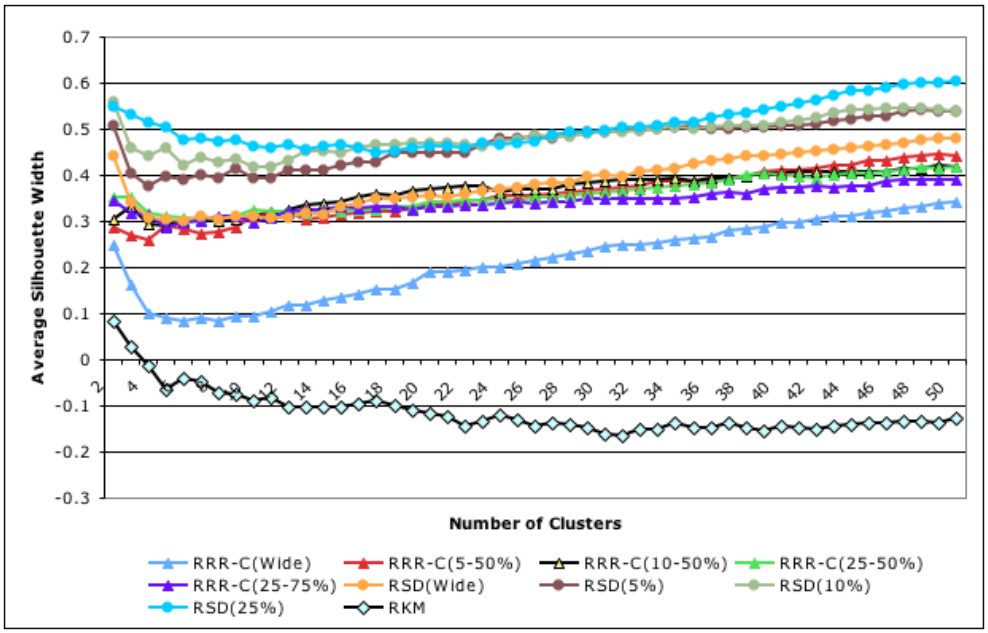


Figure 4.6: Average silhouette widths for Mutagenesis$_{RF}$

The silhouette values for the Mutagenesis datasets show clear differences – RSD produces higher silhouette values than RRR-C. The silhouette value for RSD(Wide) is lower than that for the other coverage ranges, and similarly, RRR-C(Wide) produces worse silhouette values than the other RRR-C runs. Frequently the RSD silhouette values are ordered by minimum coverage – RSD(25%) performing better than RSD(10%), and so on – although for some

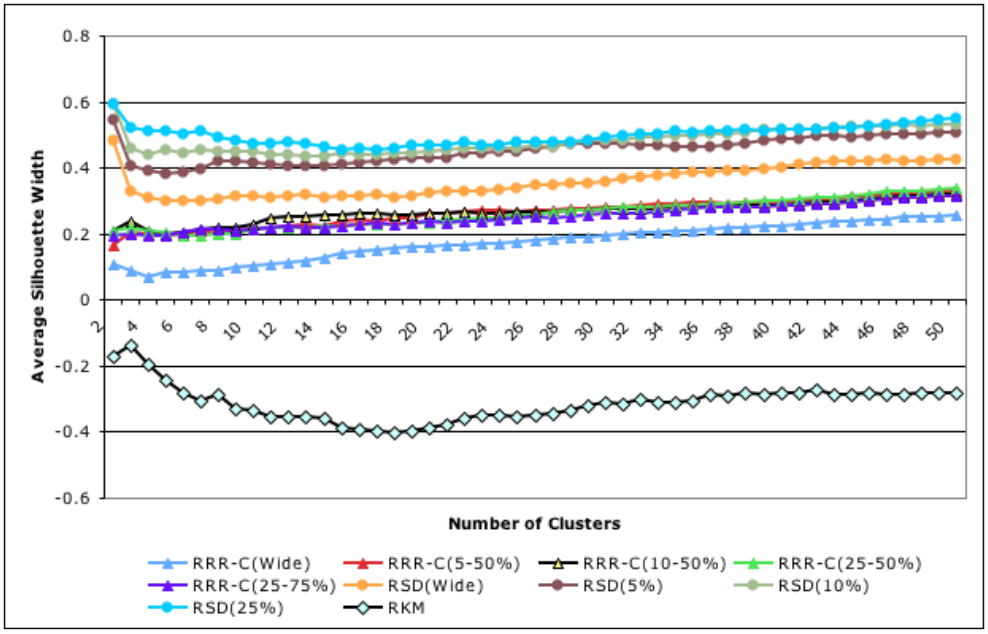


Figure 4.7: Average silhouette widths for Mutagenesis$_{All}$

numbers of clusters the values are very similar. RKM performs substantially worse than both. The high number of single-instance clusters generated by RKM explains its low silhouette width – not only do instances in single-instance clusters have silhouette values of zero themselves, they can also significantly lower the silhouette widths of instances in larger clusters that lie in close proximity. In addition to this, properties of the non-Euclidean RIBL distance measure may also affect silhouettes. The Wide coverage range tends to generate more single-instance clusters than the other coverage ranges, explaining the slightly worse silhouettes obtained by both RRR-C(Wide) and RSD(Wide). By Rousseeuw's interpretation (in Table 4.4) RSD produces clusterings that range from 'weak structure' to 'reasonable structure', while RRR-C produces 'weak structure'. The comparatively low silhouette widths for the Wide coverage runs fall into the 'weak structure' range for RSD and 'no substantial structure' for RRR-C.

On Musk$_1$, as shown in Figure 4.8, RKM produces poor silhouette values, while the silhouette values for RRR-C and RSD are very similar, with RRR-C(Wide) and RSD(Wide) producing slightly lower silhouette values than the other coverage ranges. As the number of clusters increases, all of the silhouette values tend towards zero, as the number of single-instance clusters generated also increases – and, as mentioned above, on the comparatively small Musk$_1$

dataset, all of the algorithms produce a greater number of single-instance clusters. The structure found is initially in the 'weak' range for most coverage ranges, but drops to 'no substantial structure' as the number of clusters is increased.

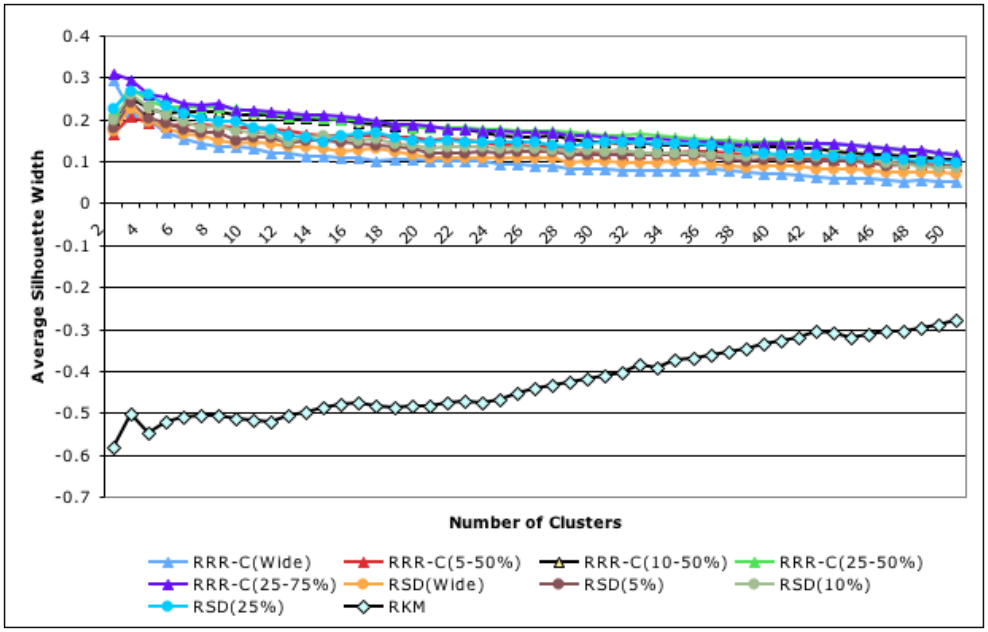


Figure 4.8: Average silhouette widths for Musk_1

On the Carcinogenesis dataset, the non-Wide RRR-C and RSD results stay within a narrow band of values as the number of clusters increases – in the high end of 'no substantial structure' and the low end of 'weak structure'. For higher numbers of clusters, RRR-C(25%-75%) shows a slight improvement over the others. Both RRR-C(Wide) and RSD(Wide) have distinctly worse silhouette widths than their non-Wide counterparts, with RSD(Wide) slightly outperforming RRR-C(Wide). RKM once again has a very low silhouette value.

On the Diterpenes datasets, the silhouette values produced show a distinct relationship to the coverage settings for both RRR-C and RSD – the silhouette values for $\text{Diterpenes}_{52,54}$ are shown in Figure 4.9. For each algorithm, as the minimum coverage for rules increases, so do the silhouette values produced. The silhouette values for RKM are improved from the results on the previous datasets. The silhouette values for RSD are substantially higher than for RRR-C (except for RSD(Wide)), falling in the category of 'weak structure', and at their peak 'reasonable structure', as opposed to 'no substantial structure' and the low end of 'weak structure' for RRR-C. For both algorithms, the Wide

Table 4.5: Number of unique instances under propositionalisation

Dataset	Number of Instances	Unique Instances			
		RRR-C (25%-75%)	RSD (25%)	RRR-C (Wide)	RSD (Wide)
Carcinogenesis	330	313.2	301	312.6	314
$\text{Diterpenes}_{52,3}$	801	796.0	599	796.3	798
$\text{Diterpenes}_{52,54}$	804	796.0	593	797.0	798
$\text{Diterpenes}_{54,3}$	709	703.0	537	702.6	704
Diterpenes_{All}	1503	1492.3	1066	1491.0	1503
Musk_1	92	92	92	92	92
Mutagenesis_{All}	230	175.0	115	172.2	141
Mutagenesis_{RF}	188	149.5	98	145.7	118

coverage range performs substantially worse than the other coverage ranges.

On Diterpenes_{All} (shown in Figure 4.10), RRR-C has slightly lower silhouette values ('no substantial structure') than on the Diterpenes subsets, but the ordering of those values the coverage ranges is the same. RSD behaves slightly differently, with RSD(25%) now producing worse silhouette values than RSD(5%). RKM has a particularly high silhouette width on Diterpenes_{All} for low numbers of clusters, in the 'weak structure' range.

One factor contributing to the high silhouette values produced by RSD on the Mutagenesis and Diterpenes datasets may be the larger numbers of instances that have duplicates under RSD's propositionalisation than under that of RRR-C (some examples of this are shown in Table 4.5. Each instance in a cluster consisting only of duplicated instances will have a silhouette value of 1 (as the average within-cluster distance is 0), as previously shown in Equation 4.2). Even in clusters that do not consist solely of duplicated instances, duplicated instances contribute to lower intra-cluster distances, which leads to higher silhouette values.

On the Musk and Carcinogenesis datasets, where RSD and RRR-C have very similar silhouette values, they also produce very similar numbers of duplicate instances.

Although in general both the penalised error rate and the average silhouette width improve as the number of clusters increases for most of the datasets and coverage ranges, they are measuring different things. The penalised error rate depends only on the agreement of class labels with clusters, and the average silhouette width only takes into account the relative groupings of clusters, ignoring class labels.

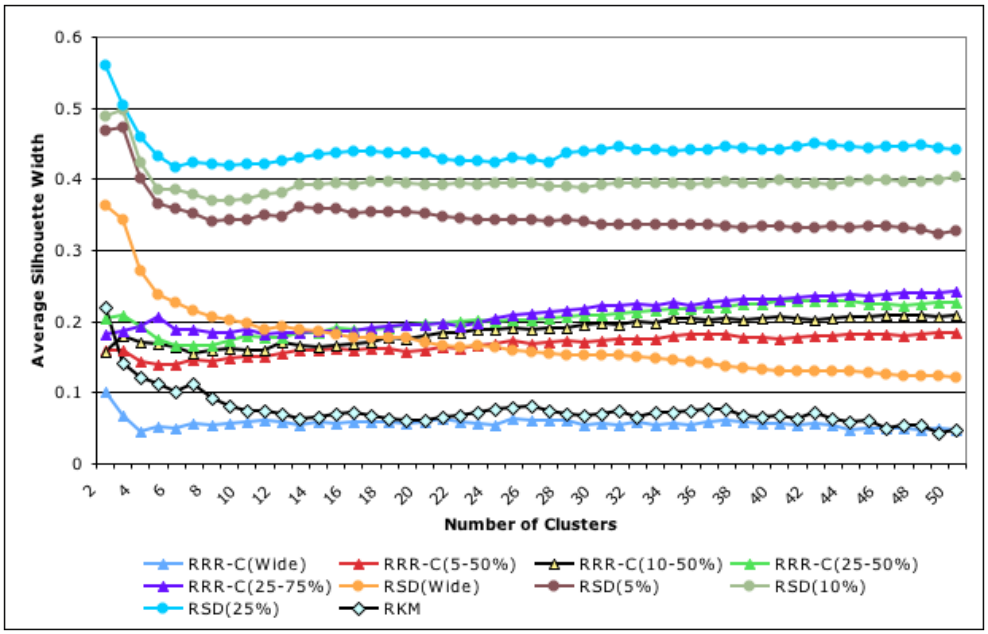


Figure 4.9: Average silhouette widths for Diterpenes$_{52,54}$

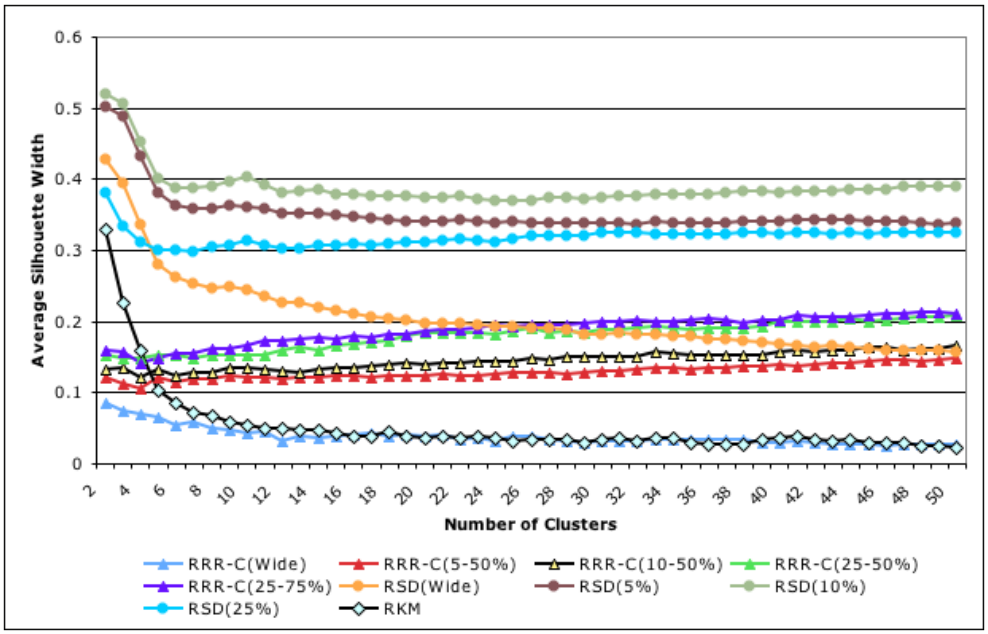


Figure 4.10: Average silhouette widths for Diterpenes$_{All}$

In particular, the silhouette value only examines the structure of the propositionalised representation of the dataset, and does not consider the relationship of the propositionalised instances to their class labels. For an extreme example, consider a single-attribute propositionalisation of a dataset, where each instance is represented by a randomly-assigned single Boolean value. Such a propositionalisation would have a perfect silhouette value when clustered, as each instance would have zero distance from each other instance in its cluster. However (unless the single attribute corresponded directly to the class of each instance) this propositionalisation would certainly not have a perfect Penalised Error Rate. Additionally, the silhouette value was originally intended for purposes such as determining the 'best' number of clusters to use in clustering a particular dataset, rather than cross-representation comparison.

The Penalised Error Rate can be said to reflect to some extent the quality of propositionalisation. Instances that are mutually similar should be grouped together by clustering, and with a 'good' propositionalisation instances of the same class should be similar (assuming that these similarities exist in the original data).

This divergence between Penalised Error Rate and silhouette value can be observed in the Diterpenes results. RSD has a substantially higher silhouette value than RRR-C on these datasets, but also a substantially higher Penalised Error Rate. This indicates that while RSD's clustering has created clusters that are more clearly separated than those produced by RRR-C, those clusters are not as class-pure. Furthermore, although the information obtained from the silhouette value is of interest, in the two-step setting where a propositional representation is generated and then clustered, a measure of clustering that takes into account the relation of the propositionalisation to the original class labels should be preferred to one that does not, as this is a definite indicator that groupings in the propositionalisation reflect groupings in the original data.

4.3.4 Example of Clustering

To get a further insight into the quality of clustering, Figure 4.11 depicts the class distribution for a particular 20 cluster partition of the 188 regression-friendly compounds from the Mutagenesis$_{RF}$ dataset using only 10 random rules. Still, 8 of the 20 clusters are class-pure, though all for the active class. Two of the random features generated are:

```
rule(MolId) :-
        bond(MolID,_,_,BondType),
        BondType == 2,
        atom(MolID,_,_,QuantaType,_),
        QuantaType == 27.

rule(MolId) :-
        atom(MolId,AtomId1,_,QuantaType1,Charge),
        Charge >= 0.178,
        atom(MolId,AtomId2,_,QuantaType2,_),
        AtomId2 != AtomId1,
        QuantaType1 == QuantaType2.
```

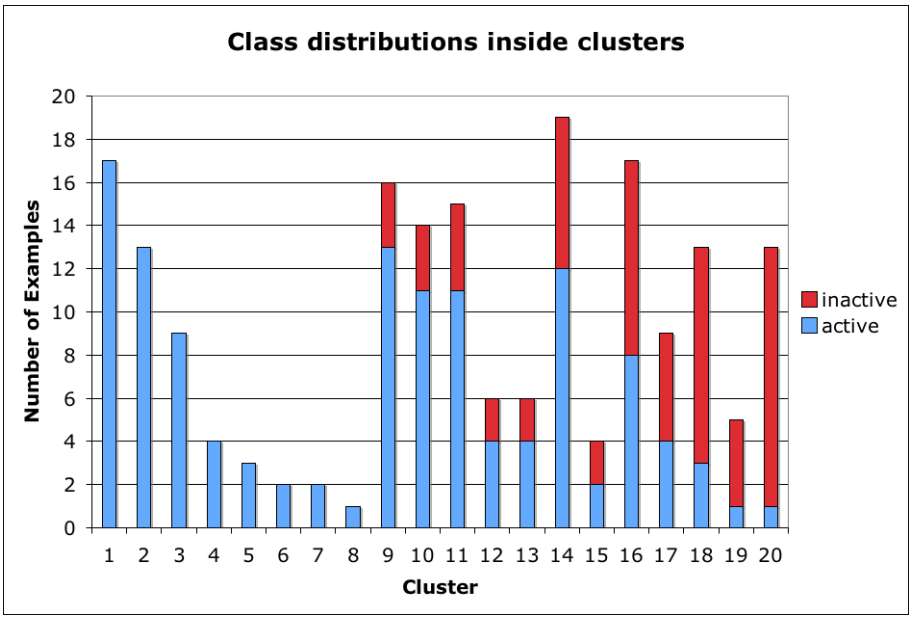
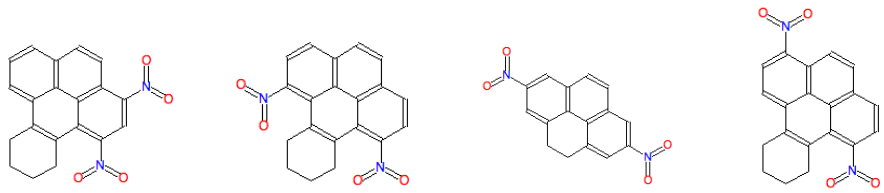


Figure 4.11: Class distribution for 20 clusters on Mutagenesis$_{RF}$

Respectively, they represent compounds with at least one double bond plus an atom of Quanta type 27, as well as compounds with two distinct atoms of the same Quanta type, where one must have a charge of at least 0.178. Selecting, for example, cluster number 4, which comprises four examples of the same class, their Boolean feature values are:

```
1  2  3  4  5  6  7  8  9 10
f, t, f, f, t, t, t, t, t, t  example1
f, t, f, f, t, t, t, t, t, t  example2
f, t, f, f, t, t, f, t, t, t  example3
f, t, f, f, t, t, t, t, t, t  example4
```

Figure 4.12: The four, all active compounds of cluster 4



Notice that these four examples are almost identical under this propositionalisation, with only one exception for attribute 7 for `example3`. Figure 4.12 shows the structure formulas for these four compounds, and indeed three of the four are almost identical, only one nitro-group is positioned differently for each of them, and the fourth compound (`example3`, third from the left) is also very similar in structure to the other three.

4.4 Summary

This chapter has described RRR-C – a two-tiered approach to relational clustering based on randomised propositionalisation and an arbitrary propositional clustering algorithm – and compared the results to two other approaches to relational clustering. The experimental results reported above look promising – as a point of reference, most of the penalised error rates for RRR-C are quite competitive to error rates that have been reported in the literature for relational classification algorithms on these datasets. The exceptions are Musk_1, due to the previously noted effect of the smaller dataset on the penalised error rate, and Diterpenes_{All}, which suffers from the combination of having 23 classes and having three of those classes make up almost 77% of the dataset. The quality of the clustering, as measured by both error rate and silhouette width, depends on the minimum coverage required of the generated rules for both RRR-C and RSD. RSD produced higher silhouette values than RRR-C for the Mutagenesis and Diterpenes datasets, but the silhouette value does not take into account the class labels in the original data. In most cases, given

equivalent rule coverage, the penalised error rates for RRR-C were equal to or lower than those of RSD, indicating that the propositionalisation of RRR-C was more effective than that of RSD at capturing relational information for clustering.

Chapter 5

Semi-supervised Learning

5.1 Introduction

In supervised classification, training is performed on a set of examples with assigned class labels, and the resulting model is then evaluated on the accuracy of the class labels it assigns to unlabeled data. Semi-supervised classification differs from supervised classification in that additional unlabeled data is available for the algorithm to use in model construction.[13]. Krogel and Scheffer [42], for example, experiment with using unlabeled data to augment experiments on KDD Cup data, and Ssva [48] uses unlabeled data to enhance a support vector machine.

In this chapter, a two-tiered approach to semi-supervised relational classification that allows for the application of standard propositional learning algorithms to multi-relational data is described. In the first stage the relational data is propositionalised using randomly generated first-order rules, which are then converted into Boolean features, based on their coverage, as previously described in Chapter 3. The generation process tries to ensure that generated rules are likely to be useful for classification. This is done by requiring that rules cover a certain number of examples within user-specified minima and maxima, as described in Chapter 4. Alternatively, in a class-sensitive setting where class labels are actually present, rules can be selected based on their class-specific coverage in a manner similar to the "enrichment" property of stochastic discrimination (as described in Chapter 2)[38]. In either setting, all rules are transformed into Boolean attributes – generating a propositional representation for the second stage, where the resulting propositional dataset can be classified using any standard propositional classification algorithm, such as

SMO [60] or others.

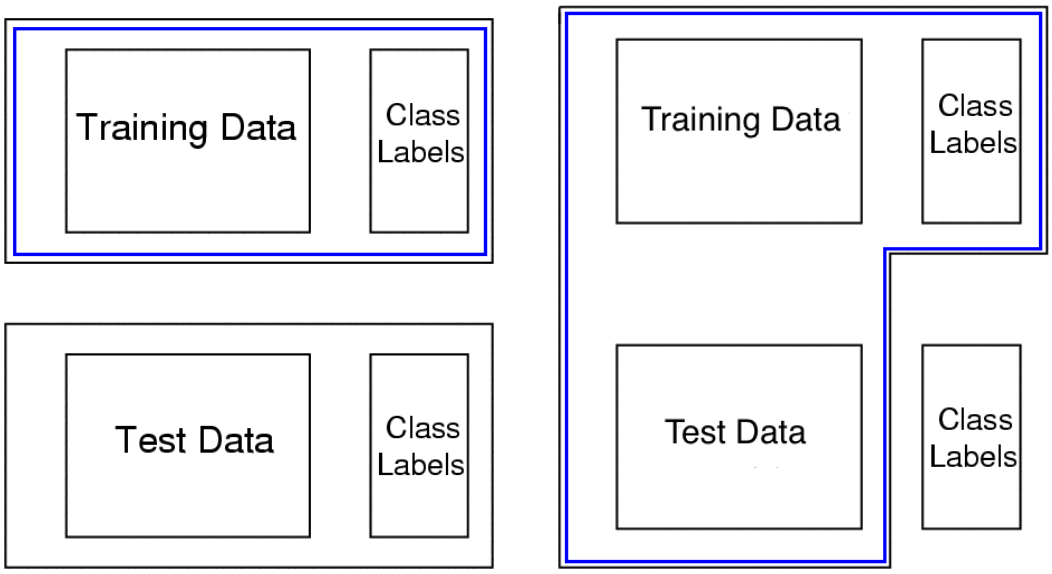
This procedure holds promise for semi-supervised learning, as one of the main explanations for the success of semi-supervised learning is the so-called cluster assumption: example clusters (or areas of high example density) tend to have similar class labels, therefore classifiers should not put decision boundaries midway through a cluster, but should cut through low-density areas instead [13]. The unlabeled data enables better estimation of cluster boundaries and can therefore also improve classification accuracy. In Chapter 4 random relational rules have been shown to work well for the clustering of relational data. Thus, their usefulness for semi-supervised learning is investigated in this chapter.

Section 5.2 describes the algorithms in more detail, Section 5.3 explains and discusses an experimental evaluation of the algorithms and finally, Section 5.4 presents a summary.

5.2 Randomised Relational Propositionalisation for Semi-supervised Learning

Unsupervised learning (such as clustering, previously discussed in Chapter 4) operates on a set of data without class labels, and looks for interesting structures in the data. On the other hand, supervised learning operates on a set of data with class labels, with the aim of finding structures in the data that map to the class labels. Semi-supervised learning falls somewhere between the two – class labels are present for some (but not all) of the data, and often the task is to determine labels for the unlabeled data.

Figure 5.1 compares supervised and semi-supervised learning graphically. The doubly-outlined sections in each diagram indicate the information available to the learning algorithm for building its model, so as shown in Figure 5.1(b), the semi-supervised learning algorithm has access to the training data and its class labels, just as the supervised algorithm does in Figure 5.1(a), but also has access to the test data (although not its class labels). The class labels for the 'unlabeled' data would be used for testing the model produced, just as the test data would be used for testing in the supervised case. The goal of semi-supervised learning is to improve beyond the model that a supervised algorithm would generate on training data by making use of the information contained in additional unlabeled data.



(a) Supervised Learning (b) Semi-supervised Learning

Figure 5.1: Comparison of Supervised and Semi-Supervised Learning

The basic RRR-P (Randomised Relational Rules – Propositionalisation) algorithm is described in Chapter 3. For this analysis of semi-supervised learning, RRR-P (for ease of differentiation denoted by RRR-P(CS) in this chapter) and a variation on that algorithm called RRR-P(SSS) were used, differing based on their determination of rule acceptability. It should be noted that RRR-P(SSS) is only a minor modification to RRR-P(CS).

- Semi-supervised Class-sensitive - generating rules on the full dataset (labeled and unlabeled), requiring enrichment on the labeled data only but rejecting rules that cover all or none of the unlabeled data – RRR-P(SSS)
- Standard Class-sensitive - generating rules only on the labeled training data with enrichment as the criterion – RRR-P(CS)

In addition, two variations on the class-blind propositionalisation algorithm described in Chapter 4 were used.

- Semi-supervised Class-blind - generating rules on the full dataset (labeled and unlabeled) with coverage-range as the criterion – RRR-P(SSB)
- Standard Class-blind - generating rules only on the labeled training data, again with coverage-range as the criterion – RRR-P(CB)

To ensure that the generated rules allow for classification, constraints are imposed on the generation process. For class-blind rule generation, only rules

are accepted that cover more than a user-defined minimum number of instances, and also cover less than a user-defined maximum, as previously used for clustering in Chapter 4. This prevents both overly specific and overly general rules. For class-sensitive rule generation, rules are required to be 'enriched', as in Chapter 2. In addition to these requirements, uniformity of coverage (again as in Chapter 2) is used for both class-blind and class-sensitive rule generation.

The algorithm for RRR-P is given in Algorithm 13, and that for class-blind RRR-P in Algorithm 14. The four variants (given in Algorithms 15-18 and illustrated in Figures 5.2-5.5) differ with respect to coverage constraints and data accessible for rule generation.

Algorithm 13 Pseudocode for the class-sensitive RRR-P algorithm

```
while Number of rules for either class is less than the minimum do
  while Number of rules in batch for either class is less than the minimum
  do
    Generate a rule
    if Rule is acceptable with regard to coverage constraints (enrichment)
    then
      Add Rule to appropriate rule batch
    end if
  end while
  Calculate the most uniformity-preserving non-empty subset of rules in
  each rule batch
  Add those rules to their corresponding rulesets
end while
```

Algorithm 14 Pseudocode for the class-blind RRR-P algorithm

```
while Number of rules in ruleset is less than the minimum do
  while Number of rules in batch is less than the minimum do
    Generate a Rule
    if Rule is acceptable with regard to coverage constraints (minimum-
    maximum coverage) then
      Add Rule to rule batch
    end if
  end while
  Calculate the most uniformity-preserving non-empty subset of rules in the
  current rule batch
  Add those rules to the ruleset
end while
```

Algorithm 15 RRR-P(SSS) process

L: labeled training data
U: unlabeled data
Generate a propositional representation $(L+U)_p$ on the full dataset $(L+U)$, using Algorithm 13, testing coverage on L and rejecting rules that cover all or no instances in U
Apply a propositional algorithm on L_p to generate a model
Evaluate the model on U_p

Algorithm 16 RRR-P(CS) process

L: labeled training data
U: test data
Generate a propositional representation L_p using the labeled training data L, using Algorithm 13
Apply a propositional algorithm on L_p to generate a model
Apply the rules generated on L to U to produce U_p
Evaluate the model on U_p

Algorithm 17 RRR-P(SSB) process

L: labeled training data
U: unlabeled data
Generate propositional representation $(L+U)_p$ on the full dataset $L+U$, using Algorithm 14
Apply a propositional algorithm on L_p to generate a model
Evaluate the model on U_p

Algorithm 18 RRR-P(CB) process

L: labeled training data
U: test data
Generate a propositional representation L_p using the labeled training data L, using Algorithm 14
Apply a propositional algorithm on L_p to generate a model
Apply the rules generated on L to U to produce U_p
Evaluate the model on U_p

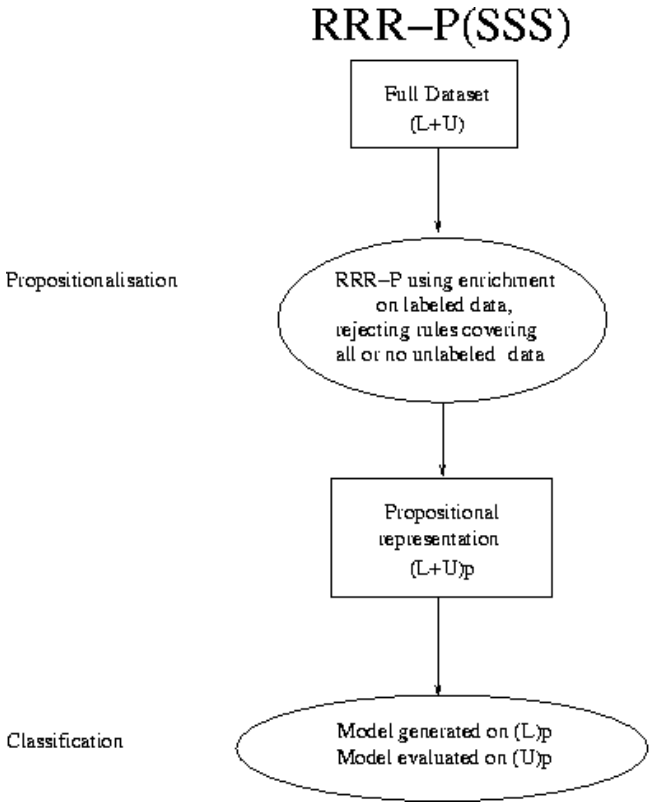


Figure 5.2: RRR-P(SSS): Semi-supervised Class-sensitive

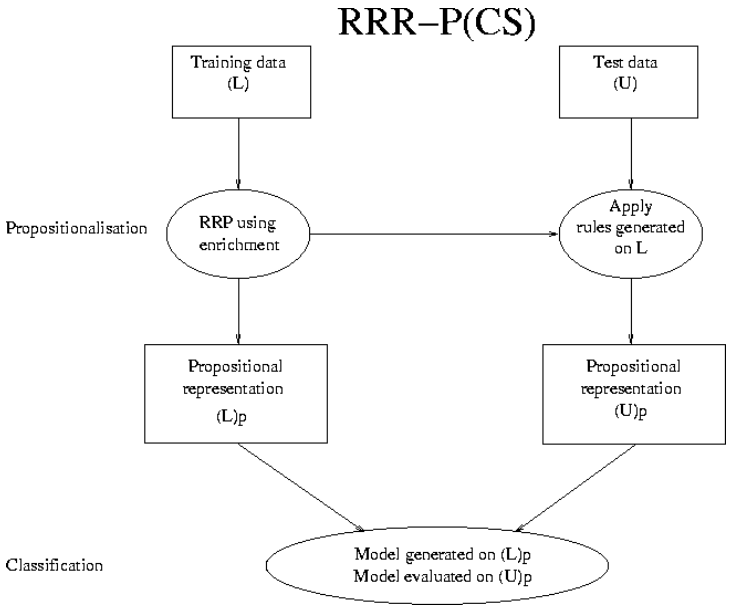


Figure 5.3: RRR-P(CS): Standard Class-sensitive

The final propositional dataset comprising solely Boolean attributes is generated by evaluating each rule on each example in the original dataset. If an example is covered by the rule, the corresponding Boolean attribute is set to

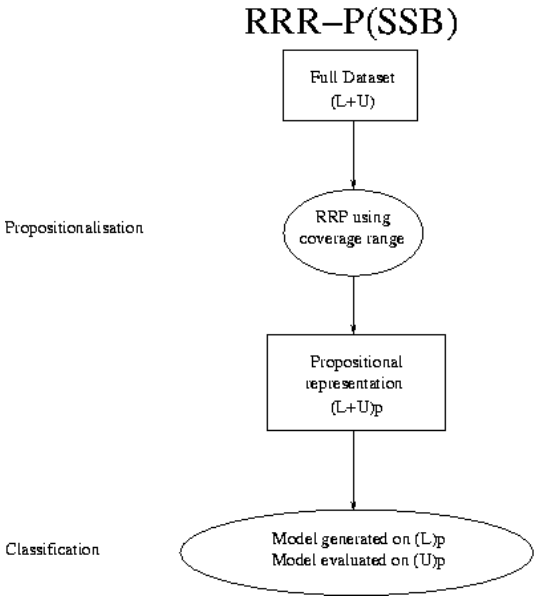


Figure 5.4: RRR-P(SSB): Semi-supervised Class-blind

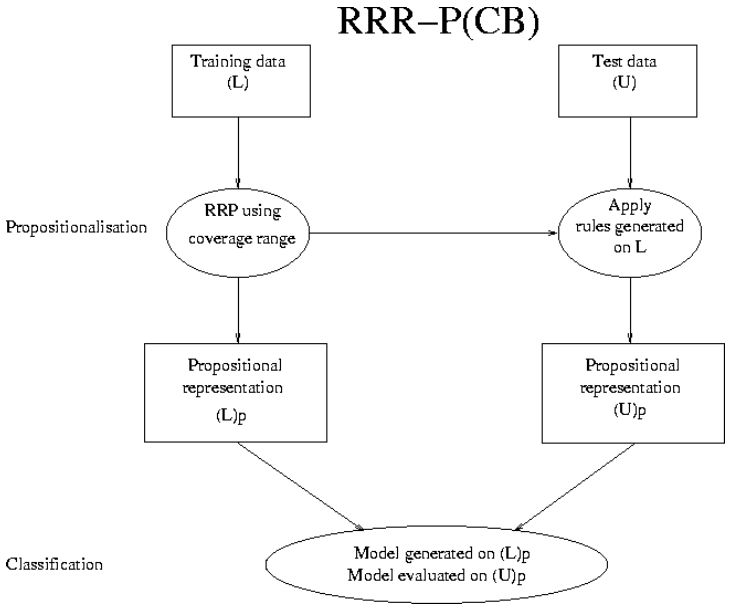


Figure 5.5: RRR-P(CB): Standard Class-Blind

true, otherwise it is set to false.

The complexity of an RRR-P variant is the sum of the complexity of both stages. Usually, when using propositionalisation in ILP, the propositionalisation stage dominates the total complexity, and this is true for RRR-P as well. Even though generating a random rule is extremely fast, its coverage still has to be determined both for checking the coverage constraints and uniformity of coverage, as well as to generate the propositional dataset. In the worst

case this coverage computation can be exponential, even for a single rule. The complexity of rule evaluation is discussed in Chapter 2. The complexity of propositional classification algorithms on the contrary is generally polynomial at worst.

5.3 Experiments

An evaluation of the four variants of RRR-P – RRR-P(SSB), RRR-P(CB), RRR-P(SSS) and RRR-P(CS) on several datasets was conducted. The following datasets were used: Mutagenesis$_{RF}$, Mutagenesis$_{All}$, Musk$_1$, Carcinogenesis, and Diterpenes. For the Diterpenes dataset, as the 'enrichment' procedure is currently limited to two-class problems, the three two-class versions (Diterpenes$_{52,3}$, Diterpenes$_{52,54}$ and Diterpenes$_{54,3}$) were used with all four algorithms, and in addition, the full 23-class dataset was used with RRR-P(SSB) and RRR-P(CB).

The resulting propositional data was classified as described in Algorithms 17-16 – using SMO [60], with the 'complexity constant' parameter determined by internal ten-fold cross-validation on the training data. The initial experiments involved random stratified 50:50 splits, i.e. 50% of the data was labeled, and 50% was unlabeled. Twenty repetitions (effectively ten two-fold cross-validation runs) were computed for each setup to produce stable average results. Linear support vector machines were used because they proved to be efficient and effective for this type of problem which comprise at most 2000 examples, but also 1000 attributes, as all setups generated 1000 random rules. Algorithms that are non-linear in the number of attributes (e.g. logistic regression) were tested but proved less effective.

For RRR-P(SSB) and RRR-P(CB), several different ranges for rule coverage were investigated, as in Chapter 4: 5%-50%, 10%-50%, 25%-50% and 25%-75%, as well as "Wide", which denotes a coverage range limited only by being required to cover at least two instances, and to not cover all instances. All proportions are relative to the size of the portion of the dataset being used for rule generation.

In the tables of results in the following sections, bold text denotes the greater result in each pair, and $\nearrow$ and $\searrow$ denote differences that are significant with 95% confidence using the standard t-test (the corrected t-test is inappropriate for these proportions of training data, so the standard t-test is used, with the caveat that it is known to overestimate significance).

Table 5.1: Accuracy for class-sensitive algorithms, 50:50 labeled:unlabeled data)

Dataset	RRR-P(CS)	RRR-P(SSS)	
Carcinogenesis	**59.09±4.29**	58.97±2.74	
$\text{Diterpenes}_{52,3}$	96.42±0.71	**96.57±0.93**	
$\text{Diterpenes}_{52,54}$	92.09±2.03	**93.20±1.48**	↗
$\text{Diterpenes}_{54,3}$	97.52±1.00	**98.21±0.81**	↗
Musk_1	**80.11±6.20**	80.00±6.52	
Mutagenesis_{All}	74.35±4.23	**74.74±4.03**	
Mutagenesis_{RF}	82.29±3.75	**83.46±3.79**	

5.3.1 Class-sensitive algorithms

A comparison of the accuracies achieved by the class-sensitive algorithms – RRR-P(CS) and RRR-P(SSS) – across the seven datasets, using 50:50 train-test splits, is given in Table 5.1 and shown in Figure 5.6. On five of the seven datasets, RRR-P(SSS) performed slightly better than RRR-P(CS), while on Carcinogenesis and Musk_1 it performed worse. A sign test across the seven datasets indicates this difference is not significant (p-value of 0.227).

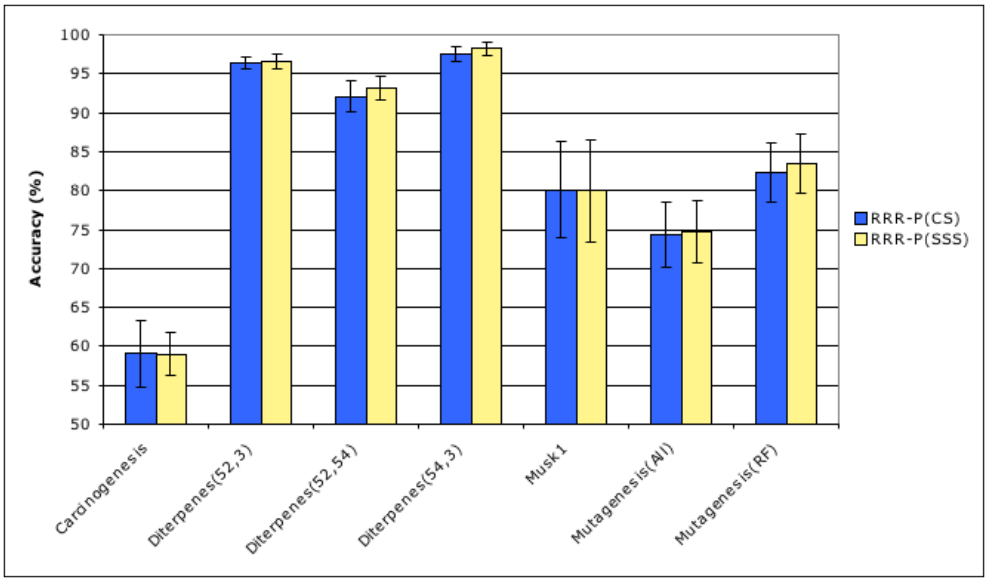


Figure 5.6: Accuracy for class-sensitive algorithms, 50:50 training-test

The extra information available to RRR-P(SSS) appears to improve the accuracy of the classifier to a minor degree in some cases. A further experiment was conducted, again comparing RRR-P(CS) and RRR-P(SSS). However, this experiment used 40 stratified 25:75 splits, with 25% of the data being labeled,

Table 5.2: Accuracy for class-sensitive algorithms, 25:75 labeled:unlabeled data

Dataset	RRR-P(CS)	RRR-P(SSS)
Carcinogenesis	**57.39±3.48**	56.83±3.21
$\text{Diterpenes}_{52,3}$	**96.18±1.03**	96.17±1.00
$\text{Diterpenes}_{52,54}$	**91.93±1.55**	91.72±1.87
$\text{Diterpenes}_{54,3}$	97.23±1.15	**97.29±0.94**
Musk_1	**70.98±7.02**	70.40±7.34
Mutagenesis_{All}	**70.24±4.49**	70.04±3.89
Mutagenesis_{RF}	76.88±3.81	**77.54±3.49**

as the benefit gained from the additional unlabeled data should become more apparent as the proportion of labeled data decreases. This method is equivalent to a form of 'inverted' four-fold cross-validation – where standard cross-validation uses one fold of the data for testing and the rest for training in each repetition, here one fold is used for training and the rest for testing. This gives four train-test splits per cross-validation, and 40 for ten such cross-validation runs. The results of this experiment are shown in Table 5.2 and Figure 5.7. The absolute accuracy is lower, due to the smaller amount of training data, but RRR-P(SSS) now only shows a slight advantage over RRR-P(CS) on two of the seven datasets. As with the 50:50 splits, the sign test across the datasets indicates the difference is not significant (p-value 0.227).

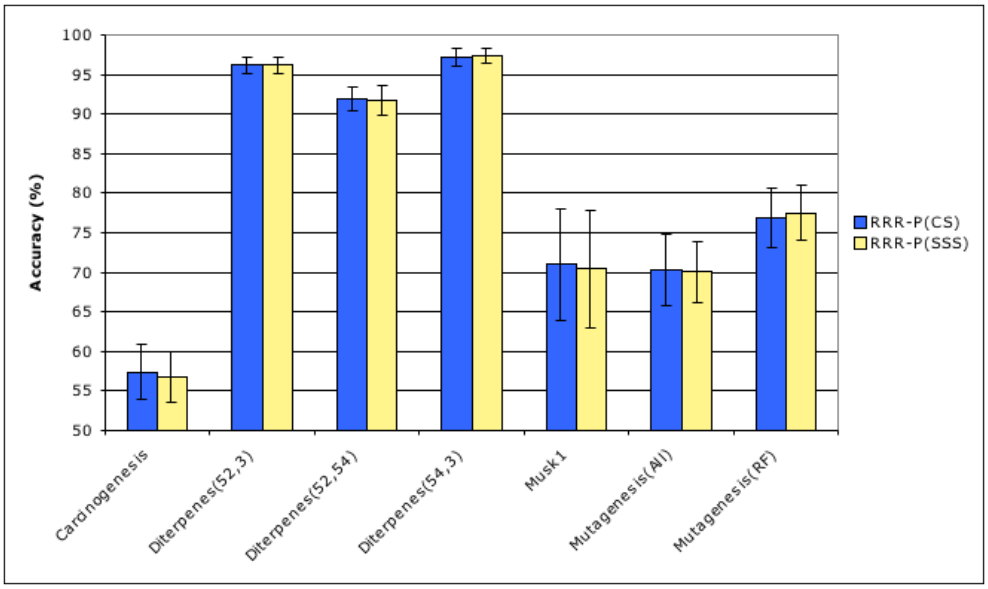


Figure 5.7: Accuracy for class-sensitive algorithms, 25:75 training-test

To investigate the result of using an even smaller amount of labeled data, an

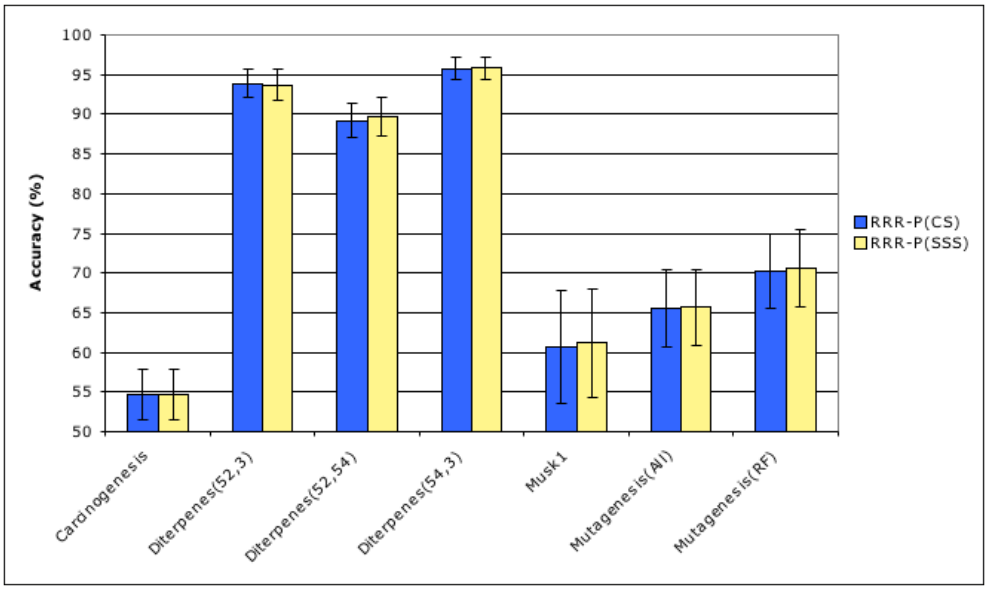


Figure 5.8: Accuracy for class-sensitive algorithms, 10:90 training-test

Table 5.3: Accuracy for class-sensitive algorithms, 10:90 labeled:unlabeled data

Dataset	RRR-P(CS)	RRR-P(SSS)	
Carcinogenesis	54.69±3.11	**54.70±3.11**	
$\text{Diterpenes}_{52,3}$	**93.89±1.79**	93.72±1.88	
$\text{Diterpenes}_{52,54}$	89.22±2.19	**89.65±2.42**	↗
$\text{Diterpenes}_{54,3}$	95.72±1.38	**95.86±1.43**	
Musk_1	60.69±7.09	**61.15±6.77**	
Mutagenesis_{All}	65.60±4.86	**65.65±4.77**	
Mutagenesis_{RF}	70.29±4.69	**70.52±4.88**	

experiment was conducted using 100 stratified 10:90 splits – 'inverted' ten-fold cross-validation. The results are shown in Table 5.3 and Figure 5.8. Here it can be seen that RRR-P(SSS) once again outperforms RRR-P(CS), achieving higher accuracy on six of the seven datasets. The sign test across the datasets gives a p-value of 0.063, indicating a fairly strong likelihood that the probability of the semi-supervised method outperforming the supervised method is greater than 0.5.

Overall, however, the differences in accuracy between the two methods are generally very small – nine of the 21 differences are less than 0.2%, and only two are greater than 1%. Only three of the differences are significant by standard t-test (although all are in favour of RRR-P(SSS)). A sign test across all 21 dataset/training-set-size combinations favours RRR-P(SSS), with a p-value of 0.194, indicating that it is likely that RRR-P(SSS), on average, has slightly better accuracy than RRR-P(CS).

5.3.2 Class-blind algorithms

The accuracies attained by RRR-P(CB) and RRR-P(SSB) are displayed in Tables 5.4-5.11.

Table 5.4: Carcinogenesis

Coverage	RRR-P (CB)	RRR-P (SSB)
5%-50%	57.76±4.12	**58.55±3.54**
10%-50%	**59.79±4.03**	59.03±4.03
25%-50%	**55.76±4.10**	54.61±3.13
25%-75%	**55.42±3.38**	55.12±2.59
Wide	**61.09±3.00**	58.36±3.41 ↘

Table 5.5: $\text{Diterpenes}_{52,3}$

Coverage	RRR-P (CB)	RRR-P (SSB)
5%-50%	98.11±0.67	**98.31±0.49**
10%-50%	**98.45±0.53**	98.36±0.40
25%-50%	**98.14±0.55**	97.98±0.73
25%-75%	**98.63±0.45**	98.49±0.58
Wide	**96.43±0.91**	94.52±1.50 ↘

Table 5.6: $\text{Diterpenes}_{52,54}$

Coverage	RRR-P (CB)	RRR-P (SSB)
5%-50%	95.90±1.11	**96.07±1.01**
10%-50%	96.08±0.91	**96.22±1.09**
25%-50%	96.38±1.35	**96.93±1.04**
25%-75%	**96.92±0.73**	96.62±0.79
Wide	**93.01±1.88**	90.66±1.87 ↘

Table 5.7: $\text{Diterpenes}_{54,3}$

Coverage	RRR-P (CB)	RRR-P (SSB)
5%-50%	**98.94±0.63**	98.82±0.74
10%-50%	**98.97±0.65**	98.74±0.88
25%-50%	**98.91±0.78**	98.79±0.59
25%-75%	99.01±0.75	**99.34±0.73**
Wide	**98.15±0.67**	96.02±1.75 ↘

Firstly, on the Diterpenes datasets (Tables 5.5-5.8), it can be seen that the Wide coverage range performs poorly compared to the other coverage ranges – an example of this trend, for $\text{Diterpenes}_{52,54}$, is displayed in Figure 5.9. On

Table 5.8: Diterpenes$_{All}$

Coverage	RRR-P (CB)	RRR-P (SSB)
5%-50%	**91.08±1.19**	91.00±1.03
10%-50%	**91.29±0.98**	90.87±1.31
25%-50%	**91.12±1.01**	90.92±1.10
25%-75%	**92.03±0.99**	91.58±0.85
Wide	**82.05±2.27**	75.53±3.96↘

Table 5.9: Musk$_1$

Coverage	RRR-P (CB)	RRR-P (SSB)
5%-50%	**77.39±6.36**	77.28±6.56
10%-50%	76.09±7.23	**78.48±5.18**
25%-50%	**72.28±6.54**	71.74±6.27
25%-75%	75.00±5.84	**76.41±8.14**
Wide	**78.26±5.42**	75.43±7.13

Table 5.10: Mutagenesis$_{All}$

Coverage	RRR-P (CB)	RRR-P (SSB)
5%-50%	**72.70±4.12**	72.43±3.01
10%-50%	**73.74±3.88**	73.30±4.31
25%-50%	75.04±3.33	**77.39±3.92**↗
25%-75%	**73.87±4.02**	73.78±3.95
Wide	75.48±3.23	75.48±3.33

Table 5.11: Mutagenesis$_{RF}$

Coverage	RRR-P (CB)	RRR-P (SSB)
5%-50%	79.47±4.38	**81.06±4.28**
10%-50%	**79.89±3.63**	79.10±5.08
25%-50%	80.32±3.36	**81.44±4.57**
25%-75%	**81.60±4.67**	80.64±3.30
Wide	80.74±2.80	**83.88±4.42**↗

Table 5.12: Proportion of rules generated that cover (2 instances – 5% of instances)

Dataset	RRR-P(SSB)	RRR-P(CB)
Diterpenes$_{52,3}$	0.7204	0.6303
Carcinogenesis	0.7180	0.5452
Diterpenes$_{52,54}$	0.7104	0.6569
Diterpenes$_{54,3}$	0.7039	0.6107
Diterpenes$_{All}$	0.6961	0.6211
Mutagenesis$_{RF}$	0.6372	0.4504
Mutagenesis$_{All}$	0.6353	0.4653
Musk$_1$	0.2946	0.1280

these datasets, RRR-P(SSB) is worse than RRR-P(CB) for the Wide coverage range, but the two algorithms produce much more similar results for all other coverage ranges. In addition, both algorithms obtain substantially lower accuracies on the Wide coverage range than they do on the other coverage ranges.

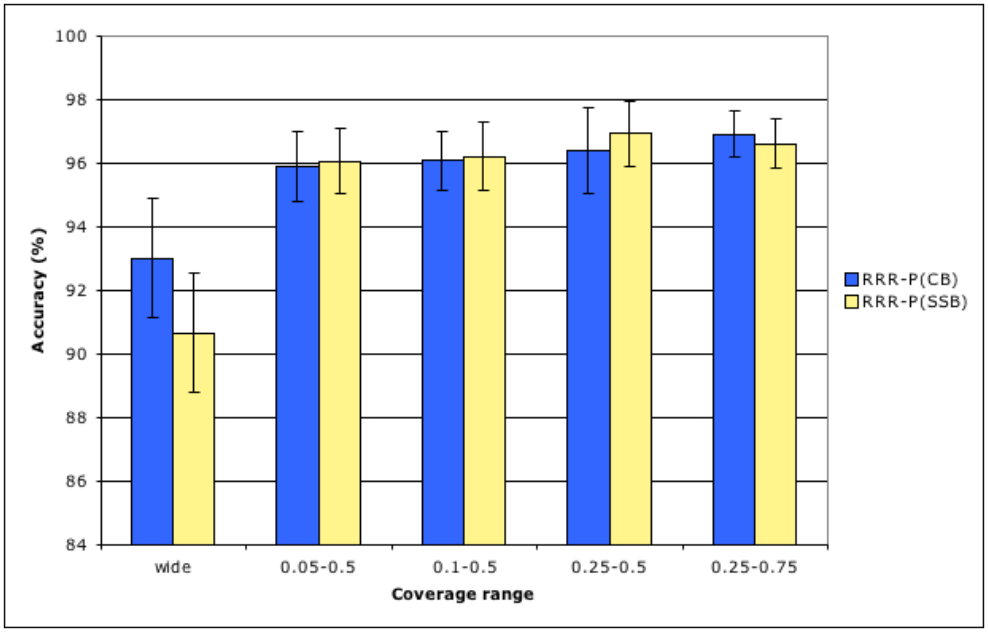


Figure 5.9: Accuracy for RRR-P(CB) and RRR-P(SSB) on $\text{Diterpenes}_{52,54}$

The poor performance of the Wide coverage range on Diterpenes was also seen in Chapter 4, and the high proportion of rules produced in the (2 instance-5%) coverage range was suggested as a possible explanation for this, as this range (being the only one using an absolute number of instances as a bound) allows rules on the Diterpenes data that cover a lower proportion of instances than those generated on smaller datasets.

Table 5.12 gives the mean proportions of rules in this coverage range for both algorithms, across all of the datasets. In all cases, the proportion of these low-coverage rules is smaller for RRR-P(CB) than it is for RRR-P(SSB). This is due to the differences in data available during rule generation. Rules produced by RRR-P(CB) have coverage bounded on the training data according to the coverage range parameters, but their coverage on the test data is not bounded, while rules produced by RRR-P(CS) have coverage bounded on the full dataset according to the coverage range parameters. For example, a randomly generated rule that covers two instances of a given dataset will always be accepted by RRR-P(SSB) using Wide coverage, but will only be accepted by RRR-P(CB) if both those instances are in the training data. If one instance

is in the training data and one in the test data, or if both are in the test data, then the two-instance coverage required by the Wide coverage range will not be met, and RRR-P(CB) will not accept the rule.

On the Diterpenes datasets, where the Wide coverage range performs noticeably worse than all other coverage ranges, RRR-P(CB) produces higher accuracies than RRR-P(SSB) and also has a lower proportion of rules in the (2 instance-5%) coverage range.

Secondly, aside from the results for Diterpenes using the Wide coverage range (discussed above), neither RRR-P(SSB) nor RRR-P(CB) is invariably superior using 50:50 train-test splits – however, a sign test across all 40 pairs of results gives a p-value of 0.011 in favour of the supervised method, indicating that it is likely that the supervised method performs better with this proportion of training data. When the number of labeled examples is sufficient to induce strong classifiers, additional unlabeled data has previously been found to be either irrelevant or even detrimental [13]. Therefore, as with the class-sensitive algorithms, experiments were performed further reducing the amount of labeled training data available. For two coverage ranges – 10%-50% and 25%-75% – propositionalisations were produced using 40 stratified 25:75 train-test splits and 100 stratified 10:90 train-test splits (the equivalent of ten inverted four-fold and ten-fold cross-validation runs respectively, as in Section 5.3.1) on each of the eight datasets. The resulting propositional datasets were classified using SMO as before. The results of these experiments are shown in Tables 5.13-5.16.

Table 5.13: Accuracy on 25:75 train-test splits, 10%-50% coverage

Dataset	RRR-P (CB)	RRR-P (SSB)
Carcinogenesis	57.05±3.10	**57.42±3.31**
$\text{Diterpenes}_{52,3}$	97.49±0.75	**97.54±0.61**
$\text{Diterpenes}_{52,54}$	94.34±1.09	**94.41±1.53**
$\text{Diterpenes}_{54,3}$	**98.17±0.96**	97.89±1.08
Diterpenes_{All}	**86.65±1.04**	86.17±1.24
Musk_1	69.02±6.30	**70.87±7.65**
Mutagenesis_{All}	**69.97±3.81**	69.26±4.08
Mutagenesis_{RF}	75.96±4.02	**76.15±4.43**

Although the variance in these results is sufficiently high that most of the differences between RRR-P(CB) and RRR-P(SSB) are not statistically significant individually, it can be seen that as the amount of training data reduces,

Table 5.14: Accuracy on 25:75 train-test splits, 25%-75% coverage

Dataset	RRR-P (CB)	RRR-P (SSB)	
Carcinogenesis	54.53±2.62	**54.94±2.48**	
$\text{Diterpenes}_{52,3}$	97.66±0.75	**97.69±0.76**	
$\text{Diterpenes}_{52,54}$	95.15±1.16	95.15±1.12	
$\text{Diterpenes}_{54,3}$	98.37±0.75	**98.45±0.79**	
Diterpenes_{All}	**87.64±1.25**	87.00±1.48	↘
Musk_1	**67.07±7.08**	66.78±7.55	
Mutagenesis_{All}	70.99±3.46	**71.14±3.10**	
Mutagenesis_{RF}	**76.38±4.57**	76.29±4.17	

Table 5.15: Accuracy on 10:90 train-test splits, 10%-50% coverage

Datatset	RRR-P (CB)	RRR-P (SSB)	
Carcinogenesis	55.45±3.45	**55.85±3.08**	
$\text{Diterpenes}_{52,3}$	95.19±1.66	**95.51±1.61**	
$\text{Diterpenes}_{52,54}$	90.47±2.55	**91.29±1.84**	↗
$\text{Diterpenes}_{54,3}$	96.39±1.16	**96.45±1.30**	
Diterpenes_{All}	77.13±2.06	**77.73±1.89**	↗
Musk_1	57.01±7.52	**57.28±7.62**	
Mutagenesis_{All}	64.30±4.64	**65.37±5.34**	
Mutagenesis_{RF}	68.48±4.06	**70.43±5.30**	↗

Table 5.16: Accuracy on 10:90 train-test splits, 25%-75% coverage

Dataset	RRR-P (CB)	RRR-P (SSB)
Carcinogenesis	53.65±2.82	**53.99±2.53**
$\text{Diterpenes}_{52,3}$	**94.96±1.74**	94.67±1.79
$\text{Diterpenes}_{52,54}$	91.33±2.32	**91.67±2.47**
$\text{Diterpenes}_{54,3}$	96.88±1.22	**97.18±1.08**
Diterpenes_{All}	78.47±1.84	**78.73±2.03**
Musk_1	**57.44±7.26**	56.35±6.71
Mutagenesis_{All}	65.13±4.72	**65.37±5.34**
Mutagenesis_{RF}	69.56±4.66	**70.79±5.60**

Table 5.17: Comparison across all class-blind experiments

Train:test ratio	RRR-P(CB) wins	RRR-P(SSB) wins	Draws	Sign Test p-value
50:50	**27**	12	1	0.012 (CB)
25:75	6	**9**	1	0.304 (SSB)
10:90	2	**14**	0	0.002 (SSB)

Table 5.18: Accuracy on Diterpenes$_{52,54}$ with low proportions of labeled training data

Train:test ratio	RRR-P (CB)	RRR-P (SSB)
4:96	84.68 ± 3.63	**86.66 ± 3.29** ↗
2:98	77.30 ± 6.92	**81.16 ± 5.97** ↗

the proportion of cases where RRR-P(SSB) produces the higher accuracy increases. A sign test on the 25:75 train-test case indicates no significant difference (p-value of 0.304), and on the 10:90 train-test case indicates a significant difference in favour of RRR-P(SSB), with a p-value of 0.002. This is shown in Table 5.17 where the number of 'wins' for each of RRR-P(CB) and RRR-P(SSB) is compared for each of the three amounts of training data.

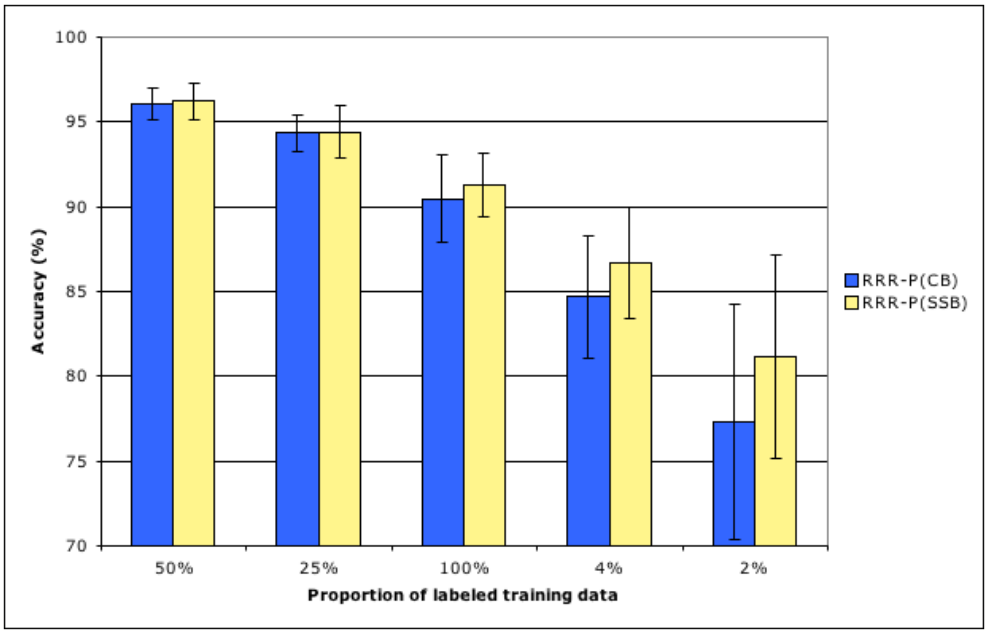


Figure 5.10: Accuracy for RRR-P(CB) and RRR-P(SSB) on Diterpenes$_{52,54}$

Indeed, a further experiment was conducted on Diterpenes$_{52,54}$ using the 10%-50% coverage range, with 250 4:96 training-test splits and with 500 2:98

training-test splits (ten inverted twenty-five-fold and fifty-fold cross-validation runs), that continued to display this trend. The results are shown in Table 5.18, and indicate that as the amounts of labeled training data decrease, RRR-P(SSB) shows an increasing advantage.

To further illustrate this trend, Figure 5.10 shows the accuracy for RRR-P(CB) and RRR-P(SSB) on $\text{Diterpenes}_{52,54}$, with each of the five tested proportions of labeled training data.

5.4 Summary

This chapter has described a two-tiered approach to semi-supervised relational learning, based on randomised propositionalisation and an arbitrary propositional classification algorithm, and compared the results to standard train-test learning. The experimental results indicate that additional unlabeled data can be beneficial to classification. RRR-P(SSS) and RRR-P(CS) are quite similar in terms of accuracy, though RRR-P(SSS) does display a slight advantage overall.

The usefulness of the extra information gained from semi-supervised learning in the class-blind case depends strongly on the percentages of labeled training data available. For smaller percentages of labeled training data the semi-supervised approach RRR-P(SSB) shows a small but consistent advantage over the corresponding standard learning algorithm RRR-P(CB).

Chapter 6

Random Forests

As previously described in Chapter 1, a decision tree is a predictive model with a tree structure, in which the internal nodes represent features and the leaf nodes represent classifications [61]. In the building of the tree, a feature in the feature space is selected for each internal node – for ID3, for example, the criterion used to select this feature is information gain.

Ensemble methods combine the individual outputs from a set of classifiers to predict values for new examples [19]. They are useful because their predictions are often more accurate than those of the individual classifiers that they are formed from, as long as those individual classifiers are diverse – that is, they make different errors on test examples. A random decision forest [30] is an ensemble of decision trees, in which diversity is achieved by building each tree on a random subset of the attributes of the training data, but the feature selection for internal nodes is still deterministic.

A random forest [11] is an ensemble of decision trees that is generated by bagging [10] the training data, and in which the feature selection for the internal nodes is also randomised. The random feature selection is accomplished by restricting the range of possible input variables to split on. A random subset of the possible variables is selected, and the best test, with regard to homogeneity of the resulting leaves, deterministically selected from that subset. More information on decision trees and random forests in general can be found in Chapter 1 – this chapter focuses on relational random forests.

In this chapter, an approach to random forests using randomly generated relational rules as splitting conditions for tree nodes is described. Section 6.1 discusses the construction of the random forests. Section 6.2 discusses the complexity of the forest construction. Section 6.3 gives the results of empirical experiments using this algorithm, and Section 6.4 discusses the results

obtained by varying methods of leaf node splitting. Section 6.5 discusses the diversity of the forests produced and Section 6.6 compares the results obtained by random forests to those produced by static propositionalisation. In Section 6.7 a semi-random method for selecting splitting rules is tested, and Section 6.8 summarises the chapter.

6.1 Forest Construction

Random Relational Rules are applied to random forests by using randomly generated rules as a splitting condition. The rules are generated as described in Chapter 2, although the enrichment and uniformity requirements no longer apply – instead, the rule is simply required to discriminate on the training data (covering neither all nor none of the training instances). As the rule generation process is independent of the current tree state it is straightforward to parallelise tree and indeed forest generation. The procedure for this Random Relational Forest algorithm (Random Relational Rules - Random Forests, or RRR-RF) is given in Algorithm 19.

Usually cover computation is the most time-consuming operation a relational learner needs to perform. This costly operation is executed exactly once for each random rule on the full dataset, and then every node on the waiting list can efficiently check whether the current rule actually properly splits its subset of the full data. This way all nodes of all trees of the ensemble can be grown in parallel. Clearly this operation would lead to identical trees, if all trees were to be started simultaneously on the full dataset. To introduce the diversity necessary for good ensemble performance the algorithm staggers the start of individual trees. Furthermore root nodes can be initialised by either the full training set or by drawing bootstrap samples of the full training set. Yet more options inducing more diversity will be discussed and evaluated in the next section. Once nodes are initialised, and are not class-pure, they are put onto a list and will wait for a rule that will split their data into two non-empty sets. Nodes that are not split within a user-defined maximum number of rules being generated (MFC, or maximum fail count) will be turned into a leaf which will predict an appropriate class distribution.

Figures 6.1 to 6.3 show three stages of forest construction, with the trees designated by letters, and internal nodes marked with an identifier corresponding to the rule that split them. Nodes are described by the tree designation followed by a numeric identifier. The training data consists of 20 instances, 10

Algorithm 19 Pseudocode for the RRR-RF algorithm

```
Initialise contents of first root node
Add first root node to list of open leaves
while Number of open leaves > 0 do
  Generate a Rule
  if the generated rule discriminates on the training data then
    for all Open Leaves do
      if Splitting the leaf using the rule produces two non-empty leaves
      then
        Create two children for the leaf according to the rule coverage
        for all Children do
          if Contents of child are not all of one class then
            Add child to the list of open leaves
          end if
        end for
        Remove current node from list of open leaves
      else
        Increment the Fail count for the leaf
        if Fail count for leaf = Maximum Fail Count then
          Remove current node from the list of open leaves
        end if
      end if
    end for
    if Number of initialised root nodes < Maximum number of trees then
      if one or more trees were modified then
        Initialise the next root node
        Add the newly initialised root node to the list of open leaves
      end if
    end if
  end if
end while
```

each of two classes. The class distribution at each node is given to the node's left. Figure 6.1 shows the state of the forest after the first rule, Rule R_1, has been added. Initially, the root node of Tree A, A_1, was the only node on the Open Leaf List. Now, A_1 has been split, and two leaves (A_2 and A_3) created. A_2, A_3 and the root node of Tree B (B_1) have been added to the Open Leaf list and A_1 has been removed from the list.

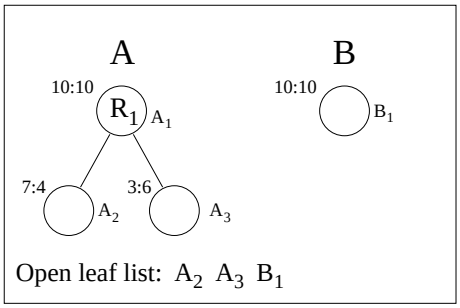


Figure 6.1: Example of RRR-RF Forest Construction, Stage 1

Figure 6.2 shows the state of the forest after another rule, R_2 has been processed. Both A_2 and A_3 have been split, B_1 has been split, and most of the new leaves thus created (A_5 through A_7 and B_2 through B_3), along with the root node of Tree C, C_1, have been added to the Open Leaf list. A_4 now contains instances of only one class (denoted by the double circle) and so was not added to the Open Leaf list, and will never be split.

Figure 6.3 shows the state of the forest after the third rule, R_3 has been processed. Nodes A_5 and A_6 have been split, adding A_8 through A_{11} to the Open Leaf list. Node A_7 has not been split by R_3, and therefore had its Fail count incremented. The maximum fail count is set to 1 (to keep this example

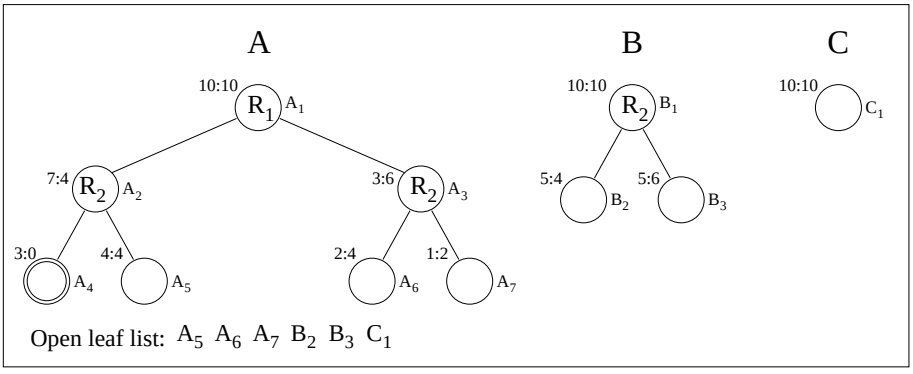


Figure 6.2: Example of RRR-RF Forest Construction, Stage 2

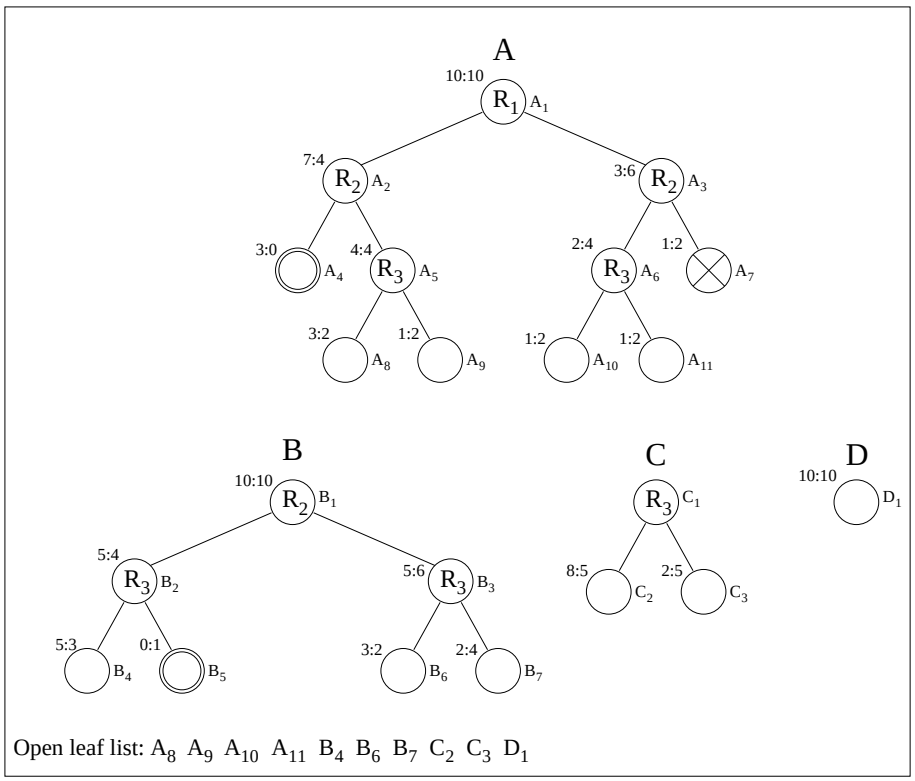


Figure 6.3: Example of RRR-RF Forest Construction, Stage 3

concise), and thus A_7 (marked by the crossed circle) will now be removed from the Open Leaf list and thus will never be split. Nodes B_2 and B_3 have been split, producing B_4 through B_7, of which B_4, B_6 and B_7 will be added to the Open Leaf list, while B_5 will not, due to it being class-pure. C_1 has been split, producing C_2 and C_3, and the root of tree D, D_1, has also been added to the Open Leaf list.

Predictions for test instances are computed as simple averages over the class-distributions returned by all trees in the forest, as is common for Random Forests or Bagging in general. This procedure is described in more detail in Algorithm 20.

RRR-RF differs from FORF [3], another algorithm that produces relational random forests, in two main ways. First of all FORF does not use full rules in every node, but in contrast paths from the root to each leaf comprise rules. As logical variables can only be shared across positive paths, this complicates both generation and interpretation of such trees. Secondly, like Breiman's original random forest, FORF randomly restricts the set of possible tests (features) and then picks the best test from that restricted set. RRR-RF on the other hand

Algorithm 20 Classification procedure for a test instance in RRR-RF

```
for all Trees in the forest do
    Traverse the tree with the instance
    Return the proportion of instances that are of class A at the leaf node
end for
if The mean of the proportions returned > 0.5 then
    classify the instance as being of class A
else
    classify the instance as being of class B
end if
```

uses a fully self-contained randomly generated relational rule as a test. As a consequence, RRR-RF can easily generate its trees in a staggered parallel fashion, with each new rule being available for all open leaves, while FORF processes both nodes and trees fully sequentially.

RRR-RF can also be seen as an example of dynamic propositionalisation [45], in that the features are generated dynamically on-demand, and do not have to be precomputed in advance as would be common in static propositionalisation [40].

6.2 Complexity of Forest Construction

When RRR-RF generates and evaluates a rule, it then applies the test derived from that rule at every open leaf in each active tree. The cost of rule evaluation is the same as for previous uses of Random Relational Rules, as described in Chapter 2. As each test is applied to all open leaves, the number of rules required to be evaluated is substantially lower than would be required if a new rule were being evaluated for each open leaf, as in standard random forests. The number of discriminatory rules (for the remainder of this chapter, discriminatory rules are simply referred to as rules) required for forest construction is heavily influenced by the number of trees in the forest and the Maximum Fail Count. A rough estimate for this value is the sum of the average number of rules required to construct a single tree and the number of trees in the forest, as when the last tree in the forest is completed, the previous trees are also likely to be complete.

$$\text{Rules required for forest generation} \approx (n + s) \tag{6.1}$$

Table 6.1: Forest size vs. number of rules generated

#Trees	Average #rules	Average #rules - #trees
10	61.79	51.79
25	78.88	53.88
100	152.01	52.01
200	251.02	51.02
500	553.29	53.29

Where:
n = the number of trees in the forest
s = the average number of rules required to construct a single tree

The estimate given by Equation 6.1 can be confirmed by experimental results. Table 6.1 shows the results for an experiment on Mutagenesis$_{RF}$ which conducted ten ten-fold cross-validation runs, recording the average number of rules generated in each fold. As the number of rules required to construct a single tree should be unaffected by the number of trees generated, the difference between the number of rules required for a forest and the number of trees in the forest should be roughly constant, regardless of forest size.

Because of the staggered fashion in which the trees are generated, each tree has access to at least one more rule than its immediate successor and so its construction has probably already finished at the time the construction of that successor finishes. Thus, when the final tree is complete, it is likely that all previous trees are complete or nearly so.

The number of rules required to construct a single tree can vary substantially. It is a function of the particular dataset, the Maximum Fail Count and the particular random rules generated. A worst case upper bound is given by:

$$\text{Maximum rules required for a single tree} = (t - 1) \times mfc \tag{6.2}$$

Where:
t = the size of the training set
mfc = the Maximum Fail Count

It is extremely unlikely that this upper bound would be reached under normal circumstances, as it describes the pathological case where each node in the tree is split only after the maximum possible number of rules have been

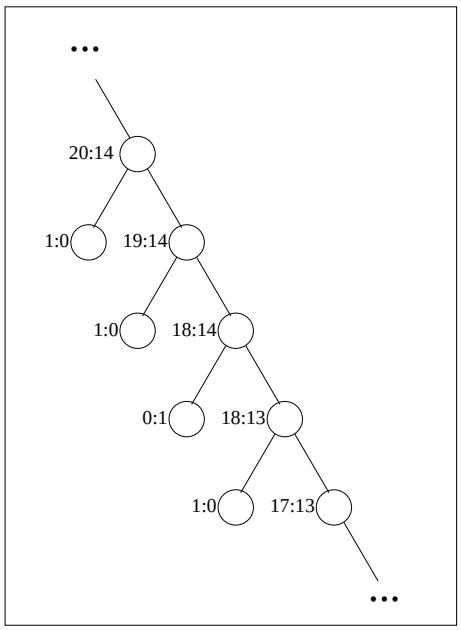


Figure 6.4: Worst-case tree construction

generated, and at every node the split has resulted in one single-instance leaf and a second leaf containing all the remaining instances (see Figure 6.4). In practice the number of rules required has always been substantially lower than the worst case upper bound.

As the number of trees in a forest increases, the proportion of useful rules that split at least one open leaf also increases. Higher MFC values, on the other hand, lower the proportion of such useful rules simply because more unproductive rules can be generated before giving up at a node. Figure 6.5 shows these trends in the proportion of globally discriminatory rules used, as Maximum Fail Count and the number of trees vary. As the number of trees increases, the proportion of discriminatory rules used also increases, as predicted, while as the MFC increases, the usage proportion decreases. A higher MFC allows more locally non-splitting rules to be seen by a leaf before it is closed, which results in the lower proportionate rule usage.

6.3 Experimental Results

RRR-RF was tested on several datasets, varying the number of trees built and the Maximum Fail Count. Three root initialisation methods were used:

- Standard - each root is initialised with the full training set of examples

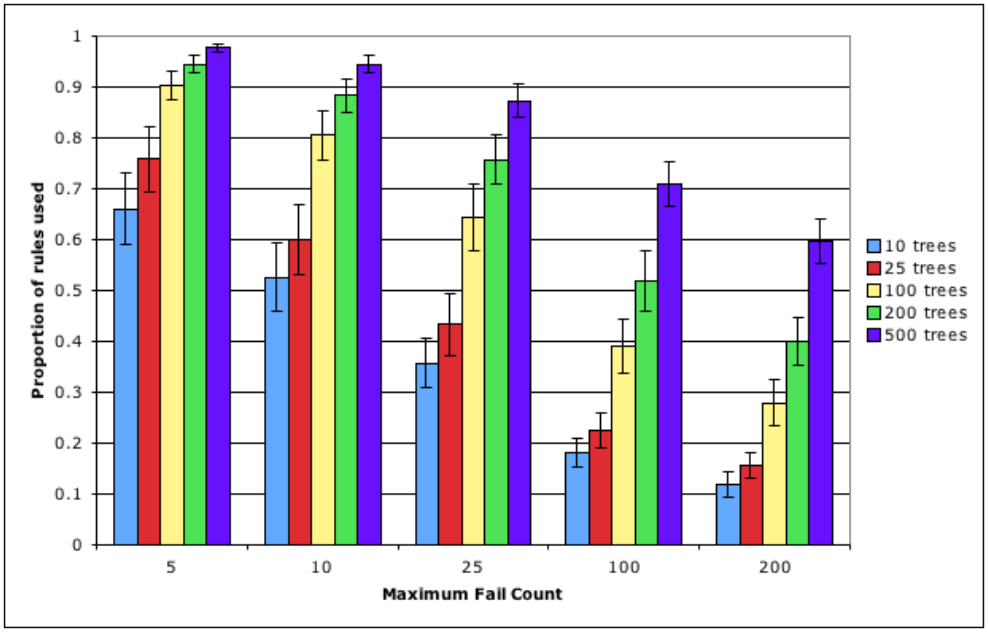


Figure 6.5: Mutagenesis$_{RF}$, rule usage

- Bagging - each root is initialised with a set of examples randomly selected with replacement from the training set

- Unique - each root is initialised as in Standard, but then may only be split by a rule that produces a split different from all previous roots. But once the root reaches its Maximum Fail Count, it will simply accept the next splitting rule, even if it is not unique.

The Unique root initialisation method is a compromise between diversity and efficiency. As described in Algorithm 19, a new root is added to the Open Leaf list for each time a rule is processed that causes the forest to change – however, unlike the Standard method, if that rule did not produce a split different from those seen in previous roots, a new root will not be split, and it is thus possible to have multiple root nodes on the Open Leaf list. Thus, when using the Unique method, each time a rule is seen that does not produce a split different from all previous roots, none of the roots on the Open Leaf list will be split, and instead their Fail Counts are incremented. If this were not the case, and a root only began to increment its Fail Count once the previous root had been split, then as unique roots became less likely to be generated (with increasing numbers of trees), each root would be increasingly likely to reach the MFC and use a previously seen split anyway. For most values of

Table 6.2: Best results for RRR-RF

Dataset	Root	Trees	Maximum Fail Count	Accuracy (%)
Carcinogenesis	bagging	500	25	61.24
$\text{Diterpenes}_{52,3}$	unique	500	200	96.83
$\text{Diterpenes}_{52,54}$	unique	500	200	94.51
$\text{Diterpenes}_{54,3}$	unique	500	200	97.83
Musk_1	standard	500	10	83.40
Mutagenesis_{All}	bagging	500	25	77.39
Mutagenesis_{RF}	unique	500	25	84.39

the MFC, this would generate many more rules than the Unique root method, while being only slightly more diverse.

This buildup of waiting roots on the Open Leaf list can result in a 'cascade' when the first one reaches its MFC. The first root will split, and the remaining ones will have their MFC incremented, resulting in the second root reaching its MFC (if the next rule it sees results in a non-unique split) and thus splitting regardless of the uniqueness of the rule, and so on. In this context, the MFC can be thought of as the maximum number of non-uniquely-splitting rules the forest can see before it begins to use rules to split roots as they are generated.

The datasets used were Musk_1, Mutagenesis_{RF}, Mutagenesis_{All}, Carcinogenesis, $\text{Diterpenes}_{54,3}$, $\text{Diterpenes}_{52,3}$ and $\text{Diterpenes}_{52,54}$.

For each combination of variable settings, RRR-RF was run ten times using ten-fold cross-validation, and the mean of the resulting accuracies taken.

Table 6.2 contains the best accuracy obtained for each dataset, and the settings that produced that result.

The results obtained for Mutagenesis_{RF}, using Standard root initialisation (accuracy, AUC, tree size and generated rules) are displayed in Figures 6.6 through 6.13. There are two figures for each of these measurements, to show more clearly the effects of both the MFC and the number of trees in the forest.

These results show that increasing the number of trees generally improves accuracy. The effect of the Maximum Fail Count parameter is less straightforward – though low values do not do well for Mutagenesis_{RF}, the highest values are not always the best, whereas for Diterpenes, the highest value of the MFC gives the best accuracy. The number of discriminatory rules generated increases with both the forest size and the MFC, and the tree size increases with the MFC. The trends shown by Mutagenesis_{RF} hold for most of the other

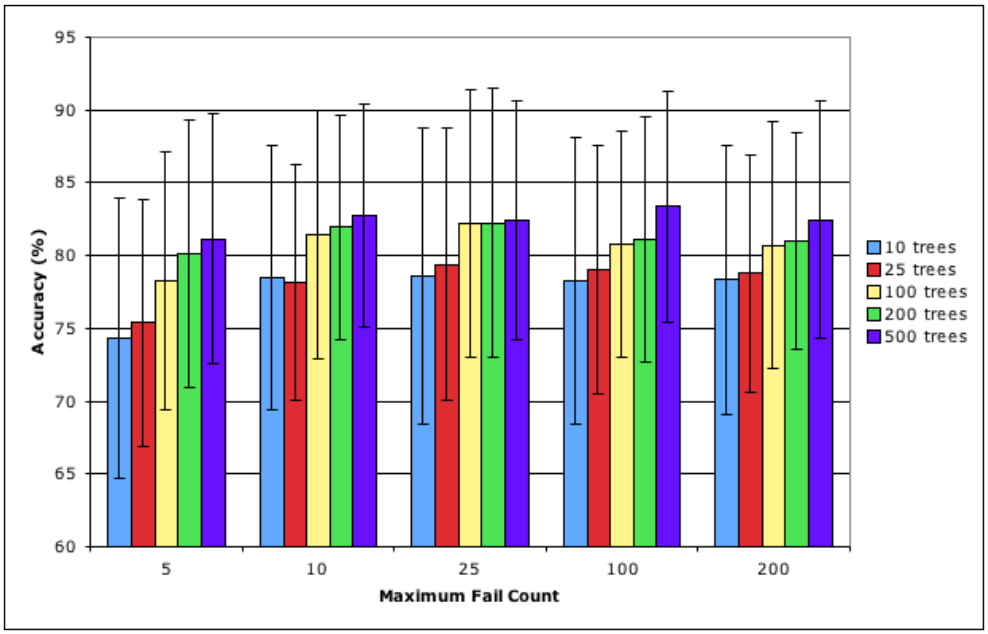


Figure 6.6: Mutagenesis$_{RF}$, Accuracy

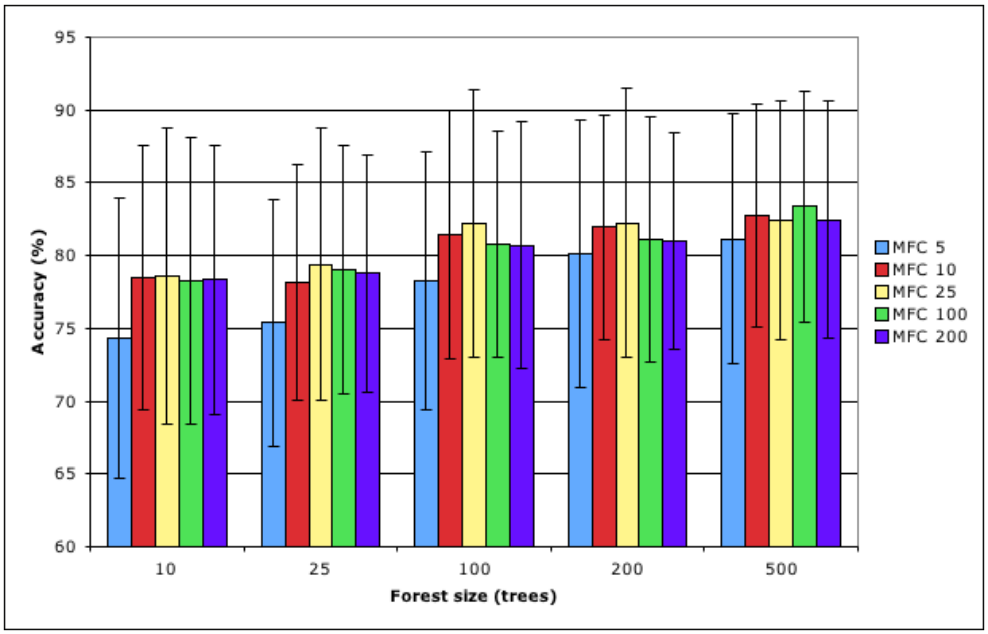


Figure 6.7: Mutagenesis$_{RF}$, Accuracy

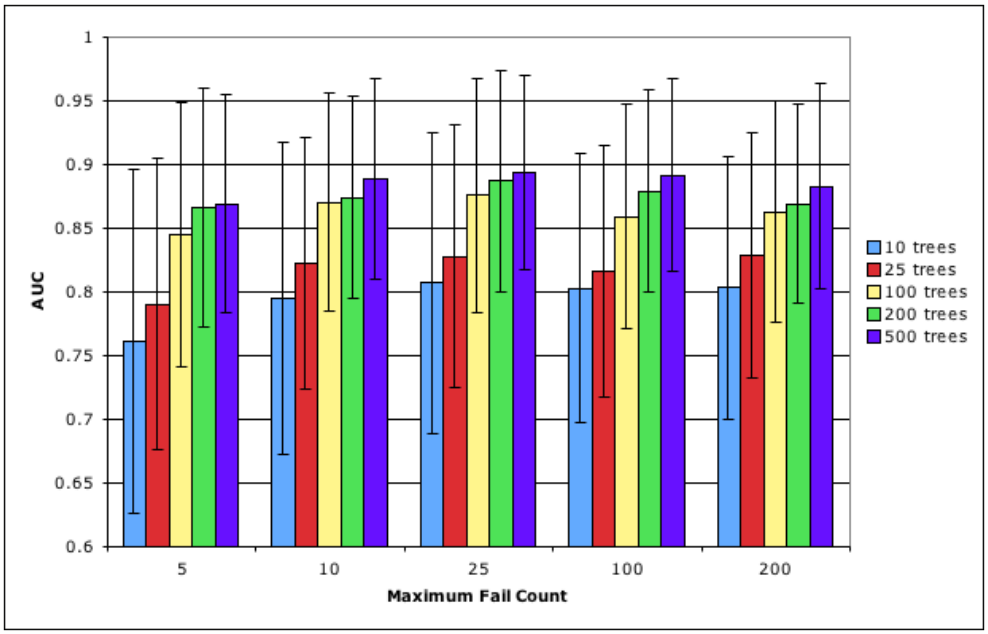


Figure 6.8: Mutagenesis$_{RF}$, AUC

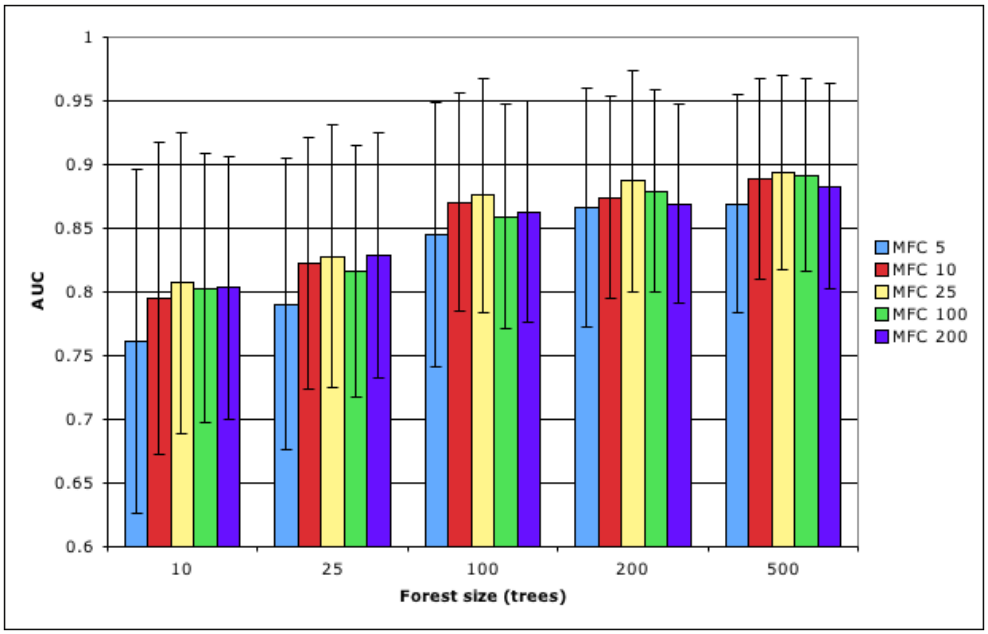


Figure 6.9: Mutagenesis$_{RF}$, AUC

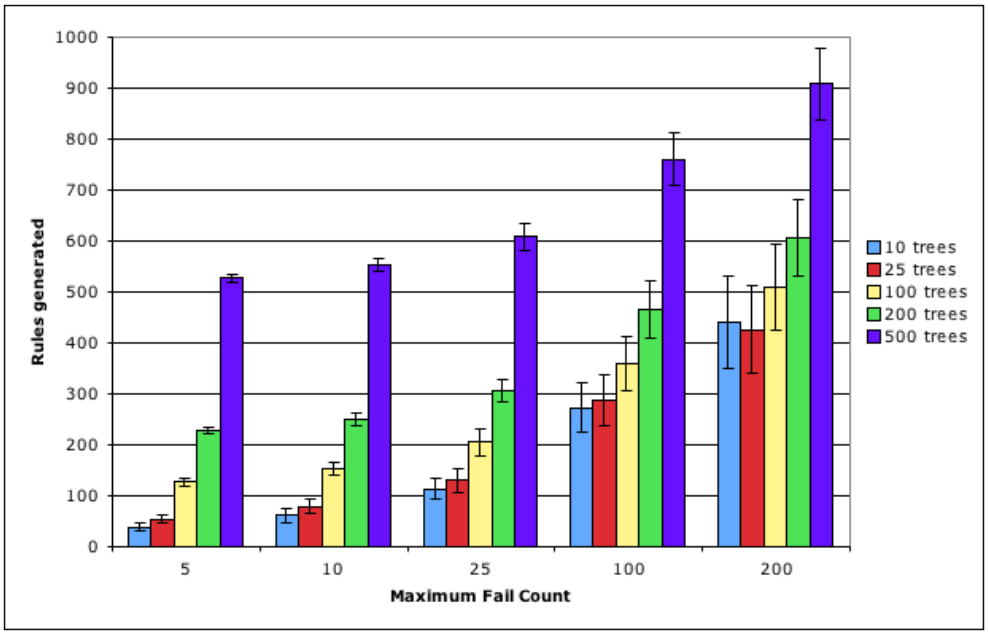


Figure 6.10: Mutagenesis$_{RF}$, Rules generated

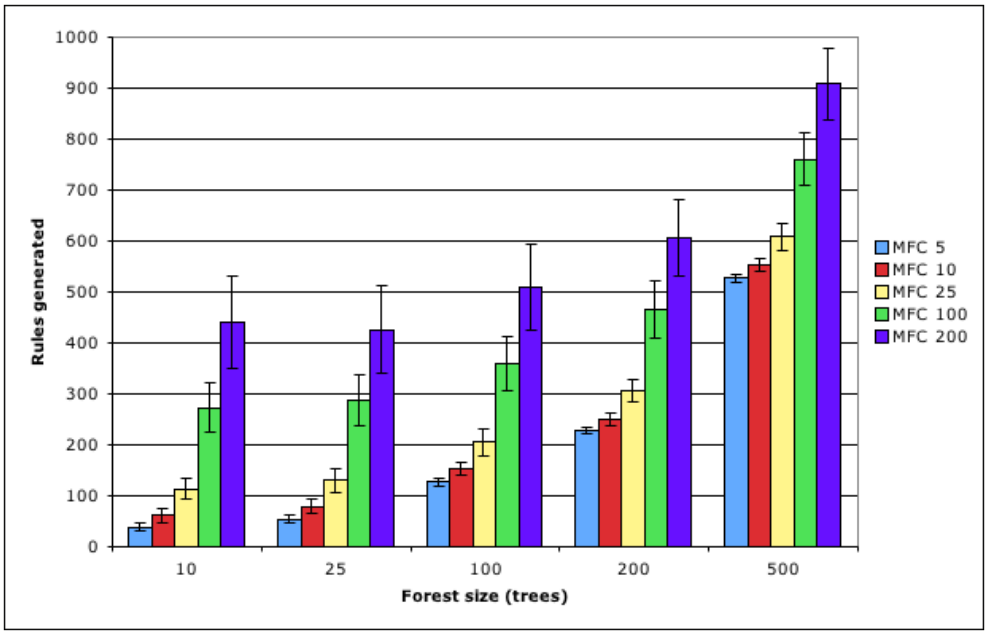


Figure 6.11: Mutagenesis$_{RF}$, Rules generated

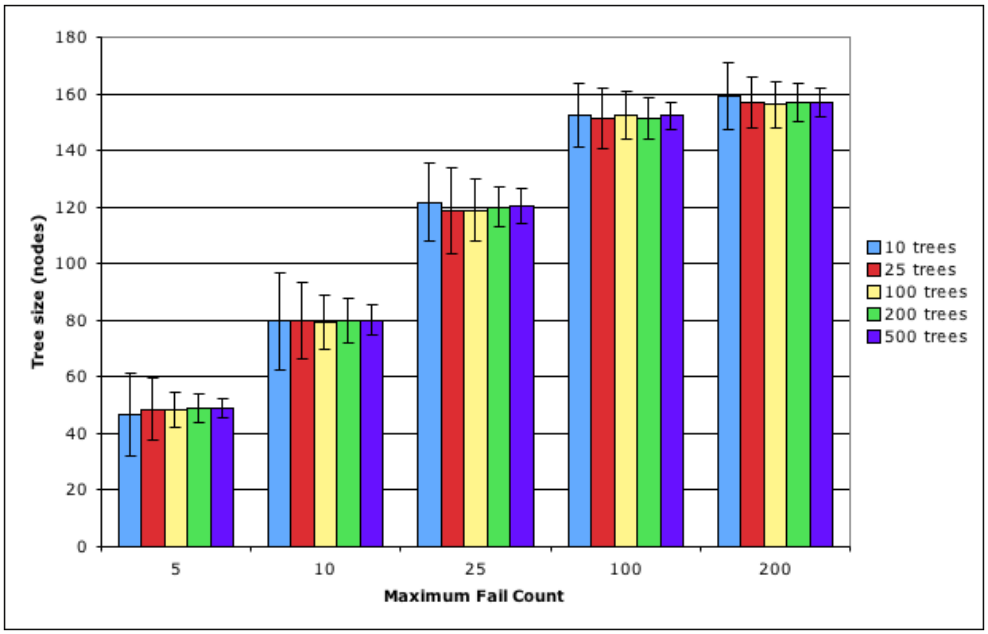


Figure 6.12: Mutagenesis$_{RF}$, Tree size

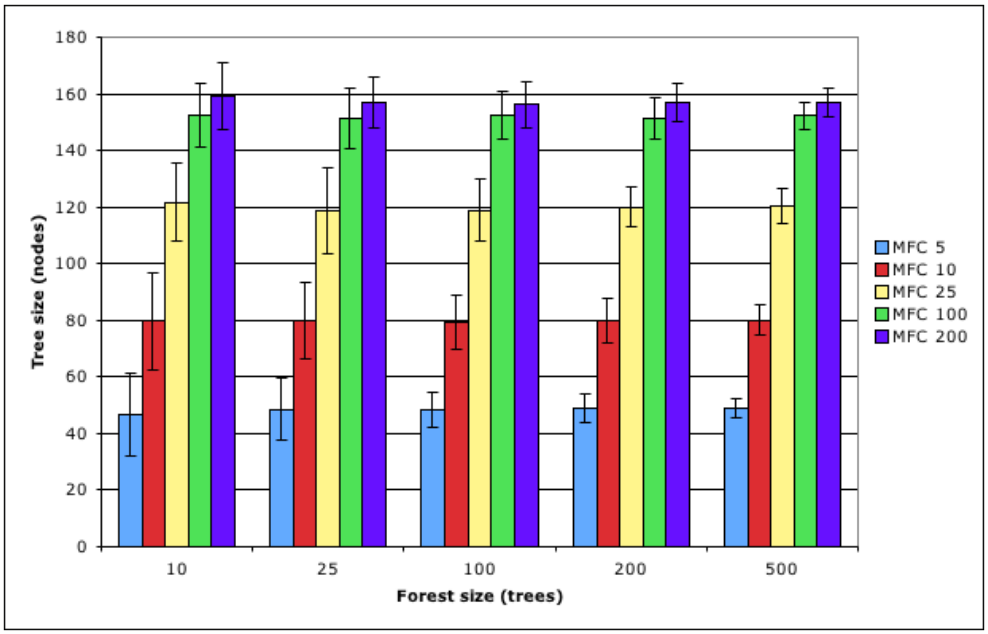


Figure 6.13: Mutagenesis$_{RF}$, Tree size

datasets as well, with some exceptions.

On Musk_1, for example, the lowest values for the MFC generally do well for accuracy and AUC by comparison to the others, while increasing MFC has only a minor effect on tree size (beyond MFC 10) and the number of rules generated, as shown in Figures 6.14 through 6.17. This is probably due to the small size of the Musk_1 dataset and the ease of differentiating between examples, so that higher MFC values have very little effect as leaves are often split after seeing a smaller number of rules.

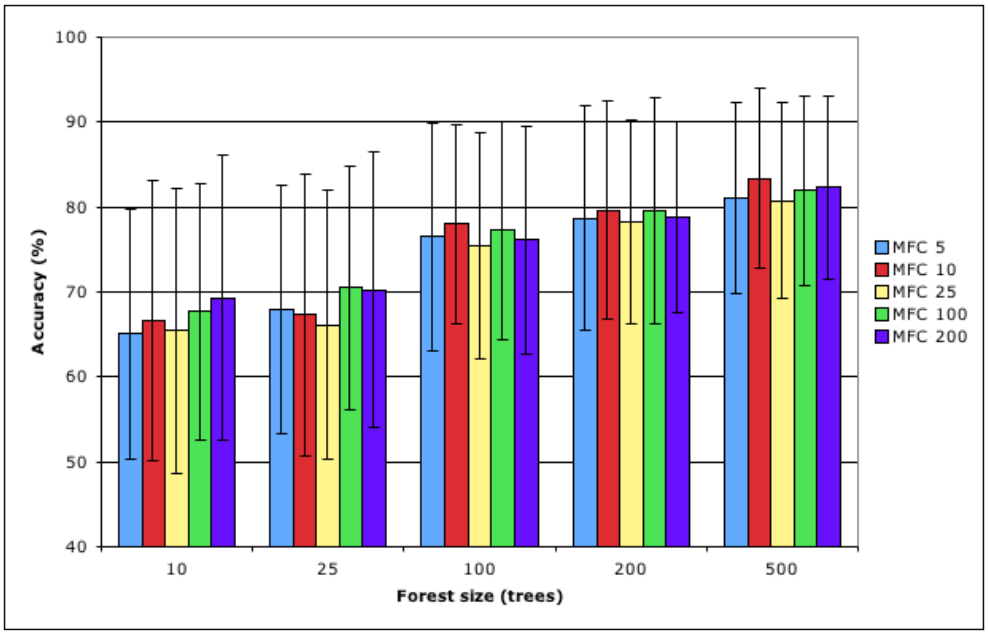


Figure 6.14: Musk_1, Accuracy

Although the number of rules generated for a forest follows the same trend across datasets, comparison between datasets shows that for low MFCs, the Diterpenes datasets require more rules to generate a forest than the other datasets, but that at higher MFC values (100 and up) both Mutagenesis datasets and Carcinogenesis generate more rules than Diterpenes. At MFC 25 (all other parameters being equal) the number of rules required for forest generation is similar across all the datasets except Musk_1, which consistently requires less rules than the other datasets at all MFC values. This appears to be linked to the difficulty of producing leaf-splitting rules, in that Mutagenesis and Carcinogenesis, on average, need to generate more rules to split a leaf than the other datasets and Musk_1 needs to generate less.

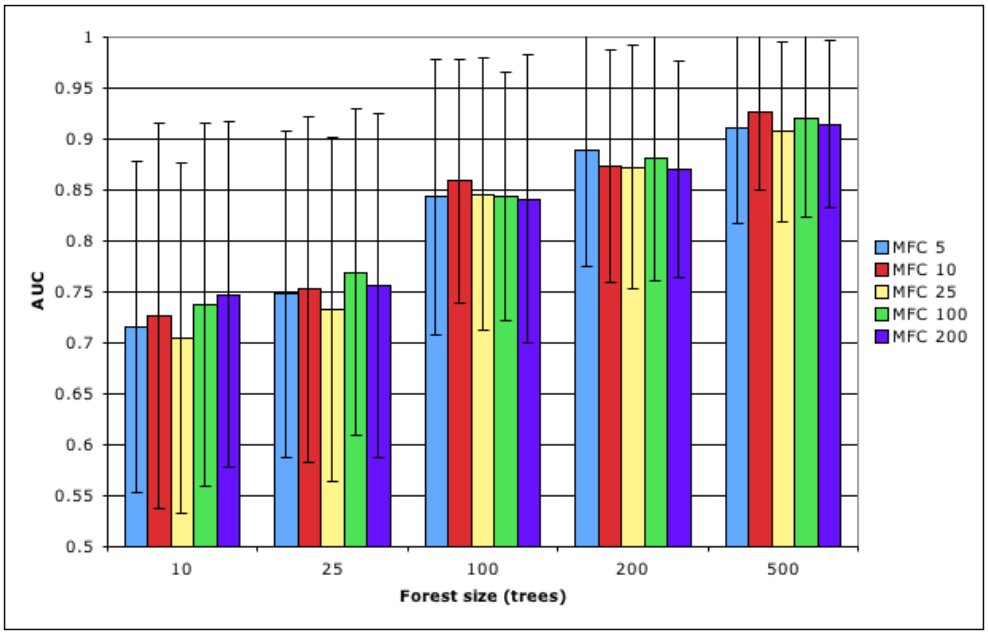


Figure 6.15: Musk_1, AUC

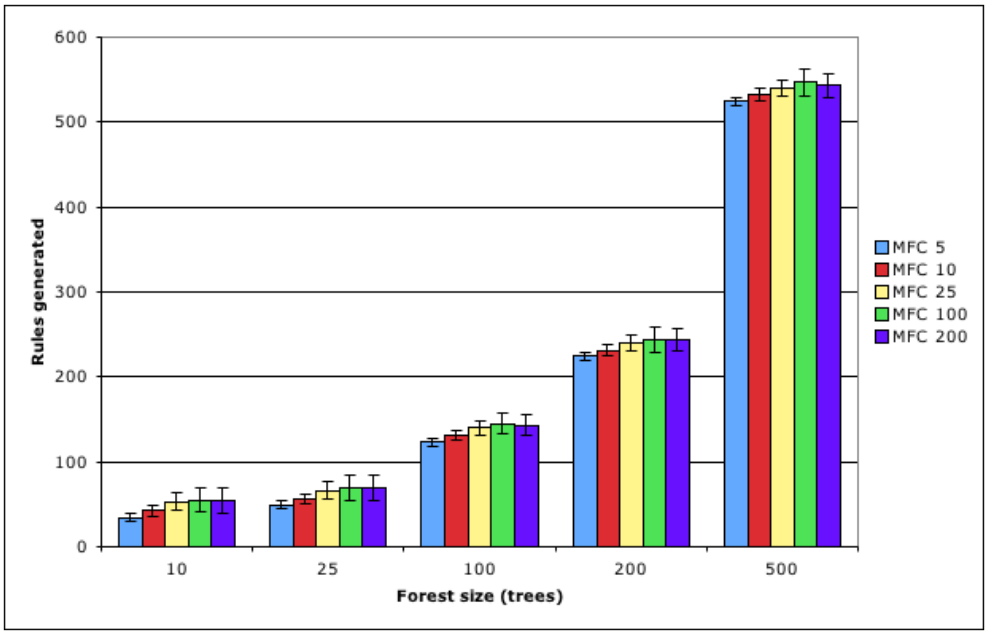


Figure 6.16: Musk_1, Rules generated

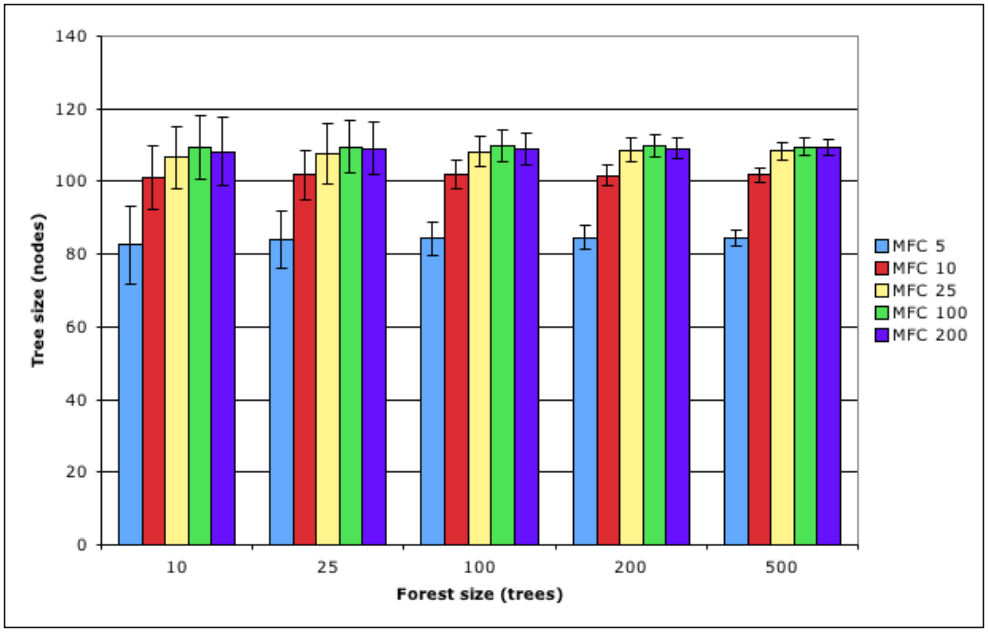


Figure 6.17: Musk_1, Tree size

At lower MFC values, the effect of the difficulty of leaf-splitting on the number of rules generated is restricted, leaving the size of the datasets to be the major factor affecting the number of rules required for forest construction, as the randomness of split selection leads to roughly binary trees, with tree sizes related to dataset size, and thus rule numbers related to tree sizes. As the MFC is increased, nodes that would have reached their MFC and become leaves for smaller MFC values will either reach the new MFC or be split to produce two new nodes. In either case, more rules will need to be generated than for lower MFCs. Mutagenesis and Carcinogenesis, which tend to require more rules to split nodes, thus require more rules for forest production at high MFCs.

When the 'Unique' root initialisation method is used, Equation 6.1 no longer provides as accurate an estimation of the number of rules required to generate a forest. As more trees are generated, the probability of a unique split decreases, and the average number of rules that are generated before the next root in the forest can split increases, resulting in an overall increase in the number of rules generated.

Bagging for root initialisation results in smaller trees, as there are fewer unique instances at the root of each tree.

The overall best results seen in Table 6.2 show that the best result is always obtained with the highest number of trees, and also that the Bagging and Unique methods of root initialisation are beneficial to accuracy. This can be further seen in Figures 6.18 through 6.24, which show the results for the three root initialisation methods for each dataset and MFC value, for a forest size of 500 trees. The error bars denote standard deviations.

For Diterpenes, Bagging is very rarely the best method, and when it is the best, it is only by a small margin. For smaller MFC values, Standard and Unique roots are fairly similar, but at the highest MFC, Unique overtakes Standard on all three datasets. For Mutagenesis$_{All}$, on the other hand, Bagging almost always produces the best result. Bagging on Musk$_1$ performs less well than the other methods at low MFC values, but is fairly similar to the other two methods for the higher MFCs (and the variance on the Musk results is quite high). For Mutagenesis$_{RF}$ the highest results are obtained by Unique and Bagging at MFC 25, and Bagging performs worst of the three methods at MFC 5, but apart from these the three methods seem to be very similar.

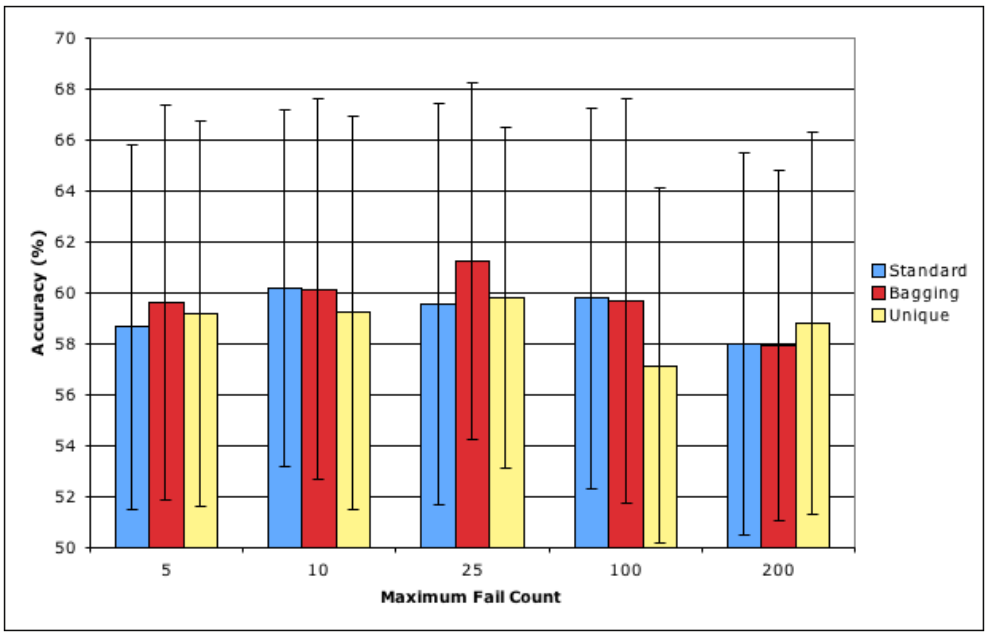


Figure 6.18: Carcinogenesis, accuracy with 500 trees

The often poorer performance of Bagging as a root initialisation method at the lowest MFC here seems to be a result of the learning curve, where changes in the amount of training data available can have a significant effect on accuracy [77] – the Bagged roots, which on the average have approximately 63.2%

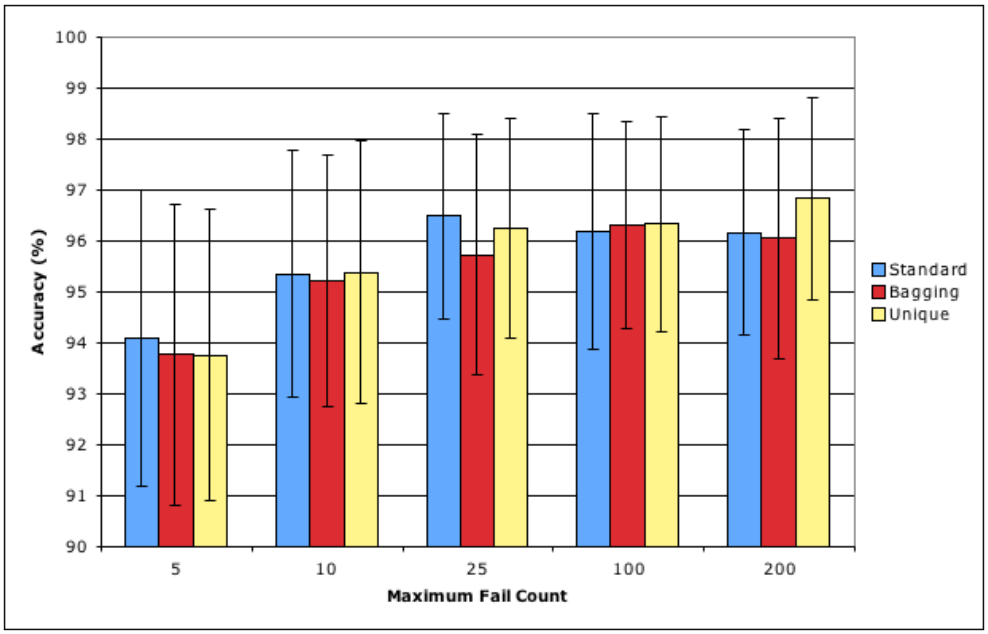


Figure 6.19: $\text{Diterpenes}_{52,3}$, accuracy with 500 trees

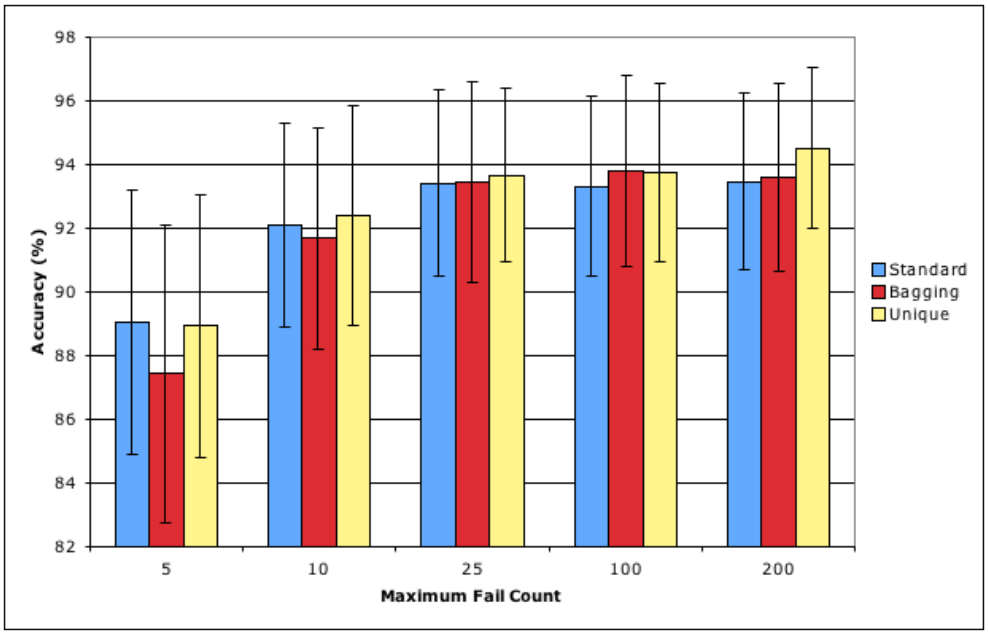


Figure 6.20: $\text{Diterpenes}_{52,54}$, accuracy with 500 trees

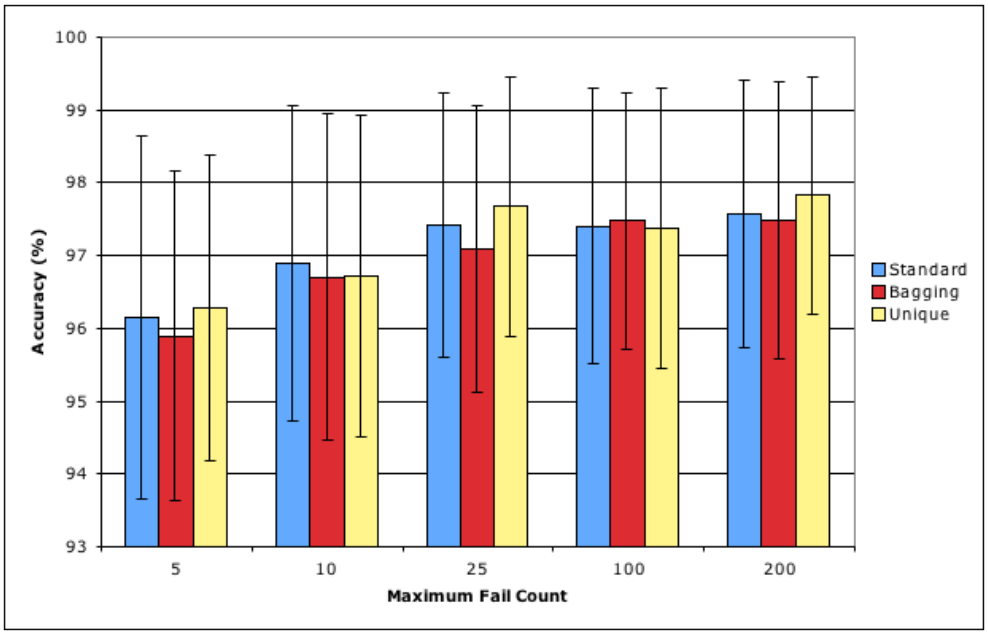


Figure 6.21: $\text{Diterpenes}_{54,3}$, accuracy with 500 trees

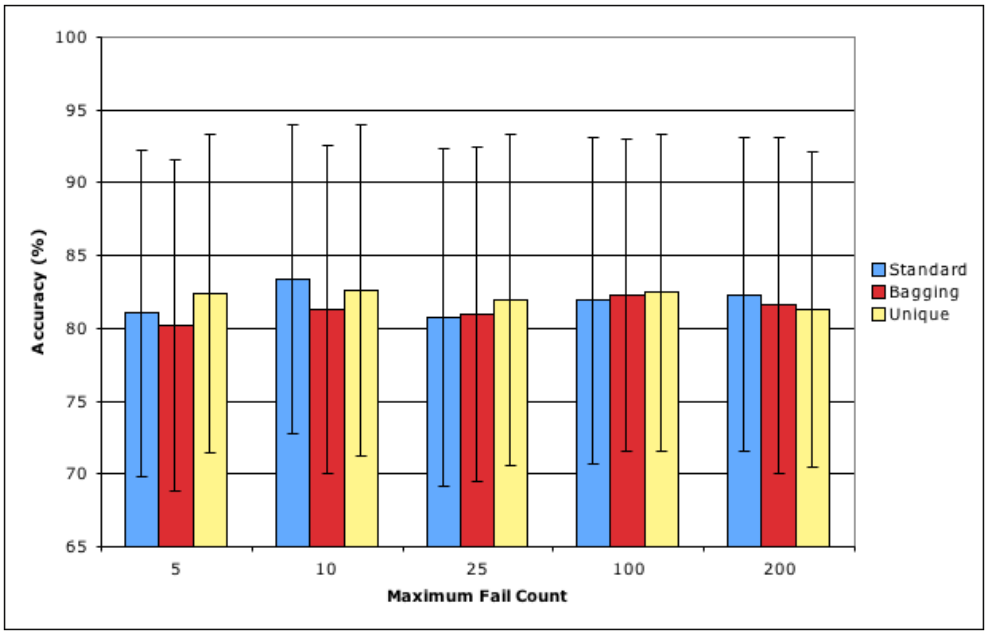


Figure 6.22: Musk_1, accuracy with 500 trees

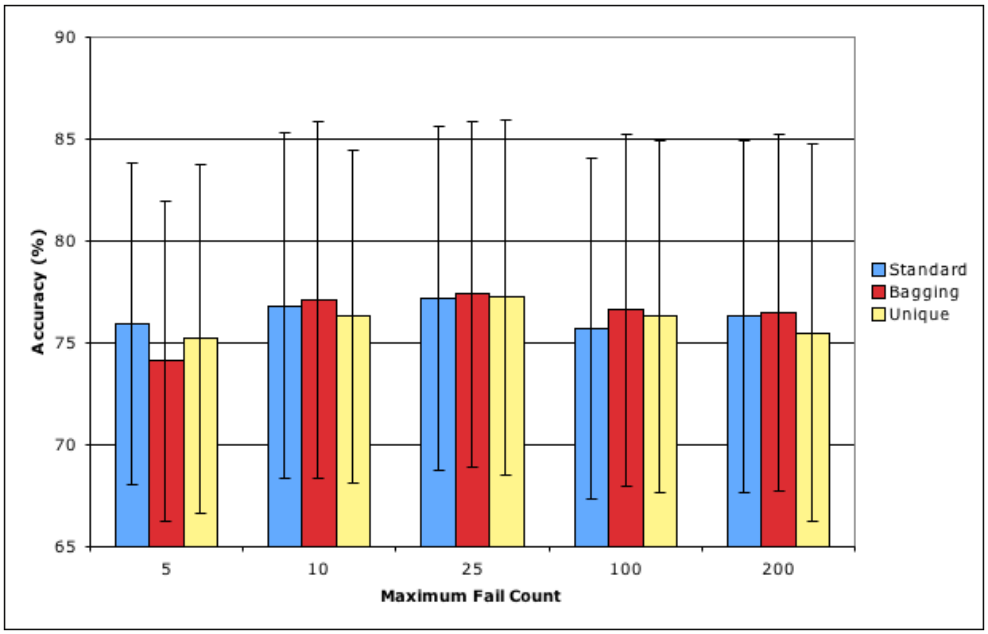


Figure 6.23: Mutagenesis$_{All}$, accuracy with 500 trees

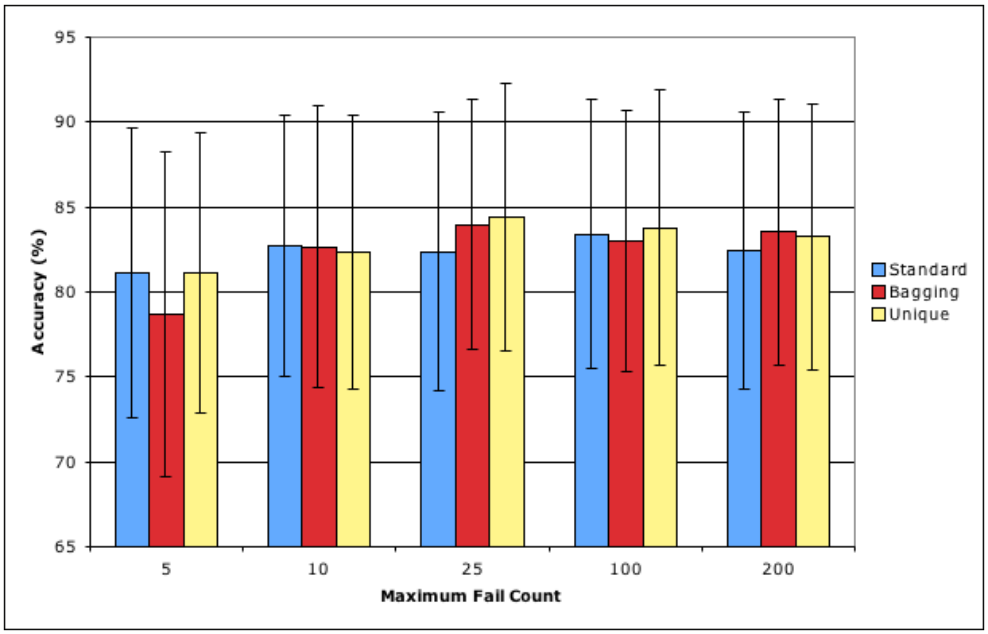


Figure 6.24: Mutagenesis$_{RF}$, accuracy with 500 trees

Table 6.3: Accuracy of individual trees in RRR-RF forests

MFC	Bagging-75	Bagging-std	Bagging-300	Unique
5	0.6964	0.7139	0.7234	0.7344
10	0.7110	0.7370	0.7448	0.7631
25	0.7119	0.7507	0.7663	0.7854
100	0.7144	0.7515	0.7786	0.7891
200	0.7140	0.7486	0.7691	0.7863

probability of containing a particular instance from the training set, have access to less of the training data than the non-Bagged roots, and this impacts the ensemble accuracy. To test this, the individual accuracies of the trees in the forests produced by ten ten-fold cross-validation runs on Mutagenesis$_{RF}$ were tracked for four different root initialisation methods – Unique, standard Bagging, Bagging-75 and Bagging-300. For Bagging-75, each root was initialised by sampling the training set 75 times with replacement, while for Bagging-300 each root was initialised by sampling the training set 300 times with replacement. A root produced using standard Bagging has roughly 63.2% probability of containing a given instance, while that probability is roughly 36% for Bagging-75 and 83% for Bagging-300 (on this particular dataset). This gives a gauge of the effect of varying the amount of available training data for the roots. The mean tree accuracies obtained are shown in Table 6.3. For every value of the MFC, the same ordering is maintained – Unique greater than Bagging, and the three Bagging methods ordered from most data sampled to least. This indicates varying the amount of training data available to the individual trees impacts their accuracy.

Although Bagging shows lower individual accuracies at all tested MFCs, the accuracy of the forests using Bagging is generally only lower than other root initialisation methods at the lowest MFC value. The increased diversity of the ensembles as the MFC increases (shown in Table 6.4) appears to compensate for the lower individual tree accuracies. The diversity here is estimated by taking the class predictions made by each tree across the test set, counting the number of different sets of predictions, and dividing this number by the number of trees. While Bagging-75 and Bagging-std increase in diversity steeply up to MFC 25 and then level off, Bagging-300 seems to be close to its peak by MFC 10. As the number of instances sampled from the training set increases, the diversity of the forests decreases.

Table 6.4: Diversity of trees in RRR-RF forests

MFC	Bagging-75	Bagging-std	Bagging-300
5	0.5322	0.4989	0.4439
10	0.7043	0.5955	0.5040
25	0.8249	0.6513	0.4715
100	0.8769	0.6749	0.4356
200	0.8596	0.6698	0.4503

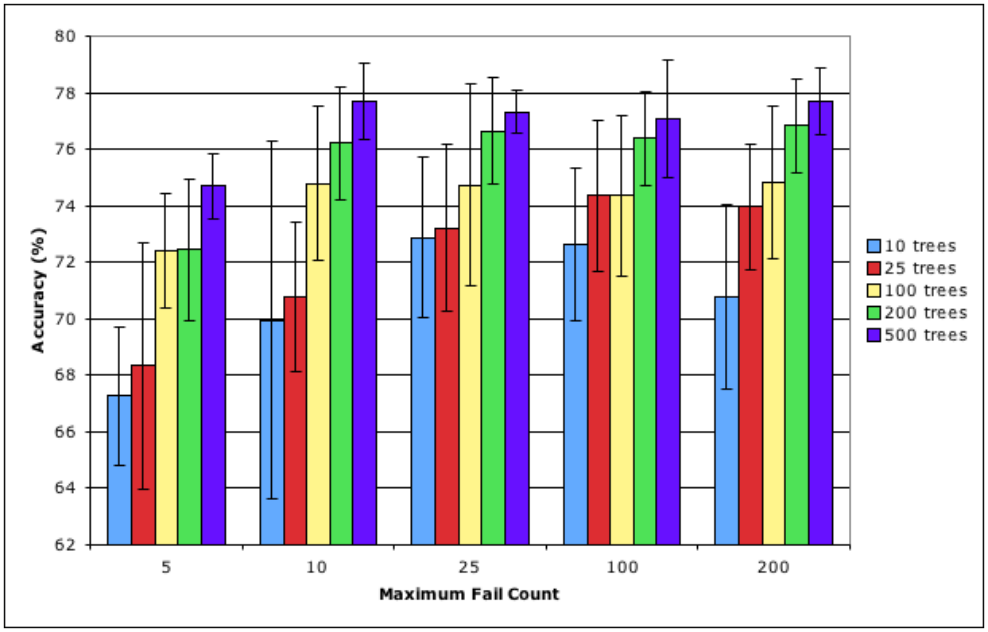


Figure 6.25: Mutagenesis$_{All}$, using out-of-bag evaluation

As expected, increasing the number of trees increases the number of rules generated, as does increasing the Maximum Fail Count, as shown in Figures 6.10-6.11 and Figure 6.16. An increase in the Maximum Fail Count also increases the size of the trees generated, up to a point dependent on the dataset (displayed for Mutagenesis$_{RF}$ in Figure 6.13)– on Musk$_1$, for example, the tree size increases very little beyond MFC 25, as shown in Figure 6.17.

For comparison with the published results for FORF, RRR-RF was also run on Mutagenesis$_{All}$ using out-of-bag evaluation rather than cross-validation. Ten out-of-bag evaluations were performed for each combination of Maximum Fail Count and number of trees. The results are shown in Figures 6.25 and 6.26.

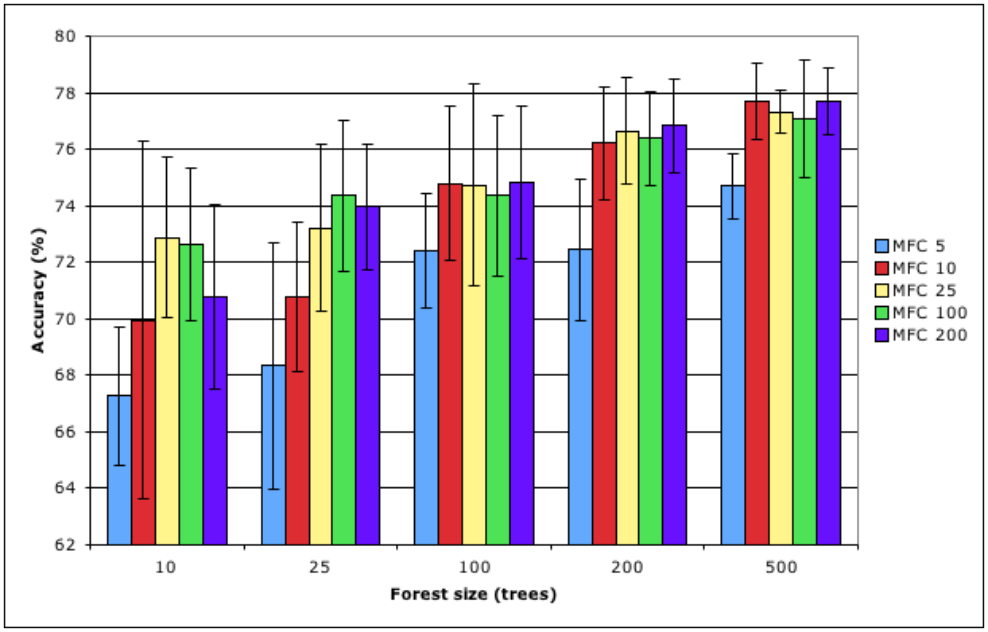


Figure 6.26: Mutagenesis$_{All}$, using out-of-bag evaluation

Table 6.5: Comparison of RRR-RF and FORF

Algorithm	Accuracy	Compared to RRR-RF	Significance
RRR-RF	77.7 ± 1.1	-	-
FORF-NA	74.7 ± 1.4	worse	95%
FORF-SA	78.9 ± 1.8	equal	$< 90\%$
FORF-RA	78.1 ± 1.2	equal	$< 90\%$
FORF-LA	79.0 ± 1.4	equal	$< 90\%$

The results from out-of-bag evaluation exhibit similar properties to cross-validation with respect to the influence of tree number and Maximum Fail Count. Comparing the highest result from RRR-RF (500 trees, MFC 10) and the highest result from FORF-NA (the version that did not use aggregates), RRR-RF was significantly better (with 95% confidence). FORF can also include so-called aggregate functions which go beyond standard relational learning. Compared to the various FORF variants using aggregates (FORF-LA: lookahead aggregates, FORF-SA: simple aggregates, and FORF-RA: refined aggregates) RRR-RF does not perform significantly differently, as shown in Table 6.5 (significance was calculated by corrected t-test, with the bagging fraction (0.632) used to approximate the train-test ratio).

6.4 Variations

Instead of a full rule any prefix of a rule can be used as long as the prefix also results in a non-trivial split. Two methods of prefix selection were tested: choosing a prefix at random (RRR-RF-RAND) and choosing the prefix which maximises information gain (RRR-RF-INFO). The two prefix-selection procedures (along with the original method, abridged slightly, for comparison) are given in Algorithms 21 through 23.

Algorithm 21 Pseudocode for the RRR-RF algorithm, selecting prefix pre-forest (RRR-RF-NORM)

```
Initialise the forest
while Number of open leaves > 0 do
  Generate a Rule
  Select a prefix randomly from those that split the training data
  for all Open Leaves do
    if the rule splits the leaf then
      split the leaf and create children
    else
      increment Fail count
    end if
  end for
end while
```

Algorithm 22 Pseudocode for the RRR-RF algorithm, selecting prefix randomly (RRR-RF-RAND)

```
Initialise the forest
while Number of open leaves > 0 do
  Generate a Rule
  for all Open Leaves do
    for all possible prefixes do
      Check if the prefix splits the leaf
    end for
    if one of the prefixes splits the leaf then
      randomly select one of the splitting prefixes
      split the leaf with the selected prefix and create children
    else
      increment Fail count
    end if
  end for
end while
```

Algorithm 23 Pseudocode for the RRR-RF algorithm, selecting prefix by information gain (RRR-RF-INFO)

```
Initialise the forest
while Number of open leaves > 0 do
  Generate a Rule
  for all Open Leaves do
    for all possible prefixes do
      Calculate information gain for the prefix
    end for
    if the prefix with highest information gain splits the leaf then
      split the leaf with that prefix and create children
    else
      increment Fail count
    end if
  end for
end while
```

The experiments previously performed using RRR-RF-NORM were repeated using RRR-RF-RAND and RRR-RF-INFO, and Table 6.6 shows the overall best accuracy for each dataset, along with the relevant parameters. The RRR-RF-INFO prefix selection procedure and the simple RRR-RF-NORM approach each produce the highest result on three of the datasets, while RRR-RF-RAND produces the best result on only one of the datasets. However, the margin by which these results were the highest was sufficiently small in most cases that those differences may not be overly significant. In general, the effect of the prefix selection method on tree sizes and the number of rules generated is minor, with RRR-RF-RAND and RRR-RF-INFO producing slightly larger trees and generating more rules for low MFC values, and slightly less rules at high MFC values. Both RRR-RF-RAND and RRR-RF-INFO determine the usefulness of prefixes at each leaf, and hence may split more leaves for a given rule than RRR-RF-NORM would, applying the full rule to each leaf. At higher MFCs, RRR-RF-NORM will still split the nodes, though it generates more rules to find successful splits.

The results for out-of-bag evaluation on Mutagenesis$_{All}$ using RRR-RF-RAND and RRR-RF-INFO are slightly higher than those for RRR-RF-NORM, but their significance relative to the FORF results is unchanged.

To compare the runtimes of both algorithms, FORF was also evaluated in a standard cross-validation fashion over all the same datasets as RRR-RF. Table 6.7 summarises these runtime results. They are the time needed in seconds for one complete ten-fold cross-validation run generating 100 trees for various

Table 6.6: Best results for RRR-RF

Dataset	Root	Prefix	Trees	Maximum Fail Count	Accuracy (%)
Carcinogenesis	bagging	norm	500	25	61.24
$\text{Diterpenes}_{52,3}$	unique	info	500	200	97.15
$\text{Diterpenes}_{52,54}$	unique	norm	500	200	94.51
$\text{Diterpenes}_{54,3}$	unique	info	500	100	98.11
Musk_1	standard	info	500	5	84.33
Mutagenesis_{All}	bagging	rand	500	25	78.00
Mutagenesis_{RF}	unique	norm	500	25	84.39

Table 6.7: Training time comparison: time in seconds for one ten-fold cross-validation

Dataset	RRR-RF MFC		FORF query sample probability				
	5	200	sqrt	0.1	0.25	0.5	0.75
Carcinogenesis	656	3940	Did Not Finish				
$\text{Diterpenes}_{52,3}$	312	552	31,505	51,212	78,527	124,578	176,535
$\text{Diterpenes}_{52,54}$	300	600	33,710	44,781	65,081	105,589	141,793
$\text{Diterpenes}_{54,3}$	258	438	27,774	54,038	93,379	163,513	228,131
Musk_1	41	44	Out of memory				
Mutagenesis_{All}	174	894	10,555	14,795	23,158	51,909	84,139
Mutagenesis_{RF}	138	696	4,615	6,419	8,626	12,426	15,760

settings of the respective main parameter governing the building process – the maximum fail count for RRR-RF and the query sample probability for FORF. Entries for Musk_1 are missing as FORF runs out of memory on this dataset and Carcinogenesis is missing because FORF did not finish within reasonable time on this dataset. All the FORF timings have to be viewed cautiously, as they have been produced by non-expert FORF users. They are the result of some exploration of the parameter space and problem representation alternatives, but yet other settings might give faster runtimes. Still, in general RRR-RF manages to generate tree ensembles about two orders of magnitude faster than FORF.

Table 6.8: Diversity for RRR-RF

Root	Prefix	Diversity
bagged	norm	0.9964
bagged	random	0.9962
bagged	info	0.9945
unique	random	0.3345
unique	info	0.3026
unique	norm	0.2953
standard	random	0.2760
standard	info	0.2394
standard	norm	0.2328

6.5 Ensemble Diversity

Each tree in RRR-RF's random forests processes a set of rules that overlaps significantly with those processed by its neighbours, As this could cause the trees to be very similar to their neighbours, and diversity is important to ensemble methods, a set of forests created by RRR-RF using ten ten-fold cross-validation runs on Mutagenesis$_{RF}$ was examined, using each of the nine combinations of root and prefix settings. For each fold, the number of trees that were 'unique' with respect to the test data were determined – i.e. if two (or more) trees produce identical predictions for all test instances, these trees were only counted as one 'unique' tree. Dividing this count by the total number of trees results in a proportion between 0.0 and 1.0, where 1.0 represents perfect diversity between all pairs of trees. The mean results across all folds are shown in Table 6.8, ordered from highest (most diverse) to lowest (least diverse).

Unsurprisingly, root nodes initialised by Bagging result in the most diverse forests, as each tree is built on a different subset of the training data. Requiring the root split to be unique increases diversity over not doing so. Comparing the prefix selection methods – excluding bagged roots, which are practically equal in diversity – shows that RRR-RF-RAND leads to more diverse forests than RRR-RF-INFO, which in turn produces more diverse results than RRR-RF-NORM. This confirms our expectations, as using the same prefix for all leaves should be less diverse than selecting a prefix via some other means, and randomly selecting a prefix should be more diverse than deterministically selecting one per leaf. However, although Bagging clearly produces more diverse ensembles, this is not the only factor affecting the accuracy of the ensembles,

Table 6.9: Static propositionalisation comparison

Dataset	Accuracy (%)	
	Static	RRR-RF
Carcinogenesis	60.91	61.24
$\text{Diterpenes}_{52,3}$	96.97	97.15
$\text{Diterpenes}_{52,54}$	94.22	94.51
$\text{Diterpenes}_{54,3}$	97.69	98.11
Musk_1	89.13	84.33
Mutagenesis_{All}	76.66	78.00
Mutagenesis_{RF}	84.95	84.39

as can be seen in the results in Section 6.4 above.

6.6 Static Propositionalisation

RRR-RF has several advantages over a static two-stage method that generates a propositional representation of the data first, and then constructs a Random Forest based on the propositionalised data. The latter approach must generate a sufficiently large number of rules in the first stage without knowing which ones will actually be useful. The propositional representation is potentially very large, needing a lot of memory, but might still not be a good enough approximation of the relational problem. Thus the number of rules to generate will be a critical parameter for the user to set. RRR-RF has a simple stopping condition: completion of the forest, so that it will always generate exactly the right number of rules. No memory is needed for any intermediate representation, and forest generation is fully parallel. Still, in practice, static propositionalisation works fairly well for the datasets studied here, Table 6.9 summarises results for generating 1000 random rules to be used as Boolean features in propositional Random Forests comprising 100 trees selecting from 30 attributes. Propositional Random Forests are used here for comparison rather than SMO (as used in Chapter 3) as their operation is much more similar to that of RRR-RF. The results from Table 6.2 are reproduced here for ease of comparison.

6.7 Tracking Information Gain

The original Random Forest algorithm chooses the best attribute of a random subset of attributes to split leaves. To emulate this approach, RRR-RF-INFO was modified to track the best rule for each leaf (as determined by information gain) until the maximum number of rules (MRC) have been seen at that leaf and then split the leaf using that rule. This modified algorithm is shown in Algorithm 24.

Algorithm 24 Pseudocode for the RRR-RF algorithm, selecting rule by information gain (RRR-RF-TRACK)

```
Initialise the forest
while Number of open leaves > 0 do
  Generate a Rule
  for all Open Leaves do
    for all possible prefixes do
      Calculate information gain for the prefix
    end for
    if the prefix with highest information gain (p_highest) splits the leaf then
      if p_highest has greater information gain than the stored prefix (p_stored)
      then
        Set p_stored to p_highest
      end if
    end if
    increment Rule Count
    if Rule Count = Maximum Fail Count then
      if p_stored exists then
        Split the leaf using p_stored and create children
      else
        Close the leaf
      end if
    end if
  end for
end while
```

As RRR-RF-TRACK requires each leaf to have seen a set number of rules before a splitting decision is made, it should generate substantially more rules than RRR-RF-INFO (which uses the first splitting rule it sees) for equal values of MFC and MRC. Experimental results for RRR-RF-TRACK using 500 trees and MRC 25 are compared to those for RRR-RF-INFO using 500 trees and MFC 25 in Figures 6.27 through 6.30.

These results show that RRR-RF-track gives improved accuracy over RRR-RF-info, given equal MRC and MFC parameters. RRR-RF-track also generally

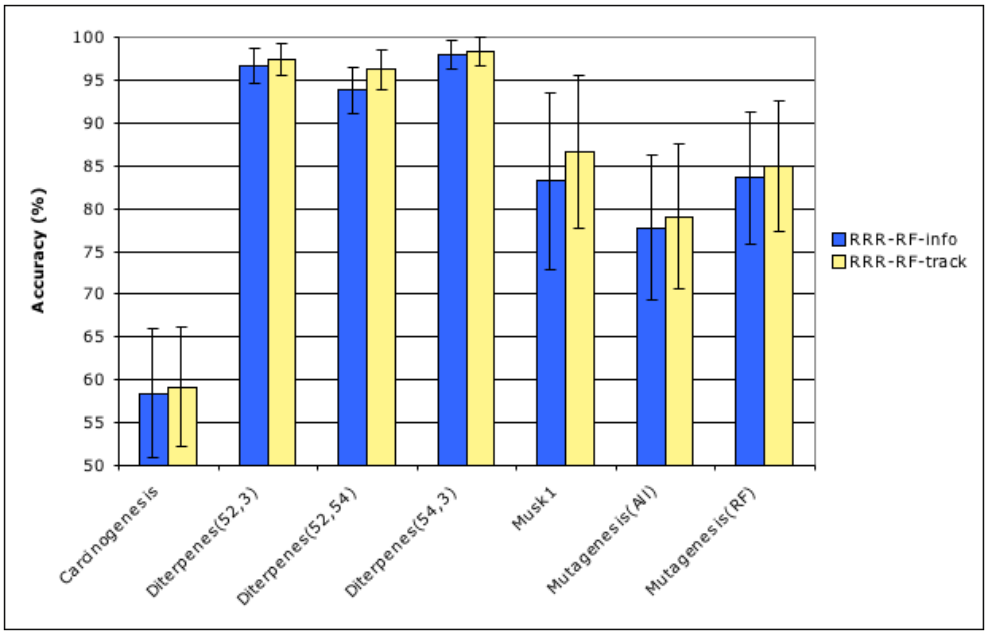


Figure 6.27: Accuracy - RRR-RF-INFO and RRR-RF-TRACK

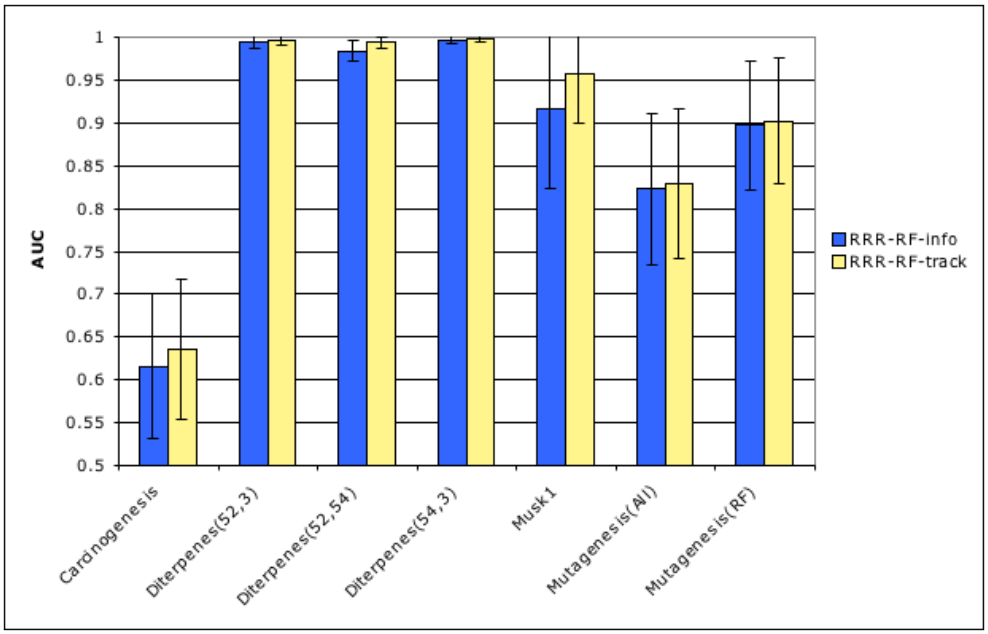


Figure 6.28: AUC - RRR-RF-INFO and RRR-RF-TRACK

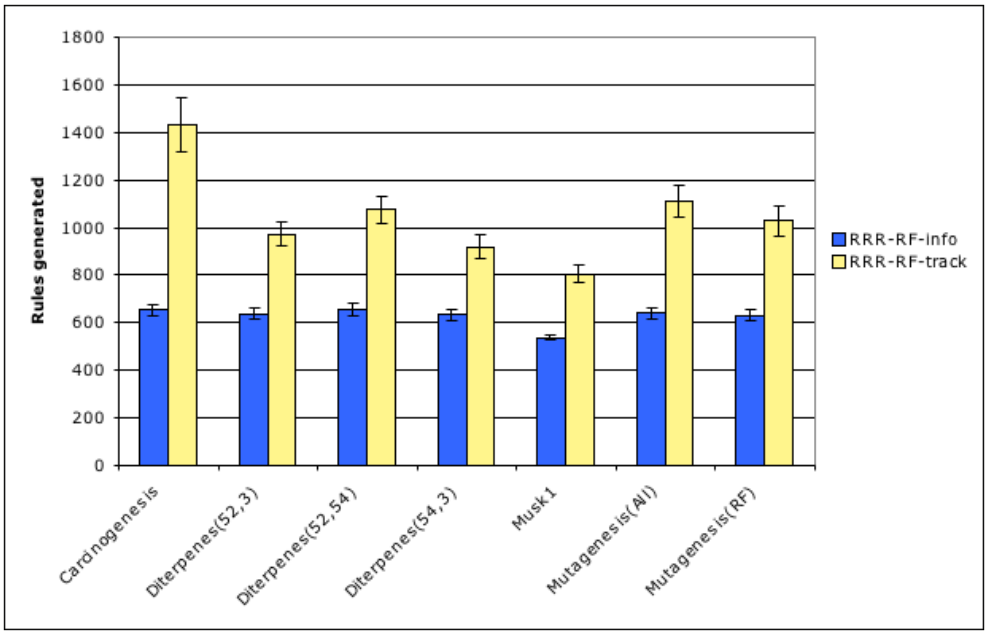


Figure 6.29: Number of rules generated - RRR-RF-INFO and RRR-RF-TRACK

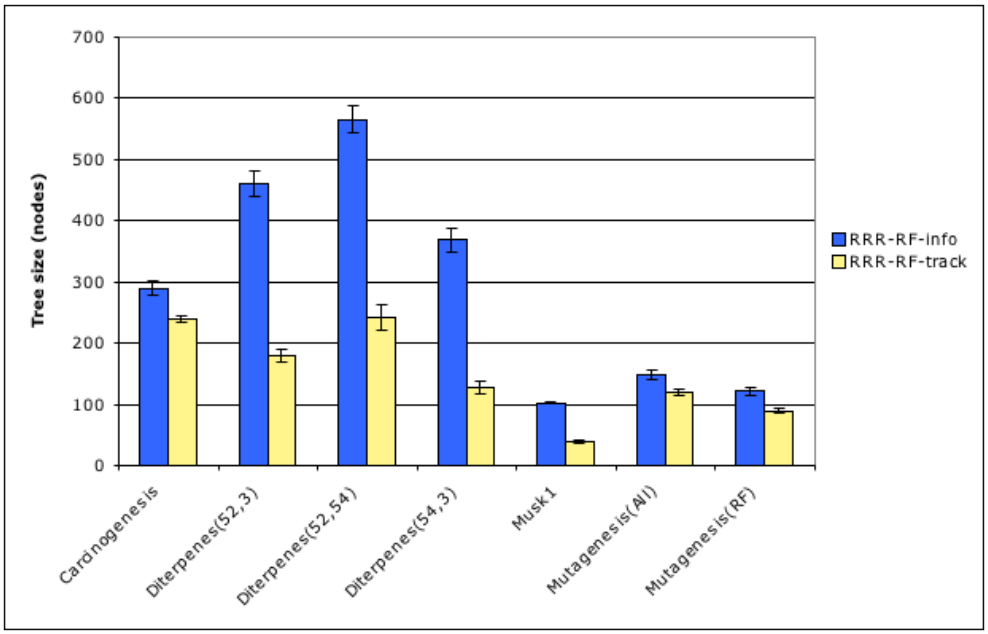


Figure 6.30: Tree size - RRR-RF-INFO and RRR-RF-TRACK

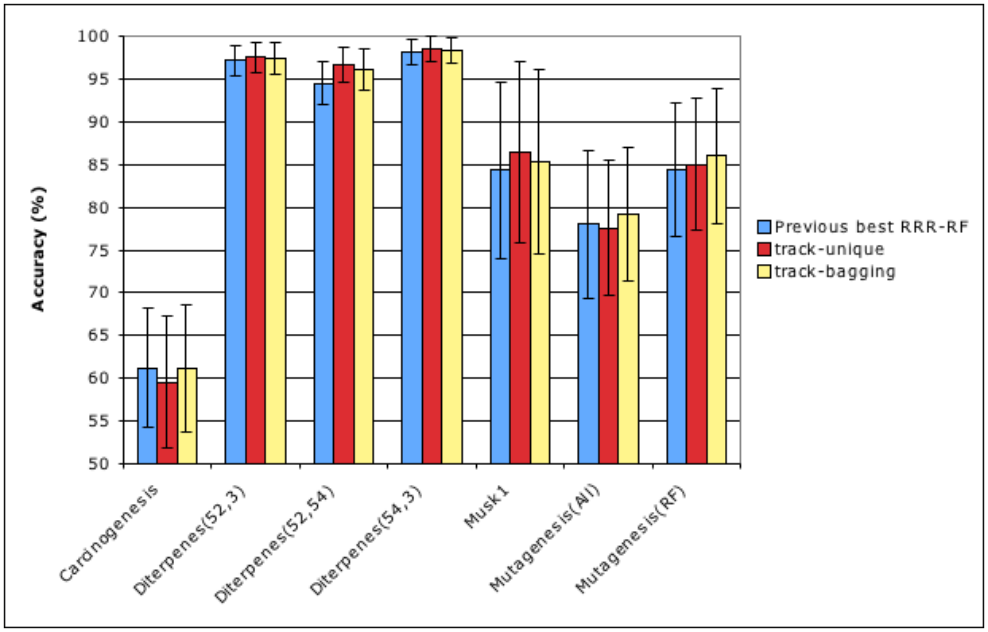


Figure 6.31: Accuracy for RRR-RF-TRACK, compared to previous RRR-RF results

improves AUC over RRR-RF-info. As predicted, the number of rules required to build the trees increases substantially when RRR-RF-track is used. However, the trees generated by RRR-RF-track using those settings are smaller (contain less nodes) than those generated by RRR-RF-info, as pure leaves are found earlier in the tree generation process by the 'better' choices being made with regard to splitting rules.

RRR-RF-track was run ten times using ten-fold cross-validation, using 500 trees, with the MRC set to 50, using Bagging and Unique methods to initialise the roots, and the mean of the resulting accuracies taken. On six of the seven datasets (Carcinogenesis being the exception), one of the two RRR-RF-track systems gives accuracy higher than the previous versions of RRR-RF, as shown in Figure 6.31 and Table 6.10.

6.7.1 FORF Comparison

RRR-RF-track was also run ten times on Mutagenesis$_{All}$ using out-of-bag evaluation (500 trees, 50 MRC). The mean of the resulting accuracies was taken and the results are compared to FORF and the best previous RRR-RF result in Table 6.11. RRR-RF-track shows an improvement in accuracy over RRR-RF, but its results are still not significantly different from those obtained by FORF

Table 6.10: Best results for RRR-RF, including tracking

Dataset	Root	Leaf	Accuracy (%)
Carcinogenesis	bagging	norm	61.24
Diterpenes(52,3)	unique	track	97.49
Diterpenes(52,54)	unique	track	96.59
Diterpenes(54,3)	unique	track	98.46
Musk1	unique	track	86.51
Mutagenesis(All)	bagging	track	79.17
Mutagenesis(RF)	bagging	track	85.99

Table 6.11: Comparison of RRR-RF, RRR-RF-TRACK and FORF (out-of-bag evaluation)

Algorithm	Accuracy	Compared to RRR-RF-TRACK	Significance
RRR-RF	77.7 ± 1.3	worse	95%
RRR-RF-TRACK	79.1 ± 1.1	-	-
FORF-NA	74.7 ± 1.4	worse	95%
FORF-SA	78.9 ± 1.8	equal	$< 90\%$
FORF-RA	78.1 ± 1.2	equal	$< 90\%$
FORF-LA	79.0 ± 1.4	equal	$< 90\%$

using aggregates (again using the approximated corrected t-test, as in Section 6.3).

For further comparison with FORF, RRR-RF-TRACK was also tested on the Financial dataset [4], using ten five-fold cross-validation runs and both Bagged and Unique root nodes. The Financial dataset is composed of 234 bank loans – 203 good and 31 bad. This class distribution is somewhat skewed, such that the accuracy obtained by always predicting the majority class is 86.75%. The non-tracking variants of RRR-RF were also tested, but resulted in accuracy roughly

Table 6.12: Accuracy for RRR-RF-TRACK on the Financial dataset

Number of Trees	Root Method	MRC				
		5	10	25	100	200
100	Unique	86.41	86.80	86.15	86.54	87.70
200	Unique	86.71	85.77	86.71	86.20	86.63
500	Unique	87.01	86.71	87.14	87.31	87.09
100	Bagging	86.67	87.31	87.70	86.93	87.14
200	Bagging	87.09	87.22	87.52	87.31	87.57
500	Bagging	86.54	87.27	87.52	87.52	87.82

Table 6.13: AUC for RRR-RF-TRACK on the Financial dataset

Number of Trees	Root Method	MFC				
		5	10	25	100	200
100	Unique	0.7448	0.7441	0.7568	0.7905	0.8202
200	Unique	0.7587	0.7622	0.7822	0.7913	0.8056
500	Unique	0.7562	0.7801	0.7984	0.8114	0.8241
100	Bagging	0.7417	0.7584	0.7756	0.7894	0.7980
200	Bagging	0.7413	0.7595	0.7932	0.8149	0.8279
500	Bagging	0.7454	0.7723	0.7822	0.7992	0.8115

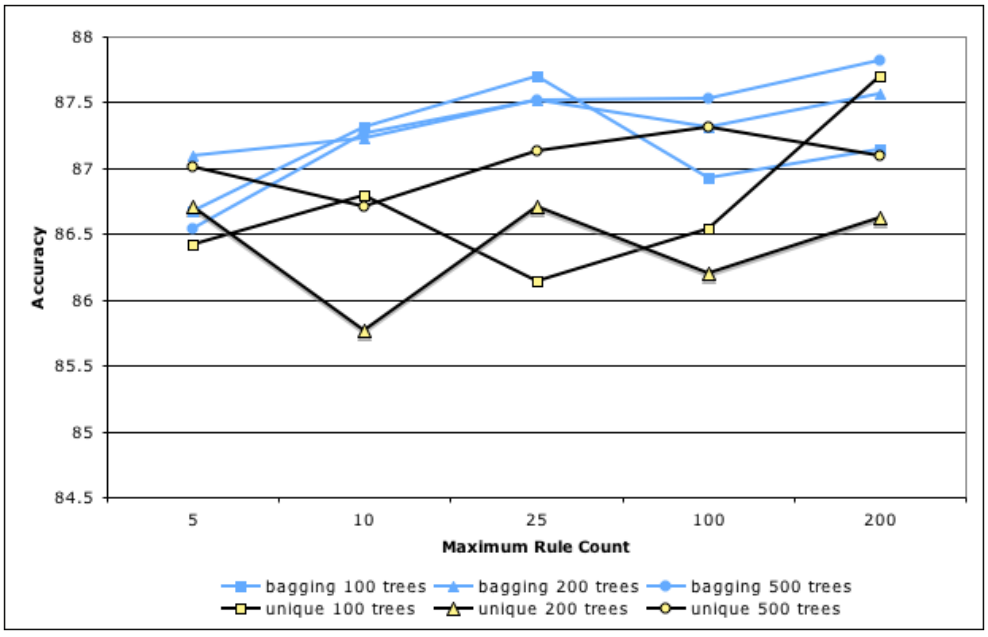


Figure 6.32: Accuracy for RRR-RF-TRACK on the Financial dataset

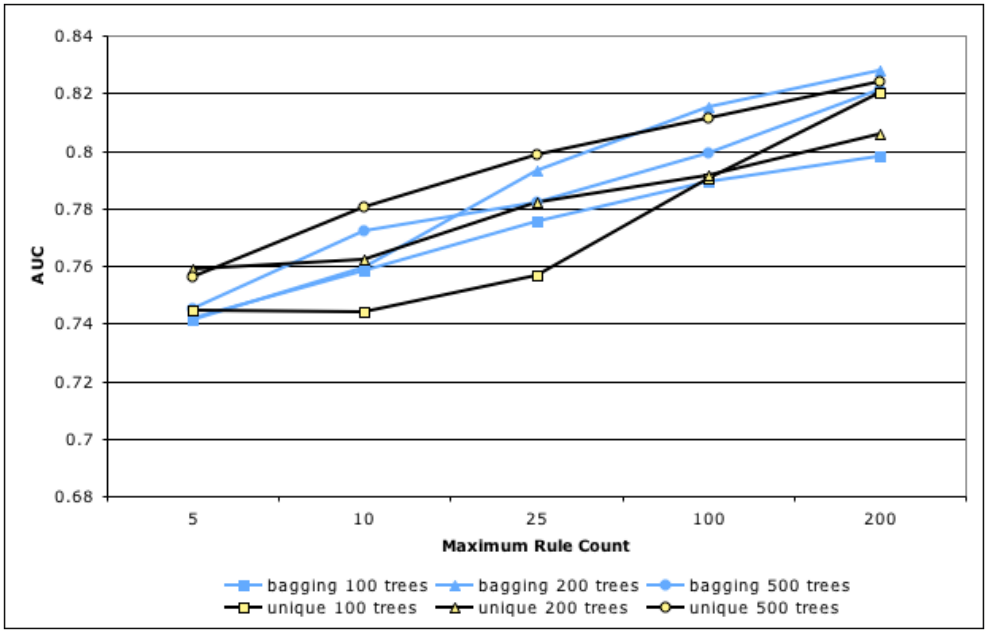


Figure 6.33: AUC for RRR-RF-TRACK on the Financial dataset

equal to that obtained by predicting the majority class and comparatively low AUC values. RRR-RF-TRACK, on the other hand, showed a small improvement in accuracy over predicting the majority class when using Bagged root nodes. The AUCs obtained improve consistently as the MRC is increased. The results are given in Tables 6.12-6.13 and shown in Figures 6.32-6.33, and for high MRCs compare favourably with the results given for FORF in [2], which peak at around 0.875 for accuracy and reach a plateau marginally under 0.8 for AUC.

6.8 Summary

The RRR-RF algorithm was produced by applying randomly generated rules to the random forests framework. Staggered root initialisation allows RRR-RF to produce trees in parallel, and the experimental results obtained are competitive with those achieved by other Relational Random Forest algorithms. The modifications to the root initialisation and prefix selection procedures to increase diversity in the trees also tend to improve the accuracy of the ensemble.

Selecting the best split from a given number of randomly generated rules also produces an increase in accuracy over the standard RRR-RF, and smaller trees, but at the cost of generating a larger number of rules.

Chapter 7

Conclusions

This chapter summarises the thesis and suggests avenues for future work. Section 7.1 summarises the contents of the previous chapters, Section 7.2 discusses the main contributions of the thesis and Section 7.3 describes potential areas for future investigation.

7.1 Summary

In Chapter 2 the RRR algorithm for generating random relational rules was introduced. Ensembles of these rules were used for classification, and the experimental results of RRR-SD were shown to be competitive with the FOIL algorithm.

In Chapter 3 RRR-P was introduced, which made use of the rules generated by RRR for propositionalisation. Applying standard machine learning algorithms to the propositionalised datasets improved on the results achieved by RRR-SD and produced results competitive with those previously published.

In Chapter 4 RRR-C was described – an algorithm that applied propositionalisation (via the RRR-P system) to the domain of relational clustering. Experimental results showed RRR-C to be competitive with the other algorithms tested.

Chapter 5 discussed the results of applying RRR-P to the domain of semi-supervised relational learning. Two semi-supervised methods, making use of unlabeled data, were experimentally compared to corresponding supervised learning methods, and it was found that as the proportion of labeled data was decreased, one of the semi-supervised methods showed results that generally improved on those produced by its supervised counterpart.

In Chapter 6 RRR-RF, an algorithm for generating relational random forests, was introduced. RRR-RF makes efficient use of rules by generating trees and leaves in parallel. Several methods for root initialisation and rule utilisation were experimentally tested. The best results were achieved by variants that utilised semi-random selection of splitting rules.

7.2 Contributions

The following contributions assist the field of relational data mining in the extraction of information from relational data while taking steps to alleviate the inherent complexity in mining that data.

- This thesis presents the RRR algorithm for generating random relational rules. This algorithm is scalable, as it generates a user-controlled number of rules and runs in linear time with regard to the number of rules generated. RRR also contains a number of optimisations to improve the efficiency of its rule evaluation, including the utilisation of rule prefixes and division of rules into subrules.

- This thesis has shown that random rules fulfilling enrichment and uniformity constraints provide an effective method for propositionalising relational data for classification. Once data has been propositionalised, sophisticated propositional learning methods can be applied to the data. Empirical results provide evidence that this process is competitive with other relational learning algorithms. A single propositionalisation can have multiple flat-file learning algorithms applied to it, reducing the amount of relational processing required.

- This thesis has shown that the propositionalised data can be used with learning techniques beyond simple classification.

  - This thesis has investigated the application of random relational rules to clustering. Empirical results show that clustering using RRR-C is competitive with the two other approaches that are compared.

  - This thesis has investigated the application of random relational rules to semi-supervised learning. Empirical results demonstrate that one of the semi-supervised learning methods investigated shows

improvements over the corresponding supervised method when the proportion of unlabeled data is increased. It should be noted that a high proportion of unlabeled data is the most common setting for semi-supervised learning.

- This thesis has shown that random relational rules can be used to generate random forests. The RRR-RF algorithm tests each generated rule at every open leaf, allowing trees to be generated in parallel. Forest generation is shown to be time-efficient compared to another relational random forest algorithm (FORF), and experimental results demonstrate that the results achieved are competitive with both FORF and previous applications of RRR described in the thesis.

7.3 Future Work

Chapter 2 described the production of random relational rules and the use of these rules in classification. While subsequent chapters discussed further uses of the rules, refinements to the rule generation process itself could be made. Alternative constraints to the enrichment and uniformity requirements could be explored. Rules could be weighted based on their coverage on the training data, allowing the generated rulesets to be pre-processed before being used for classification – for example, using only a specified number of the 'best' rules, according to their weighting.

When test literals are generated, the variable to test is selected with equal probability from the possible variables. This probability could be weighted to take into account the number of possible values each variable could take on, so that the random selection would be, in effect, selecting with equal probability from the possible variable-comparator pairs.

Unlike other relational learning algorithms that strive to induce a best possible set of rules, not all of the random rules generated by RRR need to be fully evaluated. If the evaluation time for a particular rule were to exceed a pre-specified time limit, the evaluation could be aborted and that rule discarded. Preliminary experiments measuring single rule evaluation times showed that most rules are evaluated very quickly, but occasional single rules would take particularly large amounts of time. Future work could explore this efficiency versus potential loss of information trade-off in more detail.

Further efficiency gains in rule evaluation might be made by applying query

optimisation to the generated rules before evaluation. Such techniques have been previously applied from both the ILP [72] and database [67] perspectives. To optimise the rules for evaluation, the literals that make up a rule would be reordered in an attempt to minimise the total number of literal evaluations that would need to be performed. The optimisation process utilises estimates of the branching caused by each literal, although smallest-first is not always the optimal solution.

However, this could affect the use of rule prefixes – for example, this might cause rules to be ordered such that particular (efficient) predicates always appear first. It is possible that each prefix of the rule could be separately optimised and evaluated, although this would obviously be more costly in terms of evaluation time. Whether the gain in performance due to optimisation would outweigh the drop due to evaluating multiple prefixes is unclear, but could be investigated.

In Chapter 3 two propositional algorithms were applied to the propositionalised datasets. However, any propositional classification algorithm could be applied to the propositional data.

If algorithms that report attribute weightings were used, the resulting weights could be analysed to determine the best rules that were produced, and those rules could then be translated into a more human-comprehensible form – comprehensibility of ensemble models has been previously been reported to be desirable [68, 2]. The automated translation of first-order logic into natural language is discussed in [52].

The relational clustering algorithm introduced in Chapter 4 could be compared to more standard clustering approaches. Kernels for relational data [4] could be used together with clustering algorithms like KernelKMeans [2]. The rule generation process could be replaced by either a relational association rule finder like Warmr [69], or class-blind variants of relational rule learners such as Foil [62] or Progol [54]. The suitability of this approach for different types of data could also be investigated – the datasets used in these experiments were composed of distinct examples with no linkage between single examples. There are applications where links between examples can carry essential information [35], and the effectiveness of Rrr-c could be evaluated for such data.

Chapter 5 discussed a propositionalisation approach to semi-supervised learning. After propositionalisation, any standard semi-supervised learning algorithm could be applied to the resulting propositional problem – the method

described in this paper is orthogonal to such methods as LLGC [85, 59].

If standard semi-supervised learning algorithms, which usually rely on some notion of distance or similarity, were to be applied directly to the relational representation instead of the propositionalisation approach put forward in this paper, then relational notions of distance and similarity [83, 32] would need to be exploited.

The RRR-RF algorithm for random forest generation, described in Chapter 6, applies each rule it generates to all open leaves in the forest, which, while efficient, limits the diversity of the forest, and may have an effect on accuracy. The consequences of this could be investigated by comparing two other algorithms for forest generation. The first would generate trees sequentially, restricting generated rules to only be applied to open leaves in a particular tree, while the second would generate leaves sequentially, restricting generated rules to only be applied to a single leaf.

Additionally, aggregation – a propositionalisation technique derived from the field of databases – has been shown to work well in combination with a relational random forest algorithm [3]. It is possible that the RRR-RF algorithm could benefit from the introduction of aggregates into rule generation – in fact, the result of integrating aggregates into the RRR generation algorithm itself could be investigated. This would enable RRR to utilise metadata (summary statistics such as minimum, maximum, mean and quantiles) derived from each instance in rules, and also to produce non-Boolean propositional attributes, as RELAGGS [43] does.

More broadly, this thesis has demonstrated the viability, both in terms of efficiency and accuracy, of randomised search in relational space, and future work could investigate alternative methods for randomised relational search.

Appendix A

Other Results

A.1 Parameter Effects on Propositionalisation

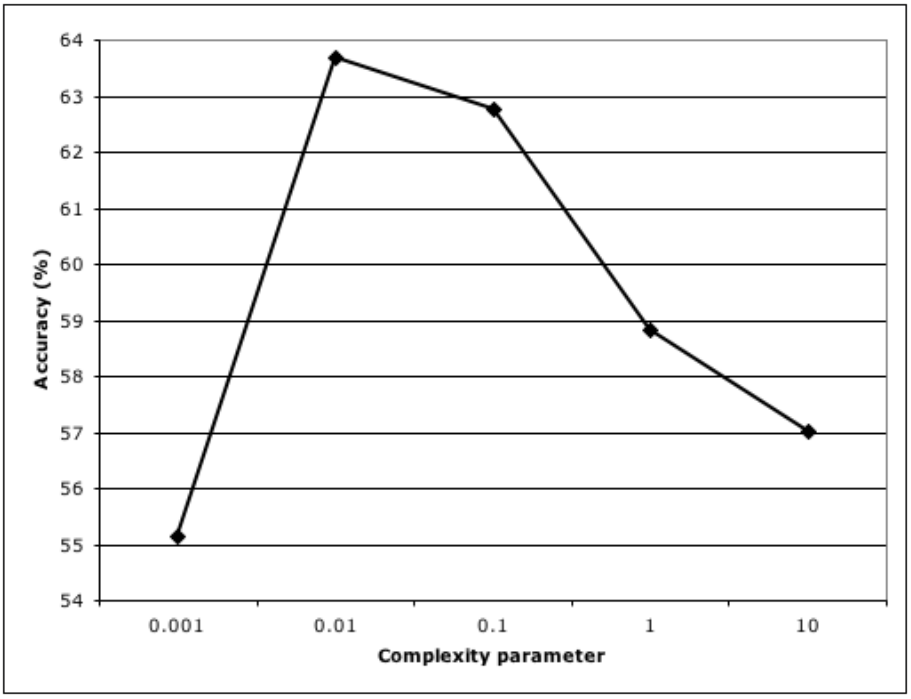


Figure A.1: Accuracy for RRR-P on Carcinogenesis, using SMO

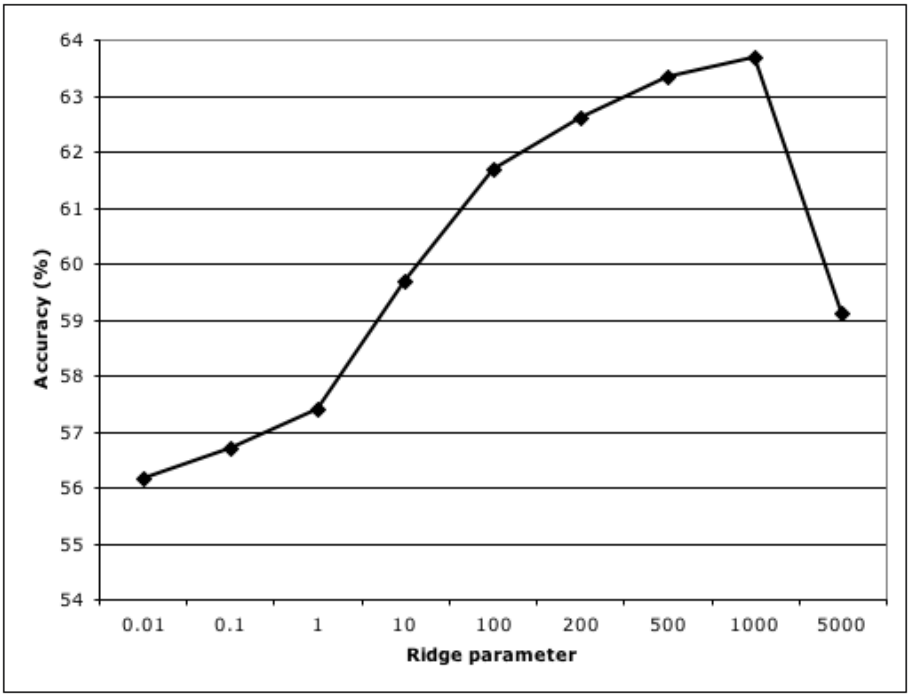


Figure A.2: Accuracy for RRR-P on Carcinogenesis, using Logistic

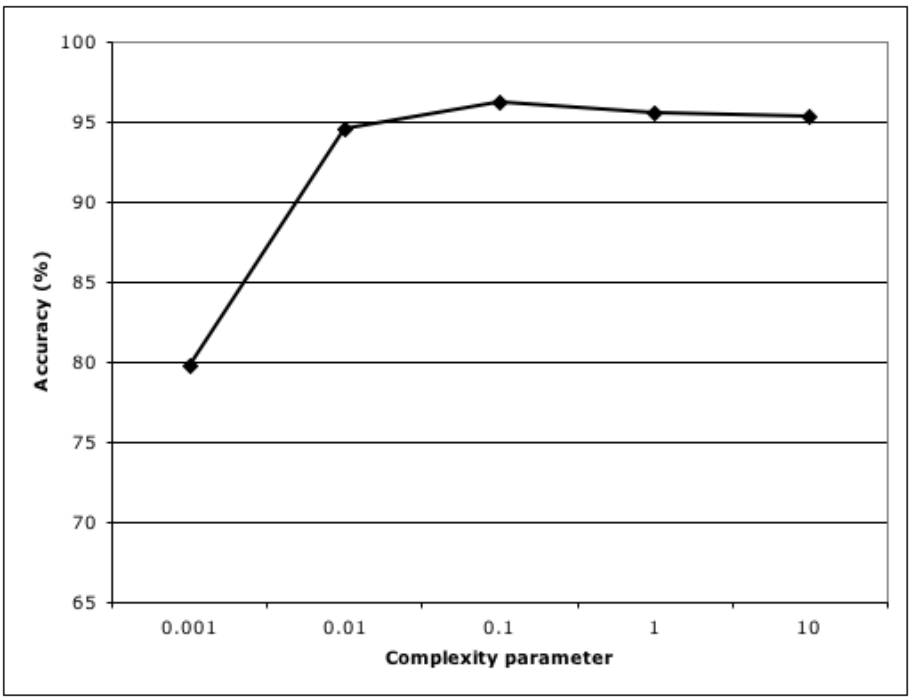


Figure A.3: Accuracy for RRR-P on $\text{Diterpenes}_{52,3}$, using SMO

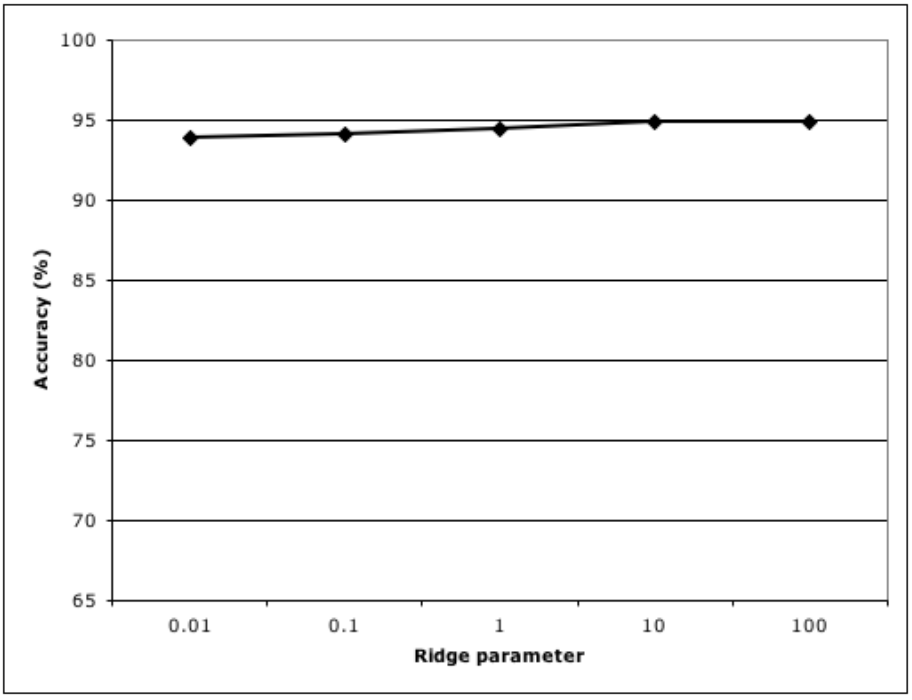


Figure A.4: Accuracy for RRR-P on $\text{Diterpenes}_{52,3}$, using Logistic

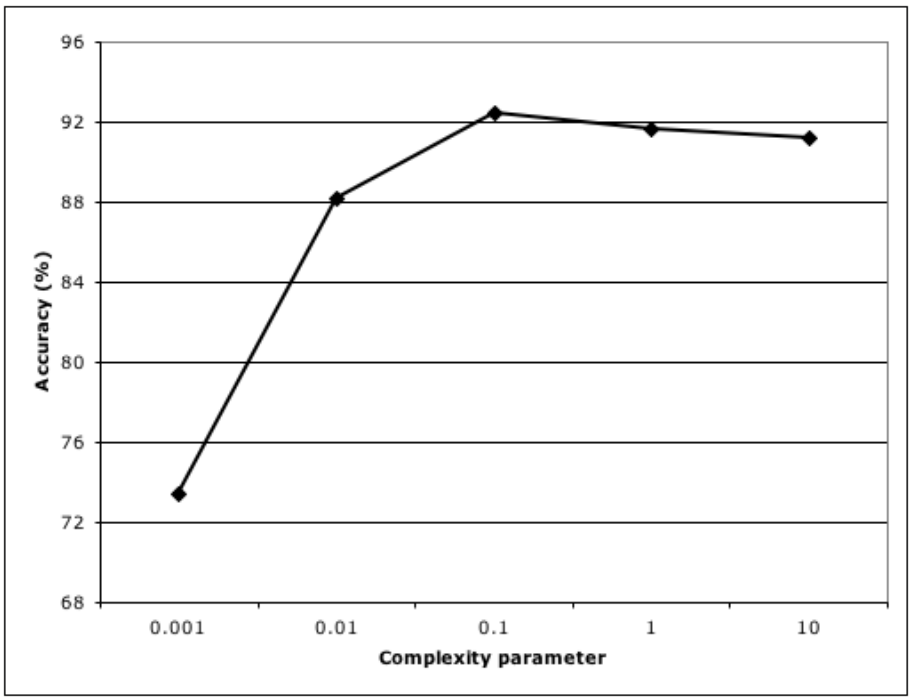


Figure A.5: Accuracy for RRR-P on Diterpenes$_{52,54}$, using SMO

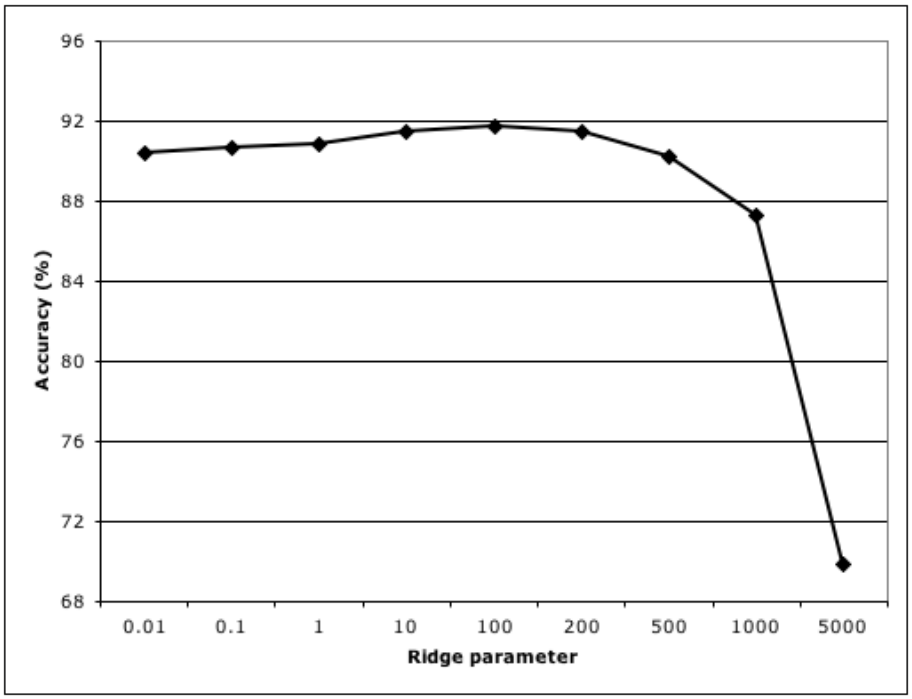


Figure A.6: Accuracy for RRR-P on Diterpenes$_{52,54}$, using Logistic

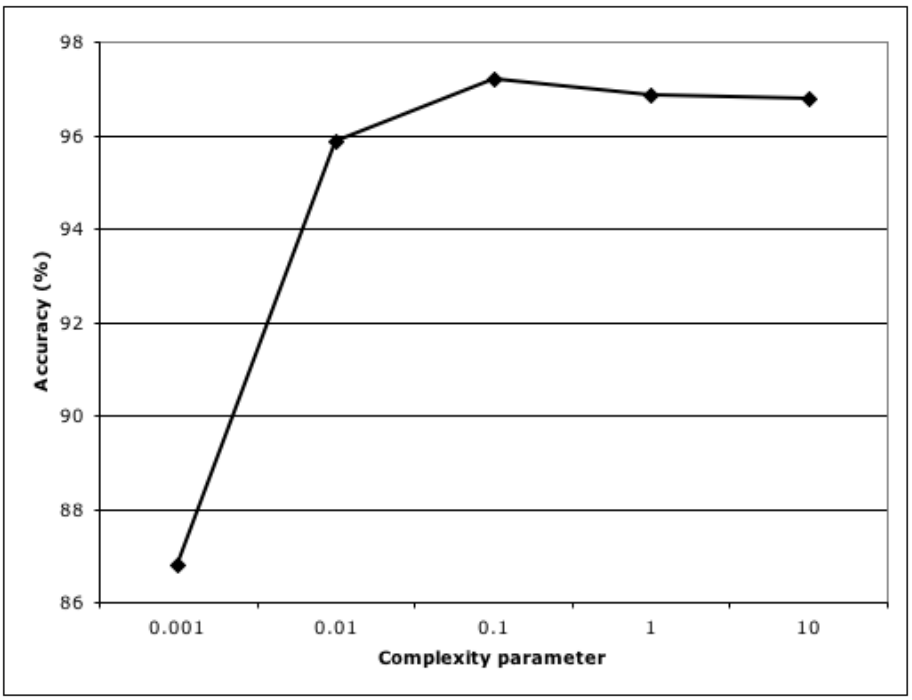


Figure A.7: Accuracy for RRR-P on $\text{Diterpenes}_{54,3}$, using SMO

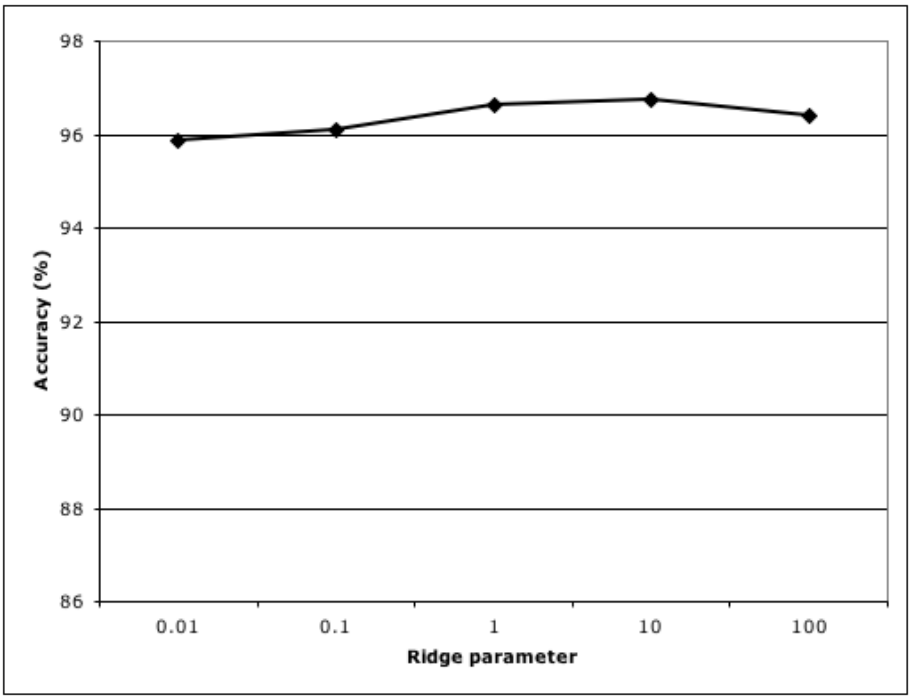


Figure A.8: Accuracy for RRR-P on $\text{Diterpenes}_{54,3}$, using Logistic

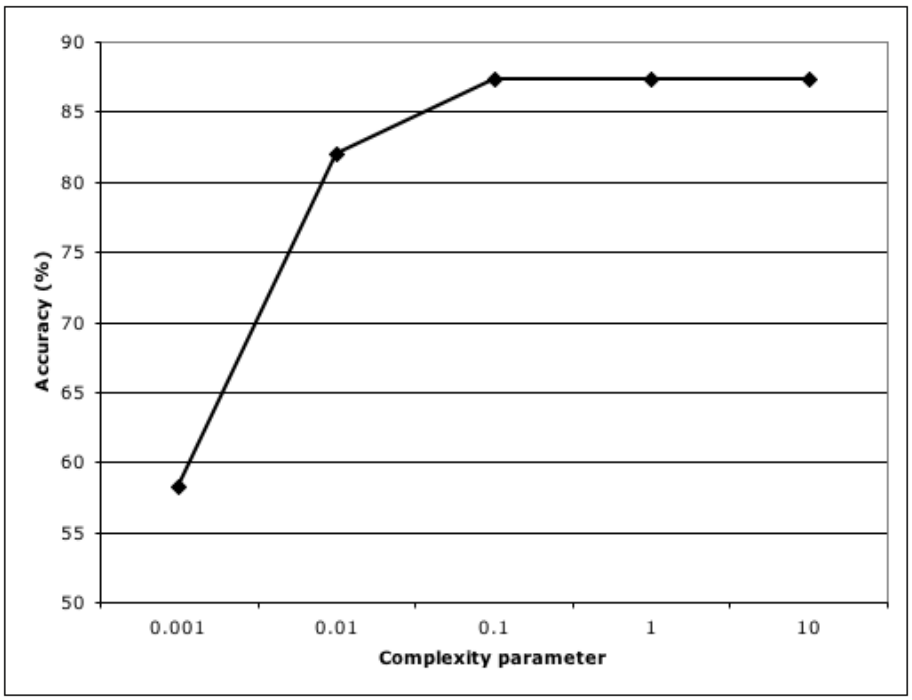


Figure A.9: Accuracy for RRR-P on Musk_1, using SMO

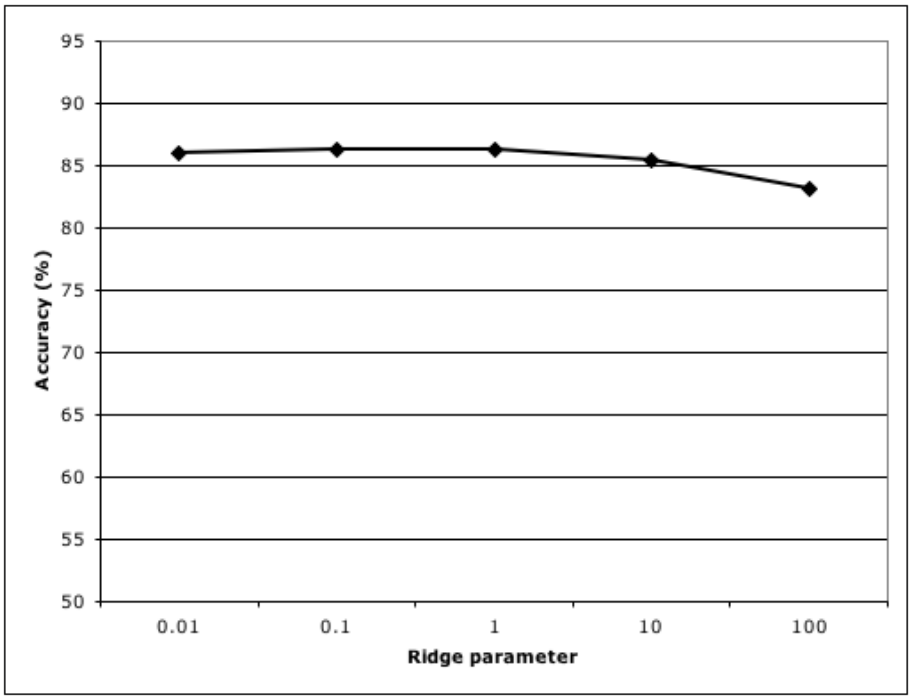


Figure A.10: Accuracy for RRR-P on Musk_1, using Logistic

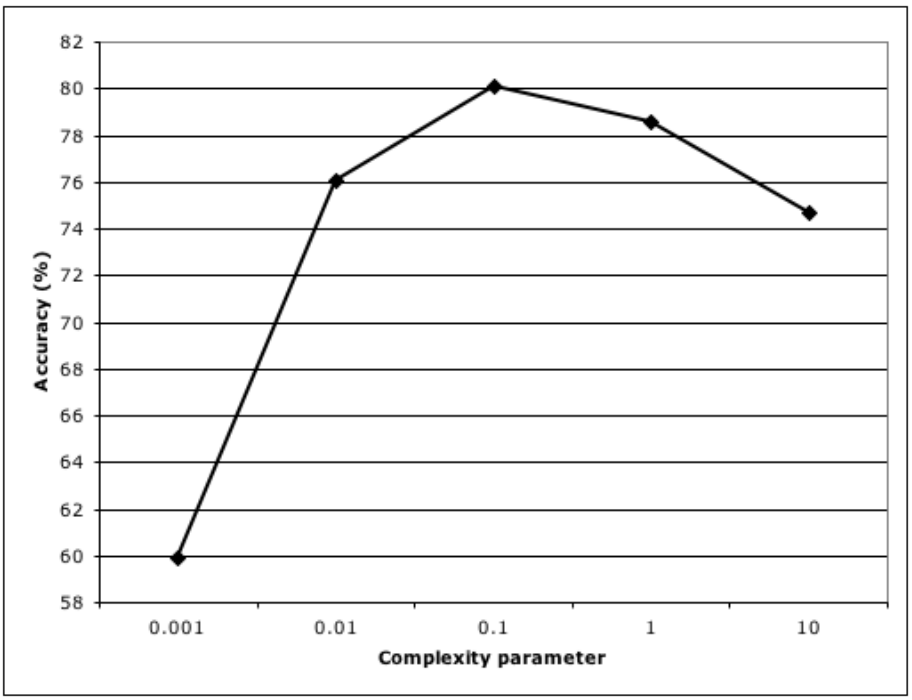


Figure A.11: Accuracy for RRR-P on Mutagenesis$_{All}$, using SMO

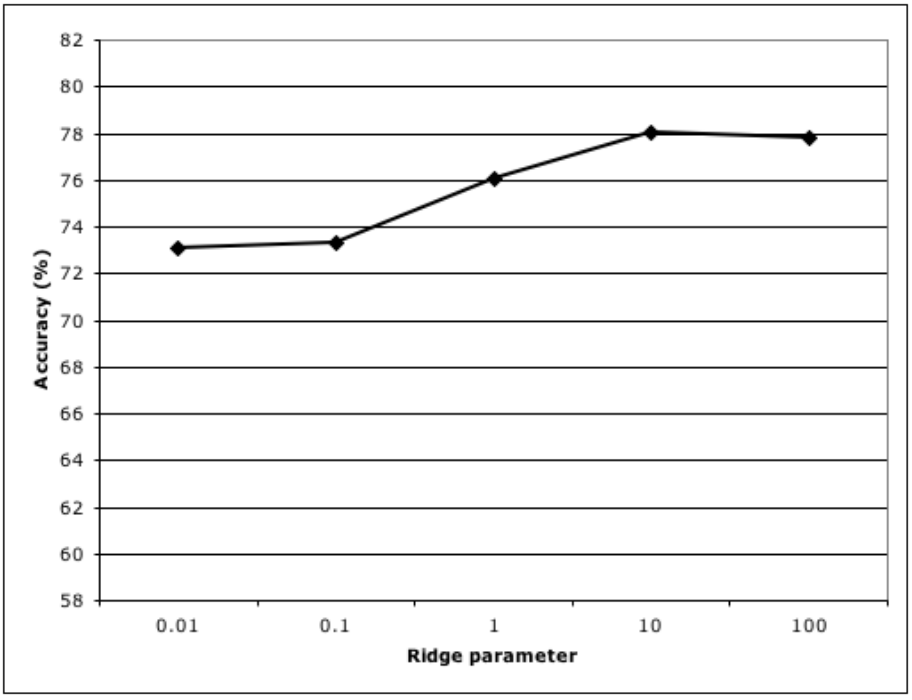


Figure A.12: Accuracy for RRR-P on Mutagenesis$_{All}$, using Logistic

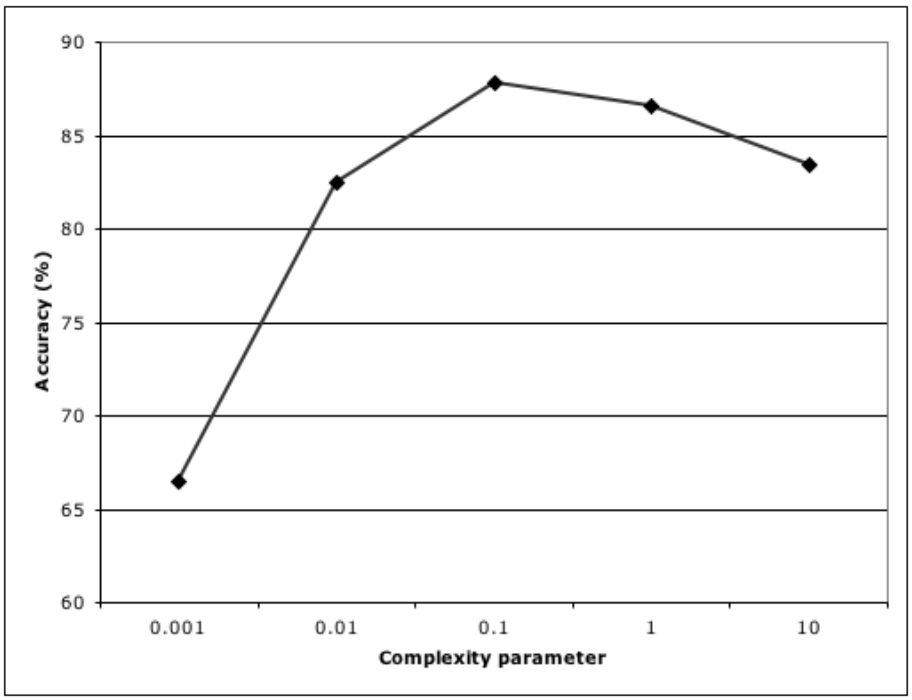


Figure A.13: Accuracy for RRR-P on Mutagenesis$_{RF}$, using SMO

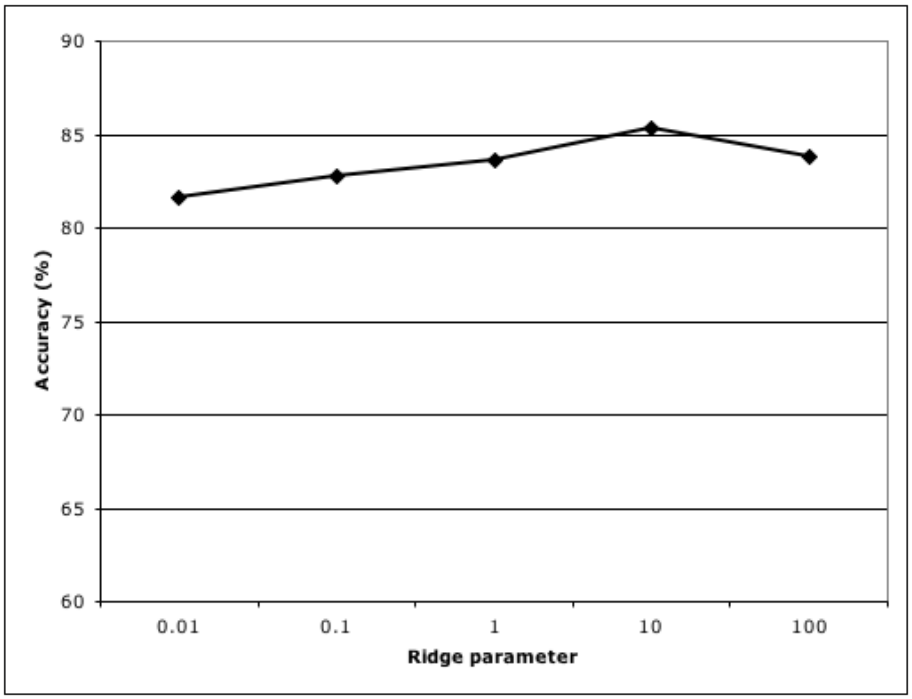


Figure A.14: Accuracy for RRR-P on Mutagenesis$_{RF}$, using Logistic

A.2 Clustering Figures

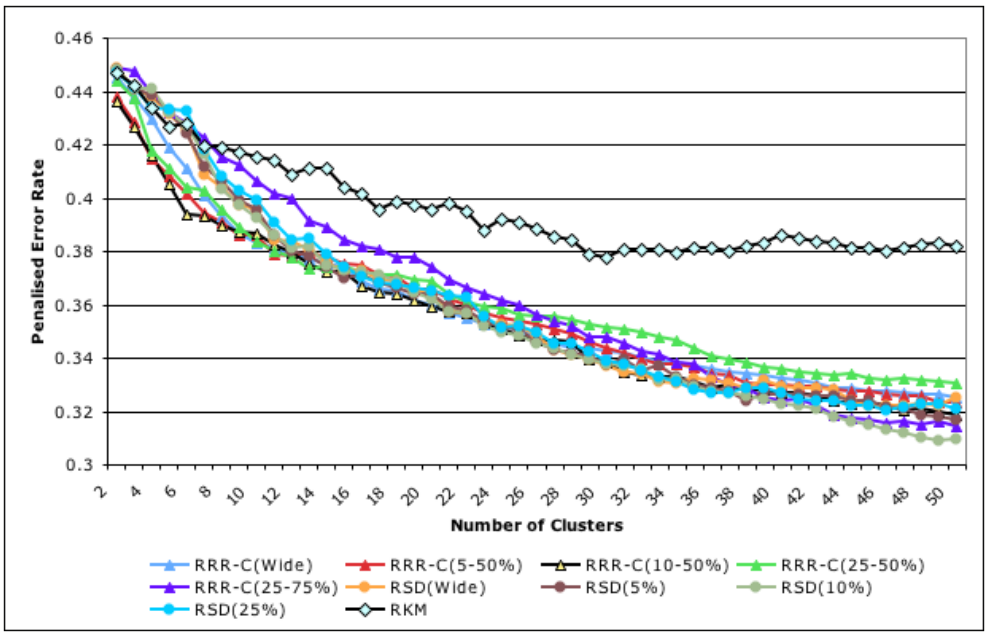


Figure A.15: Penalised error rates on Carcinogenesis

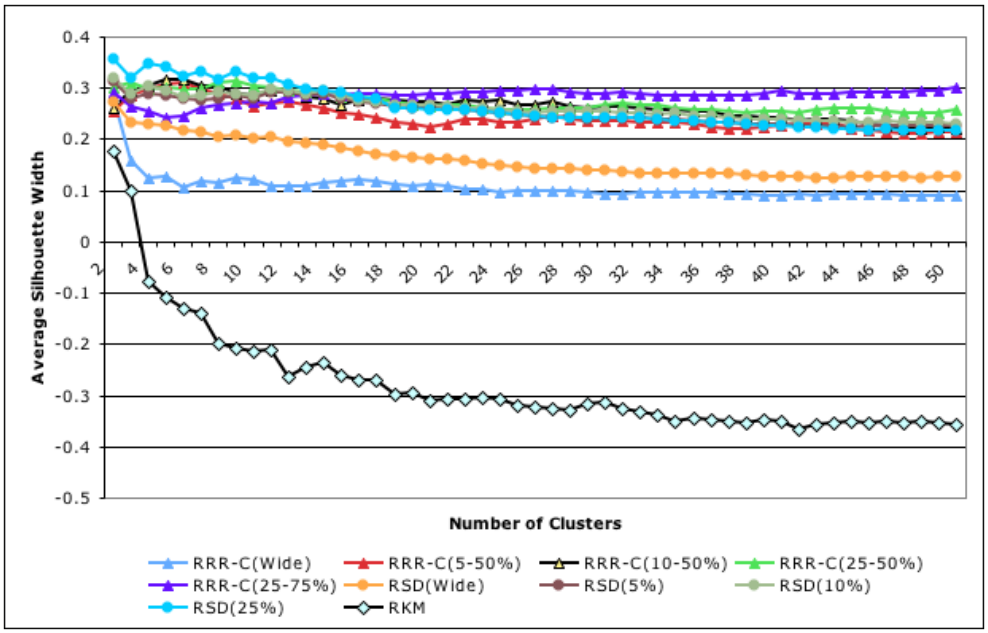


Figure A.16: Average silhouette widths for Carcinogenesis

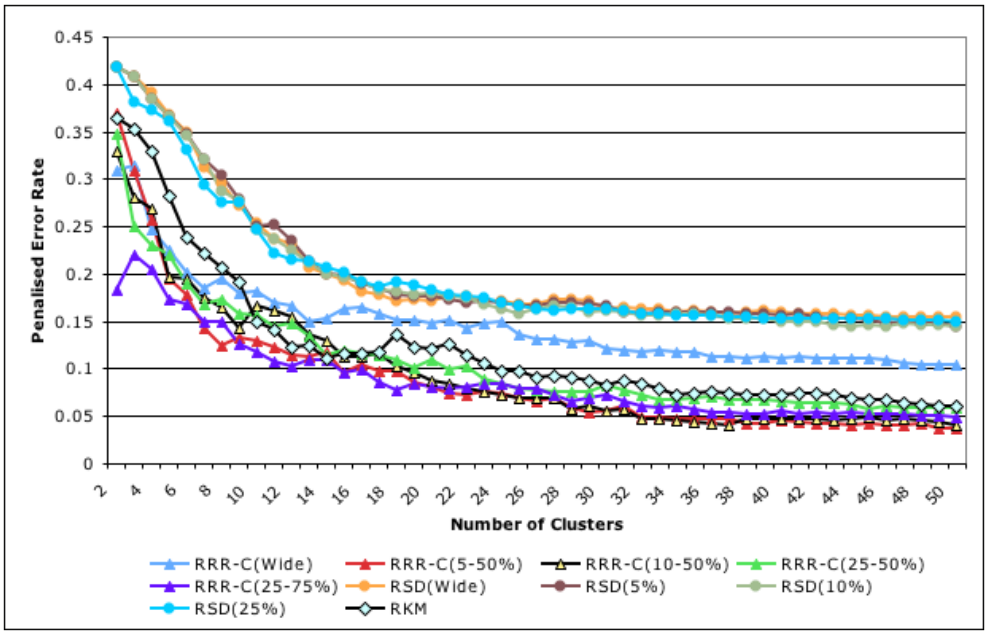


Figure A.17: Penalised error rates on Diterpenes$_{52,3}$

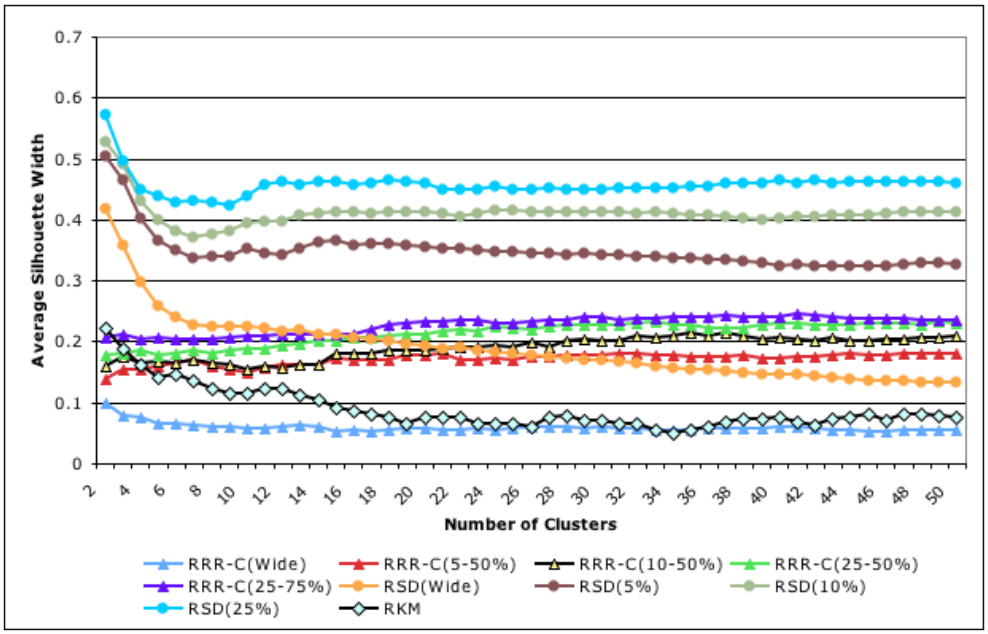


Figure A.18: Average silhouette widths for Diterpenes$_{52,3}$

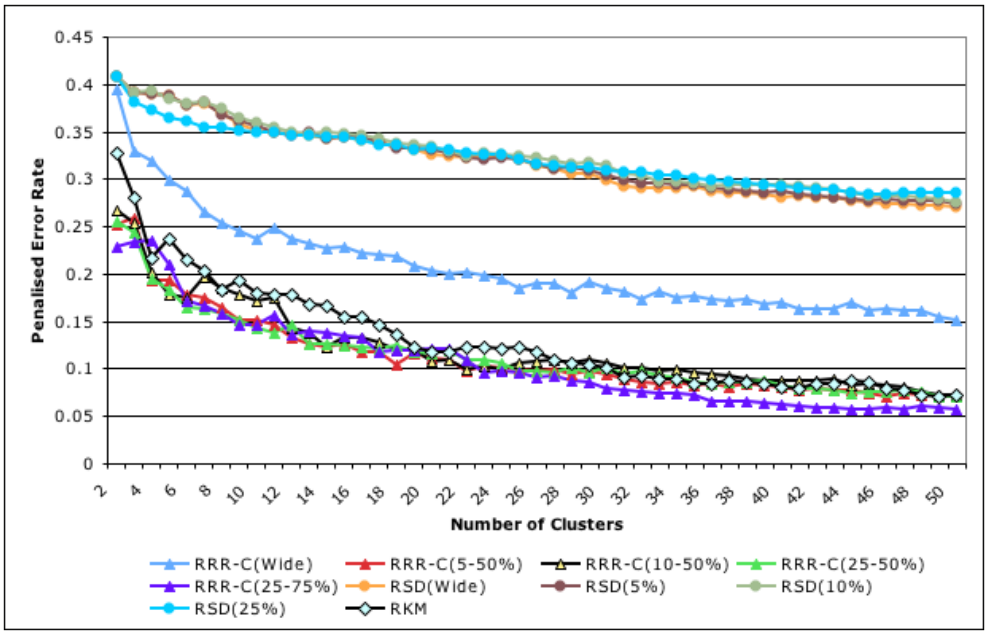


Figure A.19: Penalised error rates on $\text{Diterpenes}_{52,54}$

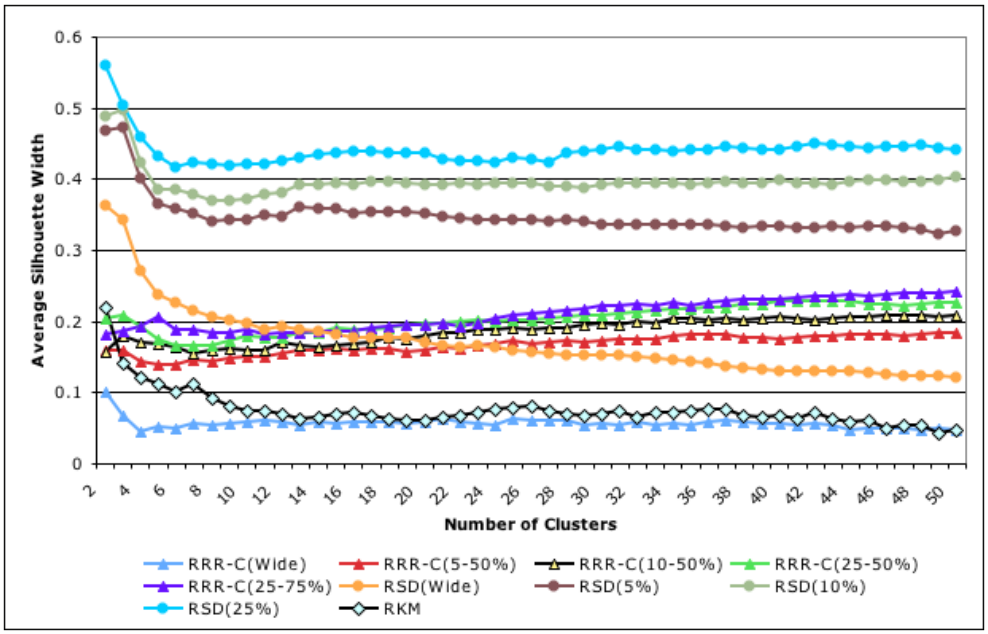


Figure A.20: Average silhouette widths for $\text{Diterpenes}_{52,54}$

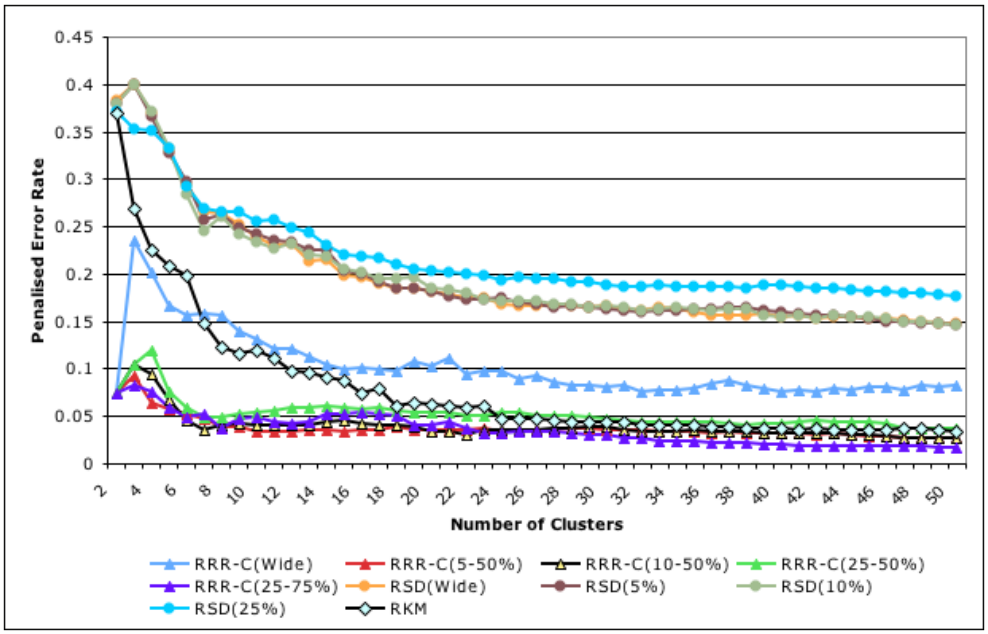


Figure A.21: Penalised error rates on $\text{Diterpenes}_{54,3}$

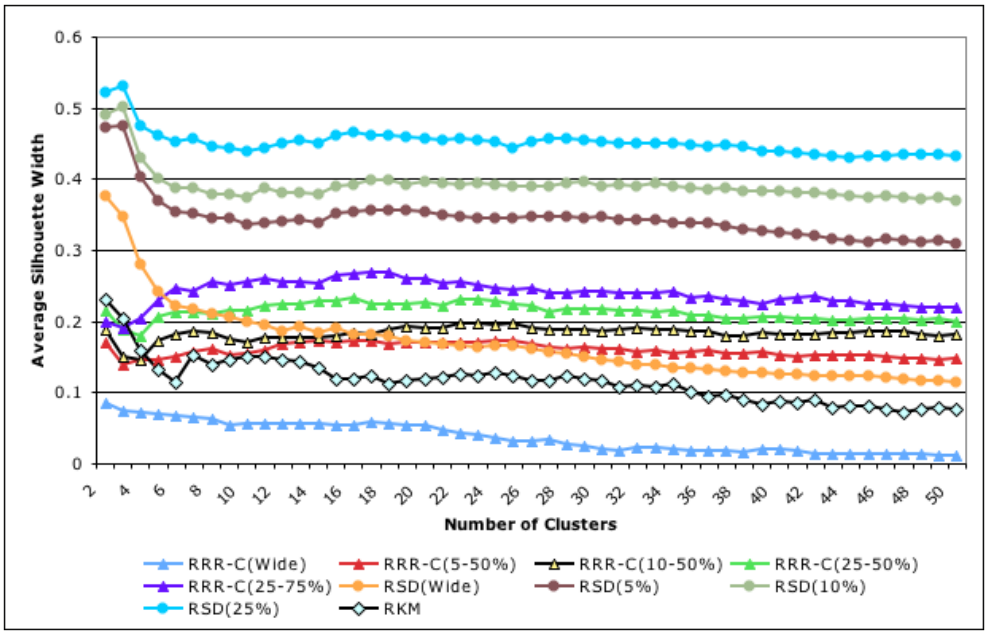


Figure A.22: Average silhouette widths for $\text{Diterpenes}_{54,3}$

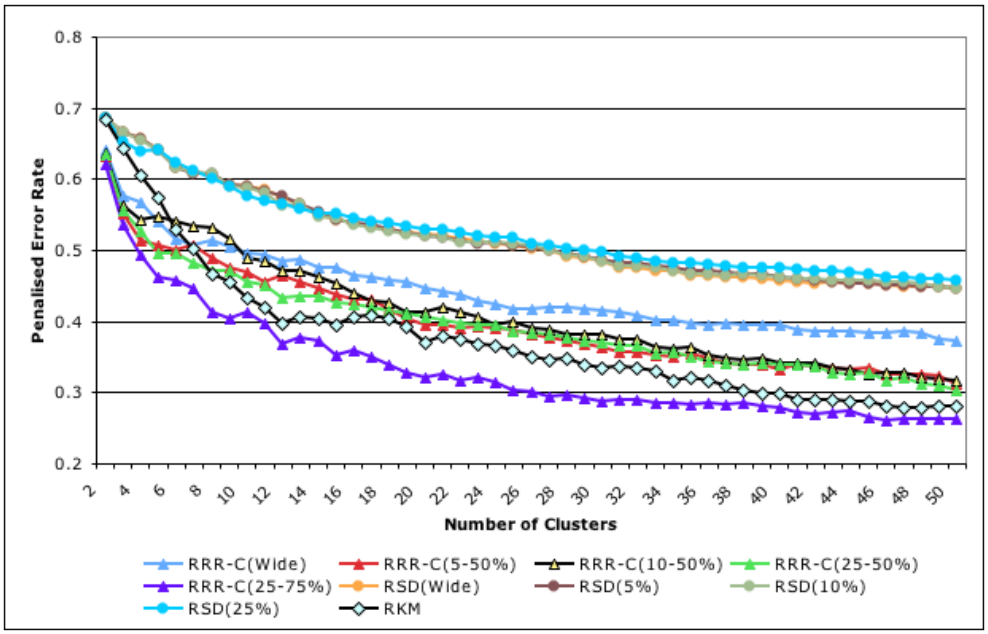


Figure A.23: Penalised error rates on Diterpenes$_{All}$

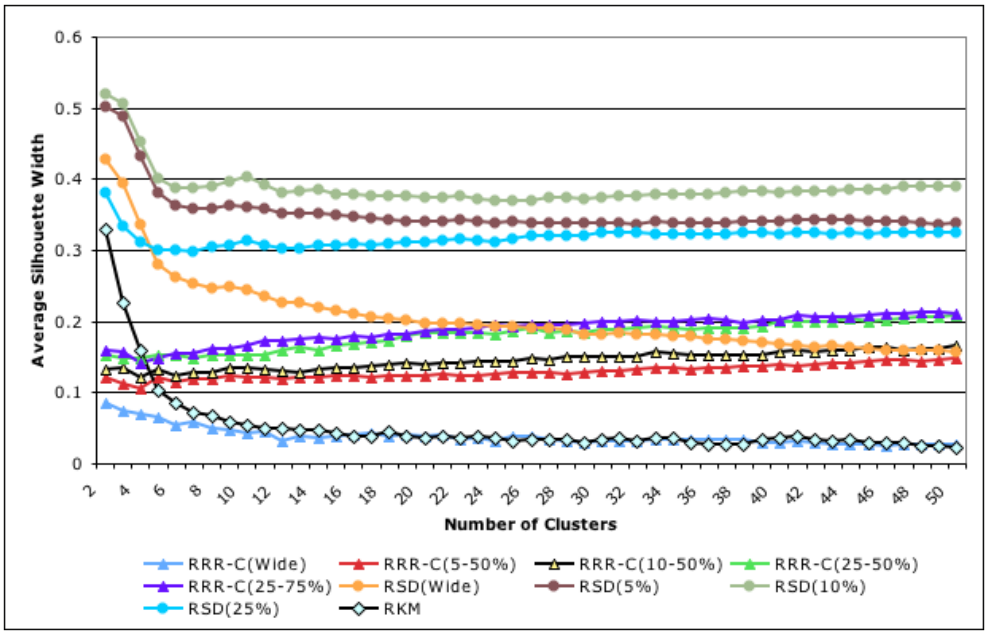


Figure A.24: Average silhouette widths for Diterpenes$_{All}$

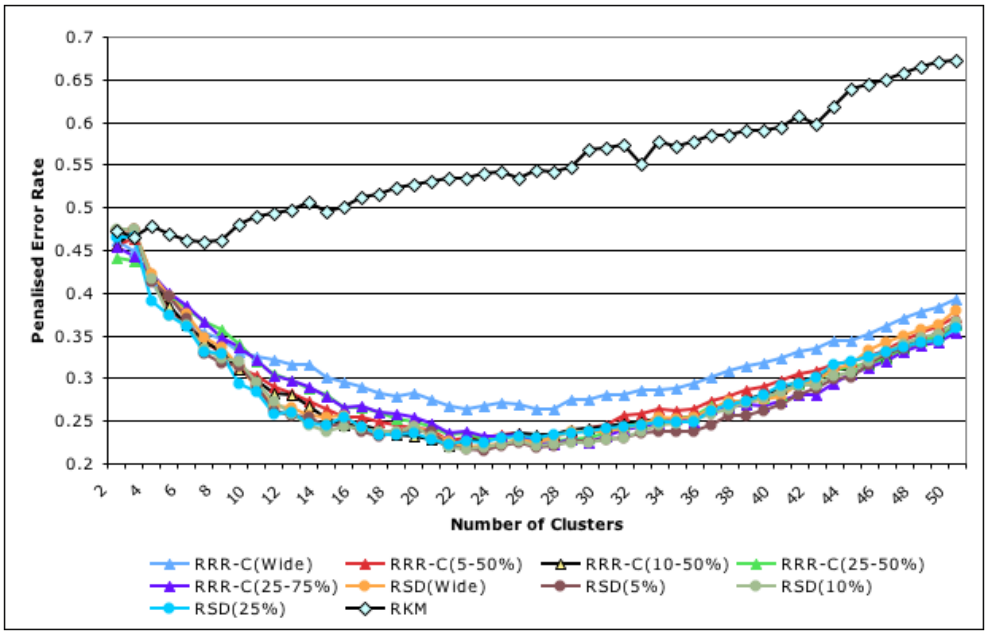


Figure A.25: Penalised error rates on Musk_1

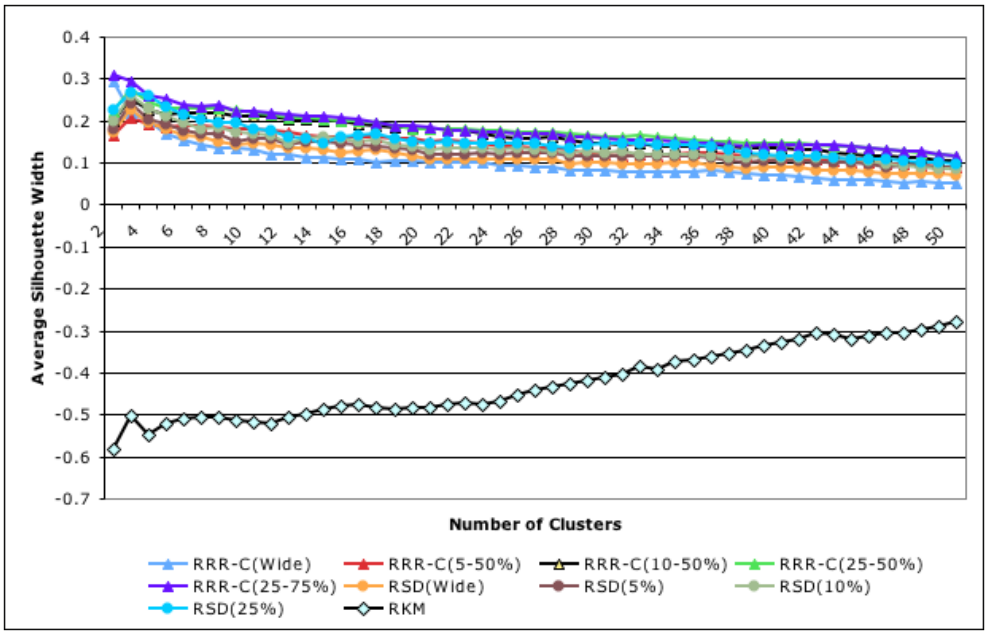


Figure A.26: Average silhouette widths for Musk_1

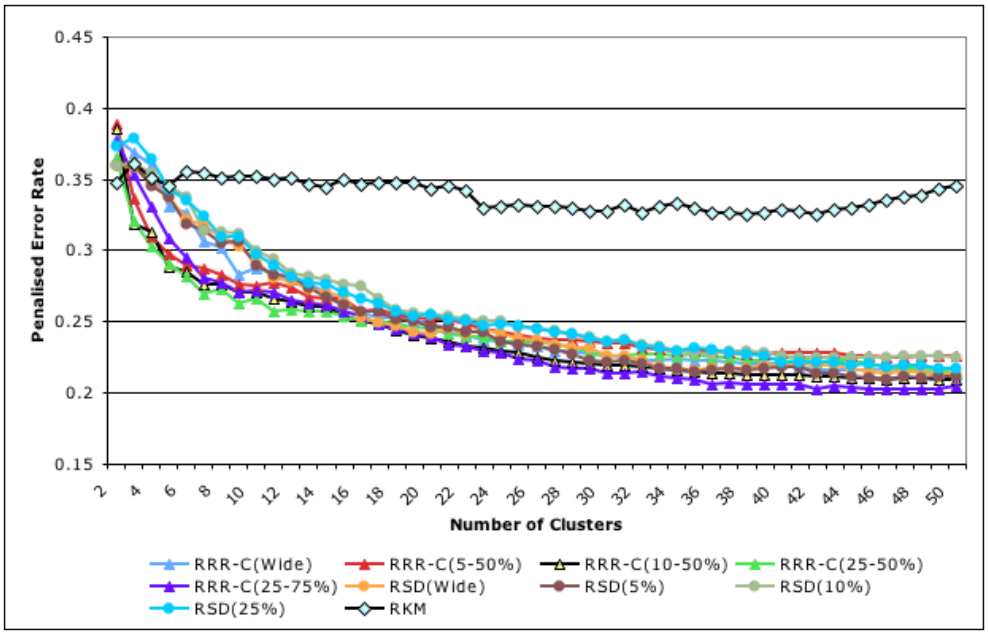


Figure A.27: Penalised error rates on Mutagenesis$_{All}$

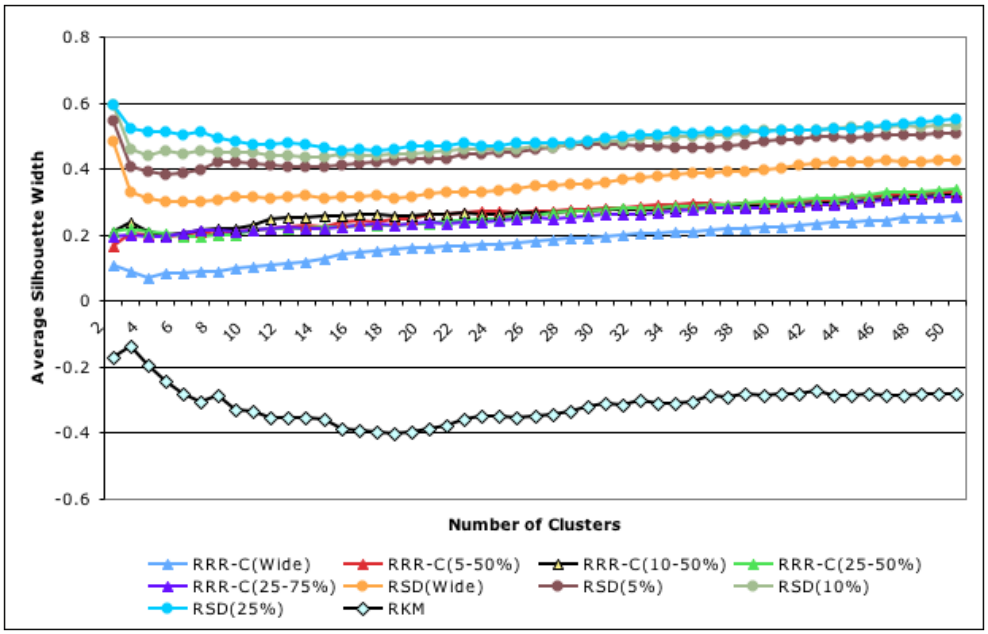


Figure A.28: Average silhouette widths for Mutagenesis$_{All}$

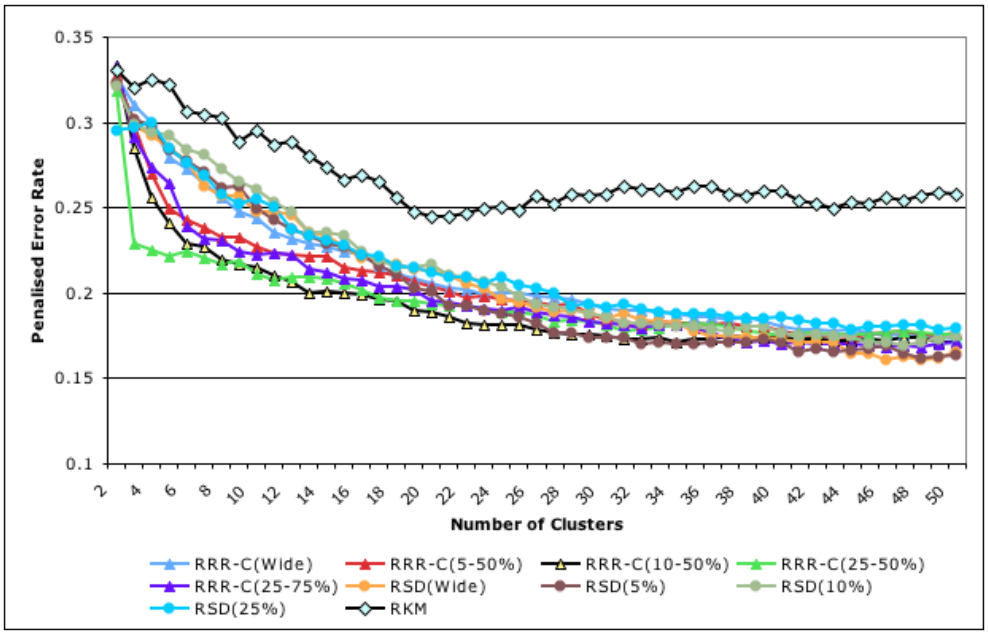


Figure A.29: Penalised error rates on Mutagenesis$_{RF}$

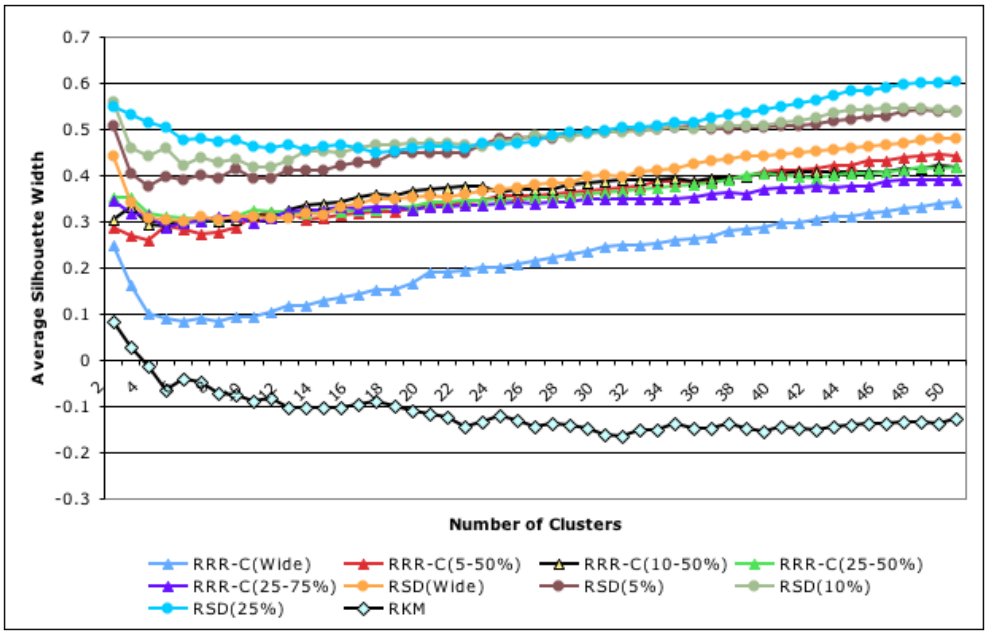


Figure A.30: Average silhouette widths for Mutagenesis$_{RF}$

A.3 Detailed RRR-RF Results

Table A.1: Carcinogenesis, Standard root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.556 ± 0.109	0.554 ± 0.085	37.2±8.0	81.3±32.0
10	10	0.563 ± 0.089	0.555 ± 0.070	70.7±13.4	167.1±38.5
10	25	0.571 ± 0.099	0.568 ± 0.084	129.3±22.0	286.3±31.8
10	100	0.567 ± 0.087	0.557 ± 0.080	334.3±63.4	384.1±17.2
10	200	0.555 ± 0.083	0.549 ± 0.081	564.3±102.1	406.4±13.6
25	5	0.568 ± 0.099	0.553 ± 0.085	53.8±8.5	82.1±27.9
25	10	0.566 ± 0.106	0.562 ± 0.092	80.7±12.7	160.9±37.5
25	25	0.587 ± 0.093	0.579 ± 0.076	149.2±22.2	282.0±31.0
25	100	0.582 ± 0.086	0.568 ± 0.080	353.3±59.9	379.7±18.1
25	200	0.571 ± 0.099	0.565 ± 0.089	568.9±98.5	405.2±15.0
100	5	0.591 ± 0.090	0.576 ± 0.069	129.4±9.5	79.1±13.3
100	10	0.602 ± 0.095	0.590 ± 0.077	159.9±14.8	157.6±24.5
100	25	0.589 ± 0.094	0.579 ± 0.085	220.9±22.5	280.8±21.5
100	100	0.588 ± 0.084	0.561 ± 0.071	426.2±57.2	382.9±12.2
100	200	0.592 ± 0.094	0.571 ± 0.082	646.5±106.9	405.1±12.1
200	5	0.595 ± 0.083	0.581 ± 0.069	230.6±7.6	80.8±11.0
200	10	0.607 ± 0.085	0.588 ± 0.076	259.7±12.7	159.6±17.5
200	25	0.614 ± 0.079	0.589 ± 0.073	323.9±23.7	280.0±19.0
200	100	0.592 ± 0.085	0.566 ± 0.077	528.4±70.2	384.0±12.5
200	200	0.594 ± 0.084	0.571 ± 0.069	733.1±97.9	406.6±9.8
500	5	0.611 ± 0.085	0.587 ± 0.071	528.2±8.5	80.7±6.2
500	10	0.618 ± 0.079	0.602 ± 0.070	554.5±12.3	159.8±11.2
500	25	0.622 ± 0.082	0.595 ± 0.079	621.1±22.8	277.6±13.4
500	100	0.624 ± 0.083	0.598 ± 0.075	822.7±59.5	381.6±8.7
500	200	0.612 ± 0.081	0.580 ± 0.075	1032.1±87.9	405.6±6.6

Table A.2: $\text{Diterpenes}_{52.3}$, Standard root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.852 ± 0.066	0.776 ± 0.072	49.9±11.9	138.7±42.4
10	10	0.902 ± 0.046	0.842 ± 0.058	83.5±11.4	297.8±63.5
10	25	0.903 ± 0.049	0.876 ± 0.048	129.8±25.5	458.2±57.7
10	100	0.902 ± 0.043	0.880 ± 0.046	182.0±50.2	509.7±76.0
10	200	0.914 ± 0.036	0.888 ± 0.039	176.5±51.4	505.6±64.5
25	5	0.888 ± 0.046	0.816 ± 0.053	62.9±9.8	153.1±47.2
25	10	0.923 ± 0.041	0.857 ± 0.049	99.8±15.1	301.6±65.4
25	25	0.925 ± 0.037	0.877 ± 0.042	150.6±27.4	463.4±60.4
25	100	0.938 ± 0.037	0.893 ± 0.042	191.2±46.4	516.2±58.8
25	200	0.925 ± 0.036	0.884 ± 0.039	199.0±48.1	516.2±65.0
100	5	0.951 ± 0.029	0.883 ± 0.044	139.9±11.6	142.1±27.1
100	10	0.967 ± 0.022	0.908 ± 0.039	172.7±15.4	303.6±36.9
100	25	0.971 ± 0.018	0.918 ± 0.035	223.6±21.9	465.1±43.6
100	100	0.972 ± 0.018	0.916 ± 0.032	266.7±36.8	511.8±48.1
100	200	0.970 ± 0.021	0.915 ± 0.035	271.2±48.8	517.7±45.6
200	5	0.973 ± 0.017	0.918 ± 0.035	240.0±10.3	148.7±22.6
200	10	0.981 ± 0.015	0.938 ± 0.031	274.2±13.6	301.3±29.0
200	25	0.985 ± 0.012	0.942 ± 0.028	322.1±23.9	462.1±30.7
200	100	0.985 ± 0.012	0.939 ± 0.027	373.1±45.0	511.2±36.0
200	200	0.986 ± 0.012	0.943 ± 0.024	368.8±46.0	512.5±38.5
500	5	0.981 ± 0.015	0.941 ± 0.029	539.3±10.2	144.1±11.9
500	10	0.990 ± 0.009	0.954 ± 0.024	570.1±13.8	295.3±19.6
500	25	0.993 ± 0.008	0.965 ± 0.020	623.8±26.9	460.0±20.4
500	100	0.993 ± 0.007	0.962 ± 0.023	668.4±45.5	511.2±22.7
500	200	0.994 ± 0.005	0.962 ± 0.020	676.7±49.0	511.3±28.3

Table A.3: $\text{Diterpenes}_{52.54}$, Standard root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.786 ± 0.083	0.722 ± 0.079	51.1±12.0	150.7±49.6
10	10	0.836 ± 0.055	0.769 ± 0.058	91.6±17.1	338.9±77.2
10	25	0.866 ± 0.045	0.825 ± 0.051	149.1±25.7	549.6±67.0
10	100	0.856 ± 0.049	0.830 ± 0.050	206.6±46.0	645.3±77.2
10	200	0.860 ± 0.044	0.833 ± 0.045	223.8±57.3	641.9±81.6
25	5	0.834 ± 0.065	0.758 ± 0.069	67.2±11.7	156.7±49.3
25	10	0.861 ± 0.061	0.793 ± 0.064	107.3±16.4	333.8±66.0
25	25	0.888 ± 0.044	0.831 ± 0.049	160.9±25.1	559.3±57.7
25	100	0.883 ± 0.049	0.837 ± 0.053	235.3±44.2	653.9±64.2
25	200	0.889 ± 0.045	0.846 ± 0.044	231.3±59.3	635.6±70.3
100	5	0.906 ± 0.050	0.824 ± 0.056	142.4±10.7	155.3±32.5
100	10	0.923 ± 0.035	0.850 ± 0.046	180.3±15.4	329.7±44.7
100	25	0.937 ± 0.036	0.866 ± 0.048	240.9±23.9	554.1±45.5
100	100	0.936 ± 0.031	0.868 ± 0.038	299.8±50.4	641.1±53.5
100	200	0.943 ± 0.032	0.878 ± 0.047	304.1±51.0	636.6±52.6
200	5	0.931 ± 0.037	0.850 ± 0.055	243.6±13.0	152.7±22.3
200	10	0.959 ± 0.022	0.892 ± 0.040	279.3±14.7	334.2±30.4
200	25	0.960 ± 0.023	0.895 ± 0.040	340.0±26.1	548.6±31.2
200	100	0.965 ± 0.019	0.902 ± 0.037	405.8±42.5	638.9±43.2
200	200	0.962 ± 0.019	0.897 ± 0.034	410.9±59.4	644.8±38.5
500	5	0.960 ± 0.023	0.891 ± 0.042	542.1±11.4	152.3±13.5
500	10	0.976 ± 0.016	0.921 ± 0.032	581.6±15.3	332.5±19.0
500	25	0.982 ± 0.012	0.934 ± 0.029	635.3±25.4	555.3±20.8
500	100	0.982 ± 0.013	0.933 ± 0.028	696.0±39.6	640.1±25.6
500	200	0.982 ± 0.013	0.935 ± 0.028	709.1±50.9	641.1±28.5

Table A.4: $\text{Diterpenes}_{54.3}$, Standard root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.881 ± 0.062	0.810 ± 0.072	49.6±10.7	127.8±39.5
10	10	0.910 ± 0.052	0.858 ± 0.063	80.5±16.7	255.5±59.2
10	25	0.935 ± 0.043	0.898 ± 0.045	131.4±24.6	382.6±61.4
10	100	0.923 ± 0.040	0.900 ± 0.046	171.7±45.2	410.1±69.0
10	200	0.922 ± 0.047	0.900 ± 0.046	166.9±40.7	413.2±75.8
25	5	0.917 ± 0.051	0.852 ± 0.065	65.6±12.1	125.1±36.8
25	10	0.933 ± 0.043	0.867 ± 0.059	98.0±15.4	254.0±53.1
25	25	0.943 ± 0.039	0.898 ± 0.050	147.6±26.0	387.9±51.5
25	100	0.944 ± 0.037	0.902 ± 0.049	183.8±41.0	412.0±68.4
25	200	0.942 ± 0.034	0.901 ± 0.044	191.2±48.6	416.8±64.1
100	5	0.965 ± 0.028	0.910 ± 0.045	141.9±12.2	127.0±24.0
100	10	0.978 ± 0.017	0.931 ± 0.034	170.9±16.4	251.9±30.7
100	25	0.979 ± 0.020	0.933 ± 0.035	220.0±24.3	376.6±37.0
100	100	0.981 ± 0.018	0.940 ± 0.031	257.2±44.2	410.9±49.2
100	200	0.979 ± 0.016	0.938 ± 0.030	271.7±49.8	414.2±46.5
200	5	0.981 ± 0.018	0.940 ± 0.029	238.1±10.7	127.5±16.9
200	10	0.988 ± 0.011	0.954 ± 0.026	271.1±16.2	246.4±24.3
200	25	0.991 ± 0.010	0.956 ± 0.029	318.1±27.0	386.8±35.2
200	100	0.990 ± 0.012	0.958 ± 0.027	357.6±44.4	416.5±38.8
200	200	0.990 ± 0.012	0.953 ± 0.026	366.1±51.2	422.6±35.5
500	5	0.990 ± 0.011	0.961 ± 0.025	539.6±9.4	128.8±10.5
500	10	0.993 ± 0.009	0.969 ± 0.022	575.5±18.8	252.6±14.7
500	25	0.996 ± 0.006	0.974 ± 0.018	614.3±22.2	380.8±21.8
500	100	0.996 ± 0.006	0.974 ± 0.019	660.6±44.5	416.9±23.3
500	200	0.996 ± 0.006	0.976 ± 0.018	669.4±45.4	413.1±22.8

Table A.5: Musk_1, Standard root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.716 ± 0.162	0.650 ± 0.148	34.3±5.2	82.6±10.7
10	10	0.727 ± 0.189	0.666 ± 0.165	42.8±6.2	101.2±8.8
10	25	0.705 ± 0.172	0.655 ± 0.168	52.8±10.5	106.6±8.6
10	100	0.738 ± 0.178	0.677 ± 0.151	55.0±14.3	109.3±8.8
10	200	0.748 ± 0.170	0.693 ± 0.168	54.1±15.1	108.2±9.6
25	5	0.748 ± 0.160	0.680 ± 0.146	49.5±4.5	84.2±7.9
25	10	0.753 ± 0.170	0.673 ± 0.166	56.8±6.0	101.9±6.8
25	25	0.733 ± 0.169	0.661 ± 0.158	66.5±11.0	107.7±8.3
25	100	0.769 ± 0.159	0.705 ± 0.144	69.4±14.6	109.4±7.2
25	200	0.757 ± 0.169	0.702 ± 0.163	69.5±14.8	109.1±7.4
100	5	0.843 ± 0.135	0.765 ± 0.135	123.5±4.9	84.2±4.6
100	10	0.859 ± 0.119	0.780 ± 0.117	131.5±6.0	101.9±3.8
100	25	0.846 ± 0.133	0.754 ± 0.134	140.0±8.4	108.2±4.3
100	100	0.844 ± 0.122	0.772 ± 0.128	144.7±12.5	109.9±4.3
100	200	0.841 ± 0.142	0.761 ± 0.134	143.4±12.5	109.0±4.2
200	5	0.889 ± 0.114	0.787 ± 0.132	224.3±4.8	84.6±3.4
200	10	0.874 ± 0.115	0.796 ± 0.128	231.5±6.3	101.6±2.8
200	25	0.872 ± 0.120	0.782 ± 0.119	240.0±9.9	108.6±3.2
200	100	0.882 ± 0.121	0.796 ± 0.132	243.7±14.8	109.7±3.1
200	200	0.871 ± 0.106	0.788 ± 0.113	244.2±12.9	109.1±2.8
500	5	0.911 ± 0.094	0.810 ± 0.112	524.4±4.7	84.6±2.2
500	10	0.926 ± 0.076	0.834 ± 0.106	532.8±6.9	101.8±2.1
500	25	0.908 ± 0.088	0.808 ± 0.116	540.0±9.1	108.3±2.3
500	100	0.920 ± 0.096	0.819 ± 0.112	546.8±15.9	109.5±2.3
500	200	0.915 ± 0.082	0.823 ± 0.108	543.6±14.1	109.4±2.3

Table A.6: Mutagenesis_{All}, Standard root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.713 ± 0.119	0.683 ± 0.086	39.7±8.2	52.8±16.0
10	10	0.740 ± 0.102	0.716 ± 0.088	65.1±13.2	92.3±20.9
10	25	0.761 ± 0.105	0.734 ± 0.093	125.8±27.4	148.8±19.4
10	100	0.740 ± 0.102	0.725 ± 0.090	304.7±58.5	199.5±14.8
10	200	0.750 ± 0.109	0.728 ± 0.097	488.5±83.6	209.6±13.2
25	5	0.747 ± 0.105	0.710 ± 0.085	54.8±8.4	54.4±16.1
25	10	0.768 ± 0.097	0.737 ± 0.091	80.6±13.0	90.4±16.9
25	25	0.759 ± 0.108	0.738 ± 0.088	137.4±23.4	144.9±17.1
25	100	0.760 ± 0.093	0.730 ± 0.077	311.7±51.5	200.7±13.2
25	200	0.767 ± 0.096	0.731 ± 0.092	500.4±88.8	211.1±11.5
100	5	0.791 ± 0.102	0.738 ± 0.092	129.6±7.8	52.5±7.3
100	10	0.791 ± 0.102	0.749 ± 0.095	154.1±13.0	92.3±11.4
100	25	0.790 ± 0.105	0.759 ± 0.094	213.1±24.2	146.6±15.7
100	100	0.788 ± 0.101	0.747 ± 0.102	411.4±60.1	199.1±10.4
100	200	0.783 ± 0.082	0.738 ± 0.077	570.3±87.3	208.8±8.5
200	5	0.801 ± 0.090	0.748 ± 0.088	227.6±8.3	52.6±5.2
200	10	0.814 ± 0.092	0.769 ± 0.083	256.3±12.4	90.3±9.2
200	25	0.807 ± 0.097	0.770 ± 0.090	312.1±20.9	145.0±10.9
200	100	0.802 ± 0.094	0.750 ± 0.089	490.0±56.9	200.4±9.4
200	200	0.798 ± 0.094	0.757 ± 0.090	688.8±91.8	209.5±8.1
500	5	0.811 ± 0.090	0.760 ± 0.079	529.1±9.0	52.8±3.4
500	10	0.818 ± 0.092	0.768 ± 0.085	555.5±13.4	90.5±6.4
500	25	0.823 ± 0.093	0.772 ± 0.084	614.9±23.3	147.6±8.3
500	100	0.812 ± 0.091	0.757 ± 0.084	796.0±52.8	200.5±6.7
500	200	0.813 ± 0.091	0.763 ± 0.086	970.1±88.4	210.0±6.3

Table A.7: Mutagenesis$_{RF}$, Standard root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.762 ± 0.135	0.743 ± 0.096	38.0±8.3	46.7±14.5
10	10	0.795 ± 0.123	0.784 ± 0.091	61.8±13.4	79.7±17.0
10	25	0.807 ± 0.118	0.786 ± 0.102	113.3±20.5	121.7±13.8
10	100	0.803 ± 0.106	0.783 ± 0.099	273.1±47.5	152.5±11.4
10	200	0.803 ± 0.103	0.783 ± 0.093	439.3±90.5	159.1±11.9
25	5	0.790 ± 0.114	0.754 ± 0.085	53.2±7.8	48.6±11.1
25	10	0.823 ± 0.098	0.781 ± 0.081	78.9±13.3	80.0±13.4
25	25	0.828 ± 0.103	0.794 ± 0.093	129.7±23.0	118.5±15.1
25	100	0.817 ± 0.099	0.790 ± 0.085	287.6±49.6	151.6±10.7
25	200	0.828 ± 0.096	0.788 ± 0.081	426.3±86.3	156.9±8.8
100	5	0.845 ± 0.103	0.782 ± 0.088	126.8±7.3	48.5±6.2
100	10	0.870 ± 0.086	0.814 ± 0.085	152.0±12.9	79.3±9.5
100	25	0.876 ± 0.092	0.822 ± 0.092	204.9±25.7	118.8±11.1
100	100	0.859 ± 0.088	0.808 ± 0.078	359.2±54.3	152.4±8.6
100	200	0.863 ± 0.087	0.807 ± 0.085	510.2±85.0	156.2±8.3
200	5	0.866 ± 0.094	0.802 ± 0.092	227.8±7.3	49.0±4.9
200	10	0.874 ± 0.080	0.820 ± 0.077	251.0±12.5	79.9±8.0
200	25	0.887 ± 0.087	0.822 ± 0.093	306.0±23.1	120.0±7.2
200	100	0.879 ± 0.079	0.811 ± 0.084	464.8±56.4	151.3±7.2
200	200	0.869 ± 0.078	0.810 ± 0.074	606.6±74.7	156.7±6.7
500	5	0.869 ± 0.085	0.812 ± 0.085	527.0±7.7	49.0±3.2
500	10	0.889 ± 0.079	0.827 ± 0.077	553.3±13.1	80.1±5.2
500	25	0.893 ± 0.076	0.824 ± 0.082	608.4±25.9	120.4±6.0
500	100	0.892 ± 0.075	0.834 ± 0.079	760.7±51.5	152.3±4.8
500	200	0.883 ± 0.080	0.825 ± 0.082	907.9±70.8	156.9±5.0

Table A.8: Carcinogenesis, Standard root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.570 ± 0.095	0.555 ± 0.069	40.4±9.2	87.8±30.9
10	10	0.573 ± 0.098	0.561 ± 0.089	68.0±12.0	169.6±36.5
10	25	0.568 ± 0.097	0.565 ± 0.085	133.2±23.3	291.8±31.2
10	100	0.580 ± 0.099	0.568 ± 0.092	332.9±58.4	388.5±17.3
10	200	0.576 ± 0.089	0.565 ± 0.084	547.0±111.6	408.0±14.8
25	5	0.552 ± 0.096	0.556 ± 0.080	55.1±8.4	87.1±27.0
25	10	0.587 ± 0.100	0.581 ± 0.086	83.7±14.1	169.1±36.2
25	25	0.593 ± 0.092	0.588 ± 0.080	147.2±20.7	294.8±27.3
25	100	0.587 ± 0.098	0.569 ± 0.082	333.0±58.8	387.1±15.5
25	200	0.569 ± 0.084	0.553 ± 0.081	532.6±87.6	410.8±13.2
100	5	0.589 ± 0.100	0.571 ± 0.074	128.9±8.6	85.7±15.3
100	10	0.605 ± 0.099	0.581 ± 0.077	161.3±13.0	171.0±22.8
100	25	0.595 ± 0.096	0.580 ± 0.079	222.3±19.3	291.7±22.6
100	100	0.595 ± 0.088	0.563 ± 0.078	403.8±50.6	389.1±13.3
100	200	0.584 ± 0.091	0.559 ± 0.083	642.2±85.7	409.3±11.2
200	5	0.599 ± 0.089	0.574 ± 0.083	228.9±8.3	88.5±11.9
200	10	0.608 ± 0.091	0.588 ± 0.069	258.9±15.0	170.7±17.0
200	25	0.614 ± 0.096	0.591 ± 0.077	322.6±20.3	290.6±16.8
200	100	0.599 ± 0.086	0.571 ± 0.074	515.5±56.5	389.0±12.1
200	200	0.596 ± 0.087	0.574 ± 0.079	721.3±82.7	407.5±8.7
500	5	0.609 ± 0.089	0.587 ± 0.077	530.8±9.5	87.4±6.1
500	10	0.624 ± 0.086	0.599 ± 0.080	558.2±14.9	170.4±12.1
500	25	0.626 ± 0.077	0.603 ± 0.075	622.6±24.1	289.5±13.9
500	100	0.618 ± 0.079	0.586 ± 0.074	821.2±53.0	390.0±7.4
500	200	0.595 ± 0.096	0.568 ± 0.082	1029.0±109.2	409.6±6.9

Table A.9: Diterpenes$_{52.3}$, Standard root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.879 ± 0.055	0.803 ± 0.060	52.0±10.5	160.5±46.6
10	10	0.917 ± 0.044	0.863 ± 0.051	84.2±15.0	319.3±70.1
10	25	0.913 ± 0.043	0.876 ± 0.045	124.6±22.6	467.7±54.4
10	100	0.915 ± 0.034	0.883 ± 0.040	153.5±41.0	491.6±61.9
10	200	0.921 ± 0.036	0.892 ± 0.039	160.8±40.3	500.7±65.6
25	5	0.914 ± 0.042	0.841 ± 0.055	66.0±11.0	166.2±44.9
25	10	0.937 ± 0.034	0.873 ± 0.042	99.5±14.7	320.6±62.7
25	25	0.940 ± 0.036	0.890 ± 0.039	142.3±22.0	460.9±55.1
25	100	0.941 ± 0.031	0.890 ± 0.041	171.7±35.8	502.7±68.1
25	200	0.941 ± 0.032	0.889 ± 0.044	177.0±39.1	499.0±60.0
100	5	0.959 ± 0.024	0.899 ± 0.042	142.8±11.1	161.4±22.6
100	10	0.976 ± 0.017	0.924 ± 0.032	174.2±16.3	319.8±37.1
100	25	0.977 ± 0.017	0.923 ± 0.033	213.8±20.5	463.2±44.1
100	100	0.977 ± 0.016	0.926 ± 0.029	250.0±34.9	499.3±40.2
100	200	0.977 ± 0.017	0.927 ± 0.031	252.5±44.7	506.2±45.0
200	5	0.976 ± 0.017	0.925 ± 0.031	242.3±10.7	162.2±20.9
200	10	0.987 ± 0.011	0.948 ± 0.025	274.4±14.9	319.8±26.6
200	25	0.988 ± 0.011	0.945 ± 0.027	314.6±22.2	465.2±30.4
200	100	0.987 ± 0.009	0.946 ± 0.025	348.6±37.3	499.0±34.2
200	200	0.989 ± 0.010	0.951 ± 0.027	349.4±41.4	502.3±36.6
500	5	0.985 ± 0.012	0.943 ± 0.028	543.0±10.2	166.0±13.8
500	10	0.991 ± 0.009	0.957 ± 0.022	574.5±15.4	323.5±17.7
500	25	0.994 ± 0.006	0.966 ± 0.020	615.4±23.9	469.2±20.9
500	100	0.994 ± 0.006	0.966 ± 0.020	652.4±37.9	500.6±23.6
500	200	0.994 ± 0.005	0.966 ± 0.018	654.1±46.7	500.1±24.0

Table A.10: Diterpenes$_{52.54}$, Standard root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.805 ± 0.070	0.736 ± 0.069	53.5±11.5	170.5±53.9
10	10	0.859 ± 0.051	0.790 ± 0.061	92.5±17.9	360.9±76.2
10	25	0.875 ± 0.049	0.827 ± 0.048	147.2±23.2	571.7±65.6
10	100	0.870 ± 0.043	0.834 ± 0.042	189.6±40.0	648.7±71.3
10	200	0.872 ± 0.048	0.838 ± 0.045	197.1±51.4	637.1±76.1
25	5	0.847 ± 0.065	0.766 ± 0.065	71.3±10.7	172.4±50.4
25	10	0.869 ± 0.056	0.798 ± 0.056	108.5±15.1	358.5±66.1
25	25	0.900 ± 0.045	0.842 ± 0.049	156.6±25.2	573.1±57.8
25	100	0.901 ± 0.040	0.844 ± 0.043	208.4±43.2	644.5±71.8
25	200	0.904 ± 0.042	0.848 ± 0.045	213.8±49.4	633.3±67.5
100	5	0.917 ± 0.037	0.831 ± 0.051	146.9±12.5	171.8±31.4
100	10	0.943 ± 0.031	0.869 ± 0.044	182.5±16.7	362.2±48.0
100	25	0.951 ± 0.028	0.882 ± 0.043	231.1±24.6	561.8±38.1
100	100	0.951 ± 0.027	0.881 ± 0.041	286.9±46.4	635.3±48.0
100	200	0.944 ± 0.031	0.881 ± 0.038	291.2±46.7	631.8±46.5
200	5	0.947 ± 0.028	0.870 ± 0.044	246.7±11.7	171.1±21.6
200	10	0.967 ± 0.020	0.909 ± 0.035	282.9±17.5	366.1±34.8
200	25	0.971 ± 0.019	0.913 ± 0.037	329.2±22.9	567.9±33.7
200	100	0.969 ± 0.017	0.910 ± 0.035	381.1±44.7	637.2±36.1
200	200	0.969 ± 0.018	0.912 ± 0.031	387.9±44.7	631.4±35.1
500	5	0.964 ± 0.021	0.898 ± 0.034	543.8±11.2	178.4±15.2
500	10	0.982 ± 0.013	0.938 ± 0.027	583.9±14.4	366.6±22.3
500	25	0.984 ± 0.013	0.940 ± 0.027	635.5±23.5	569.5±21.2
500	100	0.986 ± 0.011	0.944 ± 0.027	681.8±43.0	632.7±27.6
500	200	0.985 ± 0.010	0.941 ± 0.025	688.7±47.4	634.4±25.4

Table A.11: $\text{Diterpenes}_{54.3}$, Standard root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.909 ± 0.049	0.839 ± 0.061	53.9±11.3	148.7±45.4
10	10	0.932 ± 0.042	0.881 ± 0.061	85.3±15.9	270.9±63.7
10	25	0.942 ± 0.033	0.909 ± 0.038	124.3±24.4	370.8±59.7
10	100	0.930 ± 0.040	0.904 ± 0.043	153.8±37.7	412.7±65.9
10	200	0.936 ± 0.035	0.910 ± 0.040	160.8±39.6	406.7±61.7
25	5	0.926 ± 0.048	0.860 ± 0.064	68.2±11.6	140.0±35.6
25	10	0.951 ± 0.037	0.893 ± 0.053	99.2±14.8	267.4±54.7
25	25	0.959 ± 0.027	0.911 ± 0.041	136.8±23.7	375.6±54.4
25	100	0.960 ± 0.028	0.917 ± 0.043	158.8±31.8	402.3±59.9
25	200	0.958 ± 0.028	0.913 ± 0.037	166.6±40.3	399.5±60.0
100	5	0.975 ± 0.023	0.926 ± 0.038	143.0±9.8	148.2±24.8
100	10	0.982 ± 0.014	0.938 ± 0.031	171.9±16.4	266.6±34.1
100	25	0.987 ± 0.014	0.948 ± 0.030	213.4±23.4	373.9±37.8
100	100	0.984 ± 0.015	0.945 ± 0.029	239.4±35.4	405.8±45.9
100	200	0.985 ± 0.013	0.947 ± 0.031	248.0±40.3	403.5±41.8
200	5	0.986 ± 0.014	0.948 ± 0.027	242.2±10.9	145.2±17.8
200	10	0.989 ± 0.012	0.957 ± 0.027	274.3±15.4	265.6±26.2
200	25	0.992 ± 0.009	0.963 ± 0.025	312.2±21.7	376.4±28.8
200	100	0.993 ± 0.009	0.964 ± 0.024	341.0±37.3	399.6±30.5
200	200	0.993 ± 0.007	0.966 ± 0.020	346.9±43.9	401.2±31.7
500	5	0.991 ± 0.010	0.966 ± 0.023	542.2±11.2	146.4±11.2
500	10	0.994 ± 0.007	0.973 ± 0.019	573.0±16.2	272.0±16.3
500	25	0.996 ± 0.006	0.979 ± 0.017	611.5±22.3	375.7±18.9
500	100	0.996 ± 0.005	0.976 ± 0.017	634.7±33.3	404.0±23.3
500	200	0.997 ± 0.005	0.976 ± 0.018	637.3±40.9	402.2±21.9

Table A.12: Musk_1, Standard root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.743 ± 0.167	0.672 ± 0.143	35.5±4.8	91.6±9.9
10	10	0.710 ± 0.183	0.658 ± 0.159	40.7±5.2	105.2±8.4
10	25	0.733 ± 0.164	0.671 ± 0.159	45.8±9.1	108.0±8.0
10	100	0.744 ± 0.168	0.679 ± 0.149	49.3±11.6	109.1±8.1
10	200	0.728 ± 0.182	0.676 ± 0.151	48.5±13.6	109.6±10.2
25	5	0.785 ± 0.181	0.713 ± 0.154	49.3±3.9	92.9±7.0
25	10	0.757 ± 0.158	0.698 ± 0.147	54.6±5.8	105.5±7.2
25	25	0.777 ± 0.151	0.705 ± 0.136	60.3±8.5	110.2±6.0
25	100	0.793 ± 0.144	0.715 ± 0.136	61.2±10.2	109.0±6.6
25	200	0.780 ± 0.156	0.708 ± 0.145	62.8±13.4	108.5±6.9
100	5	0.866 ± 0.121	0.772 ± 0.117	124.6±4.4	93.0±4.3
100	10	0.873 ± 0.114	0.777 ± 0.124	129.8±6.8	105.5±3.5
100	25	0.887 ± 0.118	0.797 ± 0.123	135.1±8.5	110.2±3.1
100	100	0.864 ± 0.120	0.783 ± 0.117	136.1±12.3	109.1±3.8
100	200	0.882 ± 0.109	0.804 ± 0.115	138.8±11.8	109.5±3.2
200	5	0.902 ± 0.087	0.809 ± 0.111	224.5±4.2	92.9±2.8
200	10	0.888 ± 0.103	0.786 ± 0.117	229.0±5.2	105.4±2.6
200	25	0.894 ± 0.114	0.795 ± 0.130	235.8±9.1	109.7±2.4
200	100	0.889 ± 0.103	0.804 ± 0.131	236.8±12.3	109.5±2.8
200	200	0.901 ± 0.107	0.808 ± 0.112	237.6±12.4	109.3±2.9
500	5	0.915 ± 0.083	0.824 ± 0.102	524.3±4.5	92.7±2.1
500	10	0.923 ± 0.083	0.824 ± 0.105	529.1±5.7	105.2±1.9
500	25	0.911 ± 0.090	0.805 ± 0.114	535.3±8.2	109.3±2.2
500	100	0.929 ± 0.079	0.813 ± 0.109	538.8±12.0	109.5±2.1
500	200	0.915 ± 0.083	0.816 ± 0.102	538.5±12.1	109.7±2.1

Table A.13: Mutagenesis$_{All}$, Standard root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.720 ± 0.124	0.693 ± 0.095	38.9±8.5	54.1±15.4
10	10	0.757 ± 0.105	0.737 ± 0.094	66.2±12.6	92.0±19.1
10	25	0.755 ± 0.105	0.734 ± 0.099	124.6±28.3	145.2±20.1
10	100	0.746 ± 0.104	0.717 ± 0.103	306.4±50.2	201.2±14.0
10	200	0.747 ± 0.104	0.720 ± 0.099	467.7±76.0	209.5±12.1
25	5	0.753 ± 0.099	0.705 ± 0.096	53.1±8.1	53.4±14.8
25	10	0.748 ± 0.123	0.725 ± 0.096	78.6±13.3	91.1±19.0
25	25	0.764 ± 0.108	0.737 ± 0.097	139.6±24.0	148.0±19.7
25	100	0.770 ± 0.109	0.741 ± 0.093	323.8±57.3	202.0±13.5
25	200	0.763 ± 0.102	0.733 ± 0.095	487.3±82.5	209.0±11.3
100	5	0.791 ± 0.099	0.732 ± 0.092	127.4±8.9	53.3±7.4
100	10	0.792 ± 0.099	0.747 ± 0.085	154.0±13.8	92.4±11.8
100	25	0.805 ± 0.099	0.751 ± 0.092	211.3±27.4	148.8±12.9
100	100	0.779 ± 0.092	0.737 ± 0.087	399.5±53.3	200.1±10.6
100	200	0.786 ± 0.094	0.751 ± 0.087	568.0±79.9	210.7±8.5
200	5	0.799 ± 0.099	0.753 ± 0.084	229.6±9.3	53.3±5.7
200	10	0.810 ± 0.091	0.767 ± 0.082	256.1±12.1	92.4±10.1
200	25	0.814 ± 0.091	0.768 ± 0.089	309.1±20.4	146.9±11.5
200	100	0.809 ± 0.098	0.751 ± 0.093	494.3±50.4	201.1±8.8
200	200	0.798 ± 0.096	0.752 ± 0.085	676.9±89.2	208.5±8.4
500	5	0.814 ± 0.099	0.749 ± 0.093	529.5±7.9	53.9±4.3
500	10	0.818 ± 0.093	0.769 ± 0.083	554.0±13.5	91.9±6.1
500	25	0.822 ± 0.084	0.770 ± 0.085	616.9±24.4	147.2±7.4
500	100	0.813 ± 0.088	0.764 ± 0.084	797.8±62.5	200.3±7.0
500	200	0.808 ± 0.084	0.759 ± 0.080	965.6±83.3	210.3±6.8

Table A.14: Mutagenesis$_{RF}$, Standard root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.768 ± 0.147	0.745 ± 0.097	37.6±6.7	49.6±14.0
10	10	0.798 ± 0.112	0.779 ± 0.079	61.2±14.3	80.0±15.0
10	25	0.810 ± 0.112	0.792 ± 0.091	115.2±22.0	122.5±14.7
10	100	0.794 ± 0.121	0.773 ± 0.103	266.3±52.0	153.0±11.8
10	200	0.801 ± 0.107	0.786 ± 0.097	414.3±77.8	155.3±10.0
25	5	0.807 ± 0.119	0.772 ± 0.095	52.9±7.0	51.7±12.5
25	10	0.834 ± 0.102	0.787 ± 0.091	78.1±13.0	79.0±12.8
25	25	0.822 ± 0.107	0.805 ± 0.081	128.7±24.0	120.2±15.7
25	100	0.830 ± 0.097	0.796 ± 0.088	285.9±53.9	151.1±11.4
25	200	0.810 ± 0.101	0.785 ± 0.084	414.9±79.0	156.3±10.2
100	5	0.857 ± 0.089	0.782 ± 0.098	127.8±9.1	48.0±6.4
100	10	0.867 ± 0.088	0.809 ± 0.086	153.8±12.5	81.0±9.5
100	25	0.864 ± 0.090	0.812 ± 0.084	199.1±22.7	120.1±11.4
100	100	0.857 ± 0.095	0.812 ± 0.081	359.4±57.3	153.0±8.3
100	200	0.861 ± 0.093	0.805 ± 0.087	491.5±73.6	156.9±8.5
200	5	0.866 ± 0.083	0.798 ± 0.087	229.3±8.3	49.8±5.5
200	10	0.882 ± 0.078	0.823 ± 0.077	251.8±11.4	81.4±7.7
200	25	0.886 ± 0.075	0.828 ± 0.079	304.2±22.3	121.0±8.4
200	100	0.872 ± 0.084	0.815 ± 0.083	446.4±51.3	153.4±6.7
200	200	0.872 ± 0.080	0.806 ± 0.090	612.8±85.5	157.5±7.1
500	5	0.873 ± 0.082	0.818 ± 0.085	528.2±7.8	48.8±3.1
500	10	0.887 ± 0.082	0.829 ± 0.081	553.1±14.3	81.3±5.2
500	25	0.892 ± 0.083	0.833 ± 0.088	604.2±23.8	121.9±5.4
500	100	0.895 ± 0.078	0.831 ± 0.083	752.8±53.5	152.8±5.3
500	200	0.884 ± 0.082	0.833 ± 0.081	896.9±80.8	157.3±4.9

Table A.15: Carcinogenesis, Standard root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.570 ± 0.095	0.555 ± 0.069	40.4±9.2	87.8±30.9
10	10	0.573 ± 0.098	0.561 ± 0.089	68.0±12.0	169.6±36.5
10	25	0.568 ± 0.097	0.565 ± 0.085	133.2±23.3	291.8±31.2
10	100	0.580 ± 0.099	0.568 ± 0.092	332.9±58.4	388.5±17.3
10	200	0.576 ± 0.089	0.565 ± 0.084	547.0±111.6	408.0±14.8
25	5	0.552 ± 0.096	0.556 ± 0.080	55.1±8.4	87.1±27.0
25	10	0.587 ± 0.100	0.581 ± 0.086	83.7±14.1	169.1±36.2
25	25	0.593 ± 0.092	0.588 ± 0.080	147.2±20.7	294.8±27.3
25	100	0.587 ± 0.098	0.569 ± 0.082	333.0±58.8	387.1±15.5
25	200	0.569 ± 0.084	0.553 ± 0.081	532.6±87.6	410.8±13.2
100	5	0.589 ± 0.100	0.571 ± 0.074	128.9±8.6	85.7±15.3
100	10	0.605 ± 0.099	0.581 ± 0.077	161.3±13.0	171.0±22.8
100	25	0.595 ± 0.096	0.580 ± 0.079	222.3±19.3	291.7±22.6
100	100	0.595 ± 0.088	0.563 ± 0.078	403.8±50.6	389.1±13.3
100	200	0.584 ± 0.091	0.559 ± 0.083	642.2±85.7	409.3±11.2
200	5	0.599 ± 0.089	0.574 ± 0.083	228.9±8.3	88.5±11.9
200	10	0.608 ± 0.091	0.588 ± 0.069	258.9±15.0	170.7±17.0
200	25	0.614 ± 0.096	0.591 ± 0.077	322.6±20.3	290.6±16.8
200	100	0.599 ± 0.086	0.571 ± 0.074	515.5±56.5	389.0±12.1
200	200	0.596 ± 0.087	0.574 ± 0.079	721.3±82.7	407.5±8.7
500	5	0.609 ± 0.089	0.587 ± 0.077	530.8±9.5	87.4±6.1
500	10	0.624 ± 0.086	0.599 ± 0.080	558.2±14.9	170.4±12.1
500	25	0.626 ± 0.077	0.603 ± 0.075	622.6±24.1	289.5±13.9
500	100	0.618 ± 0.079	0.586 ± 0.074	821.2±53.0	390.0±7.4
500	200	0.595 ± 0.096	0.568 ± 0.082	1029.0±109.2	409.6±6.9

Table A.16: $\text{Diterpenes}_{52.3}$, Standard root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.879 ± 0.055	0.803 ± 0.060	52.0±10.5	160.5±46.6
10	10	0.917 ± 0.044	0.863 ± 0.051	84.2±15.0	319.3±70.1
10	25	0.913 ± 0.043	0.876 ± 0.045	124.6±22.6	467.7±54.4
10	100	0.915 ± 0.034	0.883 ± 0.040	153.5±41.0	491.6±61.9
10	200	0.921 ± 0.036	0.892 ± 0.039	160.8±40.3	500.7±65.6
25	5	0.914 ± 0.042	0.841 ± 0.055	66.0±11.0	166.2±44.9
25	10	0.937 ± 0.034	0.873 ± 0.042	99.5±14.7	320.6±62.7
25	25	0.940 ± 0.036	0.890 ± 0.039	142.3±22.0	460.9±55.1
25	100	0.941 ± 0.031	0.890 ± 0.041	171.7±35.8	502.7±68.1
25	200	0.941 ± 0.032	0.889 ± 0.044	177.0±39.1	499.0±60.0
100	5	0.959 ± 0.024	0.899 ± 0.042	142.8±11.1	161.4±22.6
100	10	0.976 ± 0.017	0.924 ± 0.032	174.2±16.3	319.8±37.1
100	25	0.977 ± 0.017	0.923 ± 0.033	213.8±20.5	463.2±44.1
100	100	0.977 ± 0.016	0.926 ± 0.029	250.0±34.9	499.3±40.2
100	200	0.977 ± 0.017	0.927 ± 0.031	252.5±44.7	506.2±45.0
200	5	0.976 ± 0.017	0.925 ± 0.031	242.3±10.7	162.2±20.9
200	10	0.987 ± 0.011	0.948 ± 0.025	274.4±14.9	319.8±26.6
200	25	0.988 ± 0.011	0.945 ± 0.027	314.6±22.2	465.2±30.4
200	100	0.987 ± 0.009	0.946 ± 0.025	348.6±37.3	499.0±34.2
200	200	0.989 ± 0.010	0.951 ± 0.027	349.4±41.4	502.3±36.6
500	5	0.985 ± 0.012	0.943 ± 0.028	543.0±10.2	166.0±13.8
500	10	0.991 ± 0.009	0.957 ± 0.022	574.5±15.4	323.5±17.7
500	25	0.994 ± 0.006	0.966 ± 0.020	615.4±23.9	469.2±20.9
500	100	0.994 ± 0.006	0.966 ± 0.020	652.4±37.9	500.6±23.6
500	200	0.994 ± 0.005	0.966 ± 0.018	654.1±46.7	500.1±24.0

Table A.17: Diterpenes$_{52.54}$, Standard root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.805 ± 0.070	0.736 ± 0.069	53.5±11.5	170.5±53.9
10	10	0.859 ± 0.051	0.790 ± 0.061	92.5±17.9	360.9±76.2
10	25	0.875 ± 0.049	0.827 ± 0.048	147.2±23.2	571.7±65.6
10	100	0.870 ± 0.043	0.834 ± 0.042	189.6±40.0	648.7±71.3
10	200	0.872 ± 0.048	0.838 ± 0.045	197.1±51.4	637.1±76.1
25	5	0.847 ± 0.065	0.766 ± 0.065	71.3±10.7	172.4±50.4
25	10	0.869 ± 0.056	0.798 ± 0.056	108.5±15.1	358.5±66.1
25	25	0.900 ± 0.045	0.842 ± 0.049	156.6±25.2	573.1±57.8
25	100	0.901 ± 0.040	0.844 ± 0.043	208.4±43.2	644.5±71.8
25	200	0.904 ± 0.042	0.848 ± 0.045	213.8±49.4	633.3±67.5
100	5	0.917 ± 0.037	0.831 ± 0.051	146.9±12.5	171.8±31.4
100	10	0.943 ± 0.031	0.869 ± 0.044	182.5±16.7	362.2±48.0
100	25	0.951 ± 0.028	0.882 ± 0.043	231.1±24.6	561.8±38.1
100	100	0.951 ± 0.027	0.881 ± 0.041	286.9±46.4	635.3±48.0
100	200	0.944 ± 0.031	0.881 ± 0.038	291.2±46.7	631.8±46.5
200	5	0.947 ± 0.028	0.870 ± 0.044	246.7±11.7	171.1±21.6
200	10	0.967 ± 0.020	0.909 ± 0.035	282.9±17.5	366.1±34.8
200	25	0.971 ± 0.019	0.913 ± 0.037	329.2±22.9	567.9±33.7
200	100	0.969 ± 0.017	0.910 ± 0.035	381.1±44.7	637.2±36.1
200	200	0.969 ± 0.018	0.912 ± 0.031	387.9±44.7	631.4±35.1
500	5	0.964 ± 0.021	0.898 ± 0.034	543.8±11.2	178.4±15.2
500	10	0.982 ± 0.013	0.938 ± 0.027	583.9±14.4	366.6±22.3
500	25	0.984 ± 0.013	0.940 ± 0.027	635.5±23.5	569.5±21.2
500	100	0.986 ± 0.011	0.944 ± 0.027	681.8±43.0	632.7±27.6
500	200	0.985 ± 0.010	0.941 ± 0.025	688.7±47.4	634.4±25.4

Table A.18: Diterpenes$_{54.3}$, Standard root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.909 ± 0.049	0.839 ± 0.061	53.9±11.3	148.7±45.4
10	10	0.932 ± 0.042	0.881 ± 0.061	85.3±15.9	270.9±63.7
10	25	0.942 ± 0.033	0.909 ± 0.038	124.3±24.4	370.8±59.7
10	100	0.930 ± 0.040	0.904 ± 0.043	153.8±37.7	412.7±65.9
10	200	0.936 ± 0.035	0.910 ± 0.040	160.8±39.6	406.7±61.7
25	5	0.926 ± 0.048	0.860 ± 0.064	68.2±11.6	140.0±35.6
25	10	0.951 ± 0.037	0.893 ± 0.053	99.2±14.8	267.4±54.7
25	25	0.959 ± 0.027	0.911 ± 0.041	136.8±23.7	375.6±54.4
25	100	0.960 ± 0.028	0.917 ± 0.043	158.8±31.8	402.3±59.9
25	200	0.958 ± 0.028	0.913 ± 0.037	166.6±40.3	399.5±60.0
100	5	0.975 ± 0.023	0.926 ± 0.038	143.0±9.8	148.2±24.8
100	10	0.982 ± 0.014	0.938 ± 0.031	171.9±16.4	266.6±34.1
100	25	0.987 ± 0.014	0.948 ± 0.030	213.4±23.4	373.9±37.8
100	100	0.984 ± 0.015	0.945 ± 0.029	239.4±35.4	405.8±45.9
100	200	0.985 ± 0.013	0.947 ± 0.031	248.0±40.3	403.5±41.8
200	5	0.986 ± 0.014	0.948 ± 0.027	242.2±10.9	145.2±17.8
200	10	0.989 ± 0.012	0.957 ± 0.027	274.3±15.4	265.6±26.2
200	25	0.992 ± 0.009	0.963 ± 0.025	312.2±21.7	376.4±28.8
200	100	0.993 ± 0.009	0.964 ± 0.024	341.0±37.3	399.6±30.5
200	200	0.993 ± 0.007	0.966 ± 0.020	346.9±43.9	401.2±31.7
500	5	0.991 ± 0.010	0.966 ± 0.023	542.2±11.2	146.4±11.2
500	10	0.994 ± 0.007	0.973 ± 0.019	573.0±16.2	272.0±16.3
500	25	0.996 ± 0.006	0.979 ± 0.017	611.5±22.3	375.7±18.9
500	100	0.996 ± 0.005	0.976 ± 0.017	634.7±33.3	404.0±23.3
500	200	0.997 ± 0.005	0.976 ± 0.018	637.3±40.9	402.2±21.9

Table A.19: Musk_1, Standard root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.743 ± 0.167	0.672 ± 0.143	35.5±4.8	91.6±9.9
10	10	0.710 ± 0.183	0.658 ± 0.159	40.7±5.2	105.2±8.4
10	25	0.733 ± 0.164	0.671 ± 0.159	45.8±9.1	108.0±8.0
10	100	0.744 ± 0.168	0.679 ± 0.149	49.3±11.6	109.1±8.1
10	200	0.728 ± 0.182	0.676 ± 0.151	48.5±13.6	109.6±10.2
25	5	0.785 ± 0.181	0.713 ± 0.154	49.3±3.9	92.9±7.0
25	10	0.757 ± 0.158	0.698 ± 0.147	54.6±5.8	105.5±7.2
25	25	0.777 ± 0.151	0.705 ± 0.136	60.3±8.5	110.2±6.0
25	100	0.793 ± 0.144	0.715 ± 0.136	61.2±10.2	109.0±6.6
25	200	0.780 ± 0.156	0.708 ± 0.145	62.8±13.4	108.5±6.9
100	5	0.866 ± 0.121	0.772 ± 0.117	124.6±4.4	93.0±4.3
100	10	0.873 ± 0.114	0.777 ± 0.124	129.8±6.8	105.5±3.5
100	25	0.887 ± 0.118	0.797 ± 0.123	135.1±8.5	110.2±3.1
100	100	0.864 ± 0.120	0.783 ± 0.117	136.1±12.3	109.1±3.8
100	200	0.882 ± 0.109	0.804 ± 0.115	138.8±11.8	109.5±3.2
200	5	0.902 ± 0.087	0.809 ± 0.111	224.5±4.2	92.9±2.8
200	10	0.888 ± 0.103	0.786 ± 0.117	229.0±5.2	105.4±2.6
200	25	0.894 ± 0.114	0.795 ± 0.130	235.8±9.1	109.7±2.4
200	100	0.889 ± 0.103	0.804 ± 0.131	236.8±12.3	109.5±2.8
200	200	0.901 ± 0.107	0.808 ± 0.112	237.6±12.4	109.3±2.9
500	5	0.915 ± 0.083	0.824 ± 0.102	524.3±4.5	92.7±2.1
500	10	0.923 ± 0.083	0.824 ± 0.105	529.1±5.7	105.2±1.9
500	25	0.911 ± 0.090	0.805 ± 0.114	535.3±8.2	109.3±2.2
500	100	0.929 ± 0.079	0.813 ± 0.109	538.8±12.0	109.5±2.1
500	200	0.915 ± 0.083	0.816 ± 0.102	538.5±12.1	109.7±2.1

Table A.20: Mutagenesis_{All}, Standard root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.720 ± 0.124	0.693 ± 0.095	38.9±8.5	54.1±15.4
10	10	0.757 ± 0.105	0.737 ± 0.094	66.2±12.6	92.0±19.1
10	25	0.755 ± 0.105	0.734 ± 0.099	124.6±28.3	145.2±20.1
10	100	0.746 ± 0.104	0.717 ± 0.103	306.4±50.2	201.2±14.0
10	200	0.747 ± 0.104	0.720 ± 0.099	467.7±76.0	209.5±12.1
25	5	0.753 ± 0.099	0.705 ± 0.096	53.1±8.1	53.4±14.8
25	10	0.748 ± 0.123	0.725 ± 0.096	78.6±13.3	91.1±19.0
25	25	0.764 ± 0.108	0.737 ± 0.097	139.6±24.0	148.0±19.7
25	100	0.770 ± 0.109	0.741 ± 0.093	323.8±57.3	202.0±13.5
25	200	0.763 ± 0.102	0.733 ± 0.095	487.3±82.5	209.0±11.3
100	5	0.791 ± 0.099	0.732 ± 0.092	127.4±8.9	53.3±7.4
100	10	0.792 ± 0.099	0.747 ± 0.085	154.0±13.8	92.4±11.8
100	25	0.805 ± 0.099	0.751 ± 0.092	211.3±27.4	148.8±12.9
100	100	0.779 ± 0.092	0.737 ± 0.087	399.5±53.3	200.1±10.6
100	200	0.786 ± 0.094	0.751 ± 0.087	568.0±79.9	210.7±8.5
200	5	0.799 ± 0.099	0.753 ± 0.084	229.6±9.3	53.3±5.7
200	10	0.810 ± 0.091	0.767 ± 0.082	256.1±12.1	92.4±10.1
200	25	0.814 ± 0.091	0.768 ± 0.089	309.1±20.4	146.9±11.5
200	100	0.809 ± 0.098	0.751 ± 0.093	494.3±50.4	201.1±8.8
200	200	0.798 ± 0.096	0.752 ± 0.085	676.9±89.2	208.5±8.4
500	5	0.814 ± 0.099	0.749 ± 0.093	529.5±7.9	53.9±4.3
500	10	0.818 ± 0.093	0.769 ± 0.083	554.0±13.5	91.9±6.1
500	25	0.822 ± 0.084	0.770 ± 0.085	616.9±24.4	147.2±7.4
500	100	0.813 ± 0.088	0.764 ± 0.084	797.8±62.5	200.3±7.0
500	200	0.808 ± 0.084	0.759 ± 0.080	965.6±83.3	210.3±6.8

Table A.21: Mutagenesis$_{RF}$, Standard root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.768 ± 0.147	0.745 ± 0.097	37.6±6.7	49.6±14.0
10	10	0.798 ± 0.112	0.779 ± 0.079	61.2±14.3	80.0±15.0
10	25	0.810 ± 0.112	0.792 ± 0.091	115.2±22.0	122.5±14.7
10	100	0.794 ± 0.121	0.773 ± 0.103	266.3±52.0	153.0±11.8
10	200	0.801 ± 0.107	0.786 ± 0.097	414.3±77.8	155.3±10.0
25	5	0.807 ± 0.119	0.772 ± 0.095	52.9±7.0	51.7±12.5
25	10	0.834 ± 0.102	0.787 ± 0.091	78.1±13.0	79.0±12.8
25	25	0.822 ± 0.107	0.805 ± 0.081	128.7±24.0	120.2±15.7
25	100	0.830 ± 0.097	0.796 ± 0.088	285.9±53.9	151.1±11.4
25	200	0.810 ± 0.101	0.785 ± 0.084	414.9±79.0	156.3±10.2
100	5	0.857 ± 0.089	0.782 ± 0.098	127.8±9.1	48.0±6.4
100	10	0.867 ± 0.088	0.809 ± 0.086	153.8±12.5	81.0±9.5
100	25	0.864 ± 0.090	0.812 ± 0.084	199.1±22.7	120.1±11.4
100	100	0.857 ± 0.095	0.812 ± 0.081	359.4±57.3	153.0±8.3
100	200	0.861 ± 0.093	0.805 ± 0.087	491.5±73.6	156.9±8.5
200	5	0.866 ± 0.083	0.798 ± 0.087	229.3±8.3	49.8±5.5
200	10	0.882 ± 0.078	0.823 ± 0.077	251.8±11.4	81.4±7.7
200	25	0.886 ± 0.075	0.828 ± 0.079	304.2±22.3	121.0±8.4
200	100	0.872 ± 0.084	0.815 ± 0.083	446.4±51.3	153.4±6.7
200	200	0.872 ± 0.080	0.806 ± 0.090	612.8±85.5	157.5±7.1
500	5	0.873 ± 0.082	0.818 ± 0.085	528.2±7.8	48.8±3.1
500	10	0.887 ± 0.082	0.829 ± 0.081	553.1±14.3	81.3±5.2
500	25	0.892 ± 0.083	0.833 ± 0.088	604.2±23.8	121.9±5.4
500	100	0.895 ± 0.078	0.831 ± 0.083	752.8±53.5	152.8±5.3
500	200	0.884 ± 0.082	0.833 ± 0.081	896.9±80.8	157.3±4.9

Table A.22: Carcinogenesis, Bagging root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.554 ± 0.104	0.553 ± 0.086	38.6±8.5	63.3±22.7
10	10	0.560 ± 0.099	0.560 ± 0.089	68.4±11.7	109.8±26.1
10	25	0.590 ± 0.105	0.579 ± 0.085	130.3±20.4	190.5±19.8
10	100	0.592 ± 0.099	0.582 ± 0.095	328.9±48.3	254.3±10.6
10	200	0.575 ± 0.103	0.560 ± 0.093	539.8±104.4	263.6±7.9
25	5	0.588 ± 0.090	0.574 ± 0.077	53.0±7.6	61.0±19.0
25	10	0.584 ± 0.084	0.573 ± 0.066	81.8±13.5	119.1±25.3
25	25	0.579 ± 0.109	0.568 ± 0.096	146.9±23.0	192.0±21.2
25	100	0.580 ± 0.097	0.567 ± 0.084	350.3±58.5	254.4±8.3
25	200	0.580 ± 0.092	0.568 ± 0.084	551.0±91.3	264.5±7.3
100	5	0.590 ± 0.099	0.582 ± 0.078	128.4±7.4	60.3±10.4
100	10	0.616 ± 0.088	0.595 ± 0.069	157.3±11.7	116.5±15.7
100	25	0.599 ± 0.086	0.589 ± 0.080	222.9±21.4	190.1±13.8
100	100	0.608 ± 0.093	0.581 ± 0.082	415.4±53.0	253.1±6.6
100	200	0.604 ± 0.097	0.575 ± 0.075	643.7±104.1	265.0±6.0
200	5	0.608 ± 0.091	0.596 ± 0.074	228.8±7.9	62.5±7.9
200	10	0.609 ± 0.081	0.584 ± 0.066	255.5±11.7	116.1±13.0
200	25	0.630 ± 0.077	0.599 ± 0.077	315.7±21.0	190.7±10.7
200	100	0.605 ± 0.090	0.576 ± 0.083	516.4±54.8	254.5±5.7
200	200	0.604 ± 0.091	0.580 ± 0.075	726.0±84.4	265.7±5.2
500	5	0.614 ± 0.095	0.596 ± 0.078	530.1±8.9	61.3±5.1
500	10	0.634 ± 0.084	0.602 ± 0.075	557.3±12.3	114.1±8.1
500	25	0.633 ± 0.084	0.612 ± 0.070	620.8±23.5	193.5±7.6
500	100	0.626 ± 0.087	0.597 ± 0.079	814.6±54.8	253.9±4.9
500	200	0.612 ± 0.086	0.579 ± 0.069	1036.6±99.0	264.8±3.8

Table A.23: $\text{Diterpenes}_{52.3}$, Bagging root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.851 ± 0.066	0.778 ± 0.070	48.9±10.8	107.9±35.6
10	10	0.906 ± 0.040	0.834 ± 0.051	81.8±15.1	216.7±47.8
10	25	0.935 ± 0.036	0.880 ± 0.042	130.7±25.5	326.4±35.9
10	100	0.931 ± 0.035	0.882 ± 0.042	174.0±38.5	346.8±50.8
10	200	0.929 ± 0.038	0.881 ± 0.045	187.3±51.3	362.4±39.7
25	5	0.881 ± 0.051	0.804 ± 0.061	63.0±10.8	108.0±30.3
25	10	0.930 ± 0.033	0.862 ± 0.045	97.7±15.3	215.8±33.2
25	25	0.943 ± 0.034	0.883 ± 0.047	144.6±19.4	329.1±39.2
25	100	0.947 ± 0.027	0.892 ± 0.037	195.9±45.7	362.2±39.8
25	200	0.949 ± 0.032	0.890 ± 0.042	201.4±45.2	362.6±46.0
100	5	0.950 ± 0.032	0.887 ± 0.046	138.1±11.2	109.1±20.2
100	10	0.965 ± 0.024	0.907 ± 0.041	173.7±14.4	216.6±30.1
100	25	0.973 ± 0.018	0.918 ± 0.033	216.9±24.0	328.6±25.6
100	100	0.978 ± 0.014	0.925 ± 0.031	267.6±42.5	361.5±30.7
100	200	0.976 ± 0.016	0.921 ± 0.034	272.1±61.9	363.5±31.0
200	5	0.970 ± 0.021	0.916 ± 0.035	238.6±10.5	109.1±14.5
200	10	0.980 ± 0.016	0.933 ± 0.032	273.5±15.6	214.1±18.4
200	25	0.986 ± 0.010	0.944 ± 0.027	319.0±21.7	327.0±21.5
200	100	0.987 ± 0.010	0.947 ± 0.027	365.2±35.0	362.1±25.6
200	200	0.986 ± 0.011	0.943 ± 0.028	378.7±49.0	362.5±21.2
500	5	0.980 ± 0.016	0.938 ± 0.029	540.8±10.6	107.7±9.7
500	10	0.989 ± 0.010	0.952 ± 0.025	573.3±14.4	216.5±13.3
500	25	0.993 ± 0.006	0.957 ± 0.023	623.0±23.3	326.0±14.7
500	100	0.993 ± 0.007	0.963 ± 0.020	663.8±37.1	360.3±16.2
500	200	0.993 ± 0.007	0.961 ± 0.024	670.4±44.0	360.1±15.0

Table A.24: $\text{Diterpenes}_{52.54}$, Bagging root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.794 ± 0.069	0.723 ± 0.063	51.8±10.5	110.4±35.7
10	10	0.841 ± 0.052	0.770 ± 0.054	90.9±16.8	243.5±51.6
10	25	0.883 ± 0.045	0.819 ± 0.054	146.5±25.4	386.4±41.3
10	100	0.892 ± 0.040	0.836 ± 0.044	212.2±44.2	446.9±42.4
10	200	0.888 ± 0.048	0.833 ± 0.050	226.9±60.5	443.0±45.7
25	5	0.826 ± 0.066	0.748 ± 0.065	65.3±9.9	113.2±35.9
25	10	0.866 ± 0.055	0.790 ± 0.059	103.2±16.4	237.3±50.8
25	25	0.897 ± 0.047	0.829 ± 0.052	163.1±26.1	384.4±40.5
25	100	0.909 ± 0.039	0.844 ± 0.053	235.3±50.4	447.7±46.3
25	200	0.901 ± 0.043	0.833 ± 0.049	243.9±64.0	452.4±44.7
100	5	0.894 ± 0.052	0.812 ± 0.060	141.6±10.7	111.0±18.4
100	10	0.933 ± 0.034	0.857 ± 0.051	180.2±16.0	236.1±30.6
100	25	0.947 ± 0.030	0.881 ± 0.042	232.1±26.0	385.5±28.0
100	100	0.949 ± 0.025	0.884 ± 0.038	294.1±45.5	436.6±35.5
100	200	0.946 ± 0.028	0.878 ± 0.043	317.1±58.2	448.7±35.0
200	5	0.939 ± 0.031	0.855 ± 0.042	241.5±11.3	112.3±14.9
200	10	0.955 ± 0.028	0.889 ± 0.044	281.1±16.5	235.7±23.0
200	25	0.967 ± 0.023	0.907 ± 0.039	337.5±28.1	387.4±20.5
200	100	0.964 ± 0.022	0.901 ± 0.033	401.3±42.7	447.3±25.8
200	200	0.970 ± 0.018	0.910 ± 0.035	420.1±58.5	442.2±24.4
500	5	0.950 ± 0.031	0.874 ± 0.047	542.4±11.0	111.6±10.2
500	10	0.973 ± 0.020	0.917 ± 0.035	581.1±15.9	234.7±18.4
500	25	0.982 ± 0.013	0.935 ± 0.032	639.5±23.0	388.5±12.4
500	100	0.984 ± 0.012	0.938 ± 0.030	706.0±44.0	444.2±15.6
500	200	0.982 ± 0.014	0.936 ± 0.029	722.1±56.7	446.1±15.9

Table A.25: $\text{Diterpenes}_{54.3}$, Bagging root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.862 ± 0.071	0.794 ± 0.076	47.2±9.6	90.6±30.2
10	10	0.918 ± 0.046	0.855 ± 0.063	81.9±14.0	186.2±37.8
10	25	0.941 ± 0.036	0.890 ± 0.047	131.9±21.9	274.4±42.2
10	100	0.943 ± 0.034	0.899 ± 0.039	167.3±41.3	298.0±52.2
10	200	0.942 ± 0.029	0.895 ± 0.039	167.0±38.7	295.6±44.8
25	5	0.909 ± 0.054	0.831 ± 0.070	64.2±9.3	91.3±26.1
25	10	0.941 ± 0.035	0.876 ± 0.057	99.5±15.3	174.3±34.6
25	25	0.961 ± 0.028	0.912 ± 0.041	137.7±22.3	263.2±45.7
25	100	0.962 ± 0.027	0.917 ± 0.038	180.9±41.7	281.3±38.8
25	200	0.960 ± 0.029	0.910 ± 0.038	187.9±50.0	296.6±41.4
100	5	0.966 ± 0.026	0.911 ± 0.045	137.1±10.1	93.7±14.9
100	10	0.979 ± 0.017	0.934 ± 0.036	171.8±14.2	183.8±22.5
100	25	0.980 ± 0.017	0.934 ± 0.038	215.2±22.6	269.4±29.2
100	100	0.984 ± 0.015	0.942 ± 0.032	249.9±40.5	293.0±32.6
100	200	0.983 ± 0.013	0.942 ± 0.027	259.9±46.9	290.9±29.6
200	5	0.982 ± 0.016	0.940 ± 0.033	237.5±10.6	94.2±11.2
200	10	0.987 ± 0.013	0.950 ± 0.032	274.8±16.9	180.4±16.9
200	25	0.992 ± 0.008	0.961 ± 0.025	315.3±20.6	272.0±19.3
200	100	0.992 ± 0.009	0.960 ± 0.023	355.1±39.3	294.6±25.5
200	200	0.991 ± 0.009	0.957 ± 0.022	354.2±41.9	295.4±25.8
500	5	0.988 ± 0.012	0.959 ± 0.023	538.4±10.3	94.2±7.4
500	10	0.993 ± 0.008	0.967 ± 0.022	571.8±16.4	182.7±10.1
500	25	0.995 ± 0.007	0.971 ± 0.020	623.5±24.6	269.6±12.6
500	100	0.996 ± 0.005	0.975 ± 0.018	659.2±41.7	293.0±17.8
500	200	0.996 ± 0.006	0.975 ± 0.019	662.2±44.7	293.1±14.2

Table A.26: Musk_1, Bagging root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.744 ± 0.182	0.668 ± 0.148	32.1±4.3	57.5±6.6
10	10	0.710 ± 0.171	0.661 ± 0.158	39.1±5.6	66.2±5.5
10	25	0.724 ± 0.188	0.666 ± 0.167	47.6±8.5	71.7±5.4
10	100	0.697 ± 0.177	0.655 ± 0.160	53.6±16.5	72.3±5.1
10	200	0.697 ± 0.187	0.644 ± 0.160	52.4±14.0	72.1±5.2
25	5	0.759 ± 0.181	0.689 ± 0.154	47.7±4.6	55.5±5.0
25	10	0.777 ± 0.156	0.707 ± 0.141	56.0±6.2	67.1±3.9
25	25	0.768 ± 0.156	0.704 ± 0.136	62.9±9.2	70.8±3.5
25	100	0.755 ± 0.175	0.693 ± 0.166	69.1±17.7	71.1±3.7
25	200	0.761 ± 0.173	0.681 ± 0.138	71.1±19.1	71.8±3.7
100	5	0.859 ± 0.123	0.769 ± 0.116	122.8±4.7	56.0±2.5
100	10	0.850 ± 0.126	0.766 ± 0.125	130.7±5.7	67.3±2.0
100	25	0.834 ± 0.120	0.744 ± 0.130	138.0±9.4	71.3±2.3
100	100	0.835 ± 0.122	0.760 ± 0.127	142.9±15.1	71.9±2.5
100	200	0.851 ± 0.130	0.775 ± 0.126	143.3±15.5	71.3±2.3
200	5	0.896 ± 0.091	0.802 ± 0.112	223.3±4.4	56.2±1.9
200	10	0.887 ± 0.108	0.805 ± 0.117	230.9±5.0	67.0±1.4
200	25	0.883 ± 0.102	0.791 ± 0.106	240.7±10.7	71.1±1.6
200	100	0.877 ± 0.109	0.799 ± 0.117	243.6±16.3	71.5±1.5
200	200	0.898 ± 0.088	0.793 ± 0.113	246.2±16.1	71.9±1.8
500	5	0.919 ± 0.088	0.802 ± 0.113	523.3±5.0	56.4±1.1
500	10	0.916 ± 0.086	0.813 ± 0.113	531.6±4.9	67.0±1.0
500	25	0.899 ± 0.092	0.810 ± 0.115	543.0±10.3	71.1±1.2
500	100	0.912 ± 0.091	0.823 ± 0.107	543.9±15.0	71.7±1.1
500	200	0.920 ± 0.076	0.816 ± 0.115	544.0±16.4	71.5±1.2

Table A.27: Mutagenesis$_{All}$, Bagging root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.714 ± 0.113	0.683 ± 0.089	38.6±8.9	40.5±11.9
10	10	0.747 ± 0.109	0.720 ± 0.095	63.5±13.1	67.6±14.6
10	25	0.767 ± 0.113	0.733 ± 0.094	122.2±25.3	107.3±16.1
10	100	0.781 ± 0.108	0.730 ± 0.090	292.5±56.8	140.7±7.6
10	200	0.781 ± 0.099	0.744 ± 0.087	488.1±78.0	145.6±8.5
25	5	0.738 ± 0.102	0.696 ± 0.093	53.6±8.3	40.7±10.5
25	10	0.763 ± 0.111	0.721 ± 0.093	75.9±11.5	66.1±14.7
25	25	0.782 ± 0.106	0.745 ± 0.088	139.4±24.6	106.4±13.1
25	100	0.771 ± 0.101	0.731 ± 0.089	320.7±51.1	140.8±7.2
25	200	0.787 ± 0.102	0.734 ± 0.095	486.3±81.4	145.7±6.9
100	5	0.787 ± 0.105	0.731 ± 0.081	129.0±7.9	40.7±5.3
100	10	0.799 ± 0.088	0.753 ± 0.084	155.2±13.7	67.7±9.3
100	25	0.798 ± 0.099	0.760 ± 0.088	213.9±25.5	107.2±10.5
100	100	0.795 ± 0.105	0.748 ± 0.102	402.9±46.8	140.2±6.3
100	200	0.800 ± 0.096	0.757 ± 0.089	579.8±89.3	145.4±5.7
200	5	0.796 ± 0.099	0.743 ± 0.085	229.8±8.4	40.2±4.3
200	10	0.811 ± 0.092	0.766 ± 0.082	255.9±12.9	67.5±6.0
200	25	0.813 ± 0.096	0.770 ± 0.082	308.9±22.3	106.9±6.6
200	100	0.805 ± 0.098	0.757 ± 0.087	498.5±56.8	140.1±5.6
200	200	0.814 ± 0.093	0.755 ± 0.086	675.4±78.4	146.1±4.5
500	5	0.807 ± 0.099	0.741 ± 0.079	533.0±8.7	40.0±2.9
500	10	0.814 ± 0.097	0.771 ± 0.088	560.6±14.1	68.1±4.7
500	25	0.819 ± 0.089	0.774 ± 0.085	619.2±27.2	106.2±4.7
500	100	0.823 ± 0.093	0.766 ± 0.086	794.3±44.9	140.5±4.0
500	200	0.821 ± 0.092	0.765 ± 0.087	959.9±80.7	146.1±3.5

Table A.28: Mutagenesis$_{RF}$, Bagging root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.762 ± 0.125	0.733 ± 0.091	38.0±7.8	36.3±8.6
10	10	0.792 ± 0.111	0.765 ± 0.100	63.0±13.9	58.3±11.8
10	25	0.821 ± 0.106	0.782 ± 0.098	115.9±24.2	85.8±9.3
10	100	0.830 ± 0.097	0.801 ± 0.082	269.6±54.9	107.6±7.9
10	200	0.827 ± 0.105	0.796 ± 0.092	400.1±72.8	110.4±7.6
25	5	0.797 ± 0.120	0.748 ± 0.104	53.5±7.8	34.6±7.8
25	10	0.834 ± 0.097	0.789 ± 0.094	78.1±12.5	60.1±10.5
25	25	0.827 ± 0.102	0.791 ± 0.091	128.8±23.4	88.4±9.5
25	100	0.836 ± 0.105	0.805 ± 0.090	290.9±55.9	107.2±6.2
25	200	0.840 ± 0.093	0.794 ± 0.088	439.9±82.7	109.8±6.2
100	5	0.846 ± 0.096	0.784 ± 0.092	128.2±7.5	35.6±5.0
100	10	0.857 ± 0.096	0.804 ± 0.091	154.2±12.1	59.7±6.2
100	25	0.865 ± 0.097	0.817 ± 0.080	209.5±25.0	87.5±7.1
100	100	0.857 ± 0.095	0.814 ± 0.097	357.1±54.2	107.6±4.8
100	200	0.863 ± 0.089	0.816 ± 0.086	503.7±77.7	110.0±5.3
200	5	0.855 ± 0.095	0.780 ± 0.082	230.9±8.1	36.3±3.3
200	10	0.865 ± 0.081	0.812 ± 0.081	252.0±12.3	58.2±5.2
200	25	0.871 ± 0.092	0.822 ± 0.084	309.7±21.5	87.1±5.1
200	100	0.885 ± 0.085	0.832 ± 0.082	459.3±52.5	107.3±4.8
200	200	0.877 ± 0.090	0.816 ± 0.088	604.9±69.6	109.7±4.4
500	5	0.868 ± 0.090	0.787 ± 0.096	533.2±9.0	35.7±2.2
500	10	0.878 ± 0.081	0.827 ± 0.083	558.8±13.9	59.4±3.5
500	25	0.890 ± 0.081	0.840 ± 0.073	615.2±26.3	87.1±3.4
500	100	0.894 ± 0.080	0.830 ± 0.077	780.1±60.0	106.9±3.2
500	200	0.891 ± 0.077	0.835 ± 0.078	930.1±95.6	109.9±2.9

Table A.29: Carcinogenesis, Bagging root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.586 ± 0.088	0.561 ± 0.069	39.1±8.6	66.7±19.4
10	10	0.554 ± 0.102	0.546 ± 0.079	66.7±12.3	121.8±29.6
10	25	0.596 ± 0.099	0.576 ± 0.082	127.0±21.2	202.5±19.5
10	100	0.573 ± 0.096	0.567 ± 0.080	308.8±50.7	257.2±10.9
10	200	0.567 ± 0.096	0.547 ± 0.075	508.9±85.0	266.1±8.2
25	5	0.569 ± 0.098	0.559 ± 0.086	54.0±7.9	64.6±18.5
25	10	0.570 ± 0.098	0.564 ± 0.088	82.0±13.9	119.9±23.5
25	25	0.599 ± 0.092	0.587 ± 0.084	143.1±19.8	197.2±19.8
25	100	0.593 ± 0.094	0.568 ± 0.084	341.3±57.5	256.5±7.5
25	200	0.593 ± 0.091	0.570 ± 0.084	538.6±86.9	266.5±7.1
100	5	0.593 ± 0.086	0.578 ± 0.080	128.7±7.6	65.0±11.3
100	10	0.619 ± 0.103	0.600 ± 0.087	155.7±10.3	123.9±18.3
100	25	0.612 ± 0.085	0.591 ± 0.076	218.1±20.0	200.8±13.4
100	100	0.592 ± 0.096	0.569 ± 0.081	404.7±58.4	257.4±7.1
100	200	0.590 ± 0.086	0.575 ± 0.080	613.7±90.4	266.3±5.5
200	5	0.606 ± 0.094	0.580 ± 0.080	229.4±8.5	67.2±7.5
200	10	0.616 ± 0.086	0.590 ± 0.071	259.8±14.4	122.1±12.3
200	25	0.622 ± 0.074	0.602 ± 0.068	322.1±22.5	199.5±10.7
200	100	0.598 ± 0.097	0.577 ± 0.081	512.7±50.4	255.9±6.1
200	200	0.611 ± 0.086	0.583 ± 0.078	704.9±89.1	266.8±4.5
500	5	0.620 ± 0.083	0.596 ± 0.072	530.9±8.4	66.0±5.4
500	10	0.627 ± 0.078	0.593 ± 0.069	558.9±13.6	121.8±7.2
500	25	0.625 ± 0.078	0.605 ± 0.070	618.4±20.3	199.6±7.7
500	100	0.617 ± 0.081	0.579 ± 0.071	802.7±51.3	256.7±4.1
500	200	0.616 ± 0.085	0.585 ± 0.079	1024.7±106.2	266.7±3.7

Table A.30: $\text{Diterpenes}_{52.3}$, Bagging root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.878 ± 0.058	0.800 ± 0.064	50.3±9.0	119.5±36.1
10	10	0.920 ± 0.043	0.848 ± 0.054	82.1±15.1	234.0±48.9
10	25	0.936 ± 0.034	0.880 ± 0.045	119.3±20.1	330.0±42.7
10	100	0.936 ± 0.035	0.887 ± 0.046	154.2±35.7	349.6±46.2
10	200	0.941 ± 0.029	0.890 ± 0.039	159.7±45.8	357.3±43.8
25	5	0.906 ± 0.051	0.833 ± 0.056	65.5±9.4	128.8±34.6
25	10	0.935 ± 0.035	0.871 ± 0.046	97.7±14.9	233.0±37.2
25	25	0.950 ± 0.028	0.890 ± 0.044	139.8±21.3	334.7±41.4
25	100	0.952 ± 0.030	0.898 ± 0.041	171.7±35.6	349.9±42.0
25	200	0.951 ± 0.028	0.897 ± 0.043	169.8±38.2	353.8±41.7
100	5	0.960 ± 0.026	0.902 ± 0.041	140.9±9.6	121.2±18.8
100	10	0.976 ± 0.017	0.925 ± 0.035	172.1±13.6	231.4±26.9
100	25	0.978 ± 0.018	0.930 ± 0.032	212.1±22.1	333.7±23.8
100	100	0.981 ± 0.013	0.933 ± 0.030	251.2±35.2	349.5±32.3
100	200	0.980 ± 0.016	0.934 ± 0.033	253.1±41.1	352.3±26.4
200	5	0.977 ± 0.017	0.929 ± 0.033	240.1±9.6	123.8±15.2
200	10	0.985 ± 0.012	0.941 ± 0.027	269.9±13.1	235.2±20.0
200	25	0.988 ± 0.010	0.948 ± 0.028	308.4±18.9	331.2±18.5
200	100	0.988 ± 0.010	0.951 ± 0.024	344.9±37.9	352.9±24.0
200	200	0.989 ± 0.009	0.953 ± 0.028	346.4±40.3	350.5±22.1
500	5	0.983 ± 0.014	0.944 ± 0.029	540.8±10.4	123.5±10.9
500	10	0.990 ± 0.009	0.955 ± 0.025	571.8±12.0	233.1±13.0
500	25	0.994 ± 0.006	0.963 ± 0.022	616.3±21.0	332.1±12.7
500	100	0.994 ± 0.006	0.966 ± 0.019	647.2±36.4	353.9±15.2
500	200	0.994 ± 0.006	0.966 ± 0.018	657.6±45.3	353.3±13.6

Table A.31: $\text{Diterpenes}_{52.54}$, Bagging root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.815 ± 0.080	0.750 ± 0.072	54.7±11.0	127.7±40.7
10	10	0.863 ± 0.053	0.790 ± 0.058	88.6±14.3	261.8±46.2
10	25	0.892 ± 0.043	0.825 ± 0.052	139.2±25.2	395.8±40.8
10	100	0.888 ± 0.051	0.831 ± 0.055	183.5±41.0	442.1±44.0
10	200	0.895 ± 0.040	0.837 ± 0.048	192.1±43.6	437.9±46.9
25	5	0.842 ± 0.062	0.759 ± 0.059	70.4±12.2	129.1±35.1
25	10	0.882 ± 0.050	0.804 ± 0.056	103.3±16.6	255.1±39.7
25	25	0.908 ± 0.044	0.838 ± 0.049	158.2±22.1	394.7±40.0
25	100	0.914 ± 0.042	0.848 ± 0.051	203.9±47.0	442.8±43.9
25	200	0.907 ± 0.047	0.841 ± 0.051	215.6±51.7	453.6±43.8
100	5	0.913 ± 0.041	0.831 ± 0.048	143.5±10.1	130.4±21.0
100	10	0.933 ± 0.036	0.859 ± 0.050	180.4±15.3	259.5±31.9
100	25	0.951 ± 0.026	0.882 ± 0.042	231.3±23.4	398.7±24.3
100	100	0.956 ± 0.025	0.891 ± 0.040	279.0±41.4	441.2±33.4
100	200	0.956 ± 0.028	0.891 ± 0.043	284.7±42.3	438.4±31.3
200	5	0.946 ± 0.029	0.869 ± 0.044	243.5±11.8	126.7±15.0
200	10	0.965 ± 0.022	0.902 ± 0.036	279.5±15.5	258.6±22.2
200	25	0.972 ± 0.019	0.918 ± 0.034	329.1±24.8	398.5±20.8
200	100	0.969 ± 0.019	0.911 ± 0.033	379.6±39.9	445.2±24.7
200	200	0.971 ± 0.023	0.918 ± 0.036	388.2±45.9	441.2±25.4
500	5	0.964 ± 0.021	0.892 ± 0.039	543.3±12.2	128.5±11.0
500	10	0.980 ± 0.013	0.931 ± 0.029	578.2±14.3	259.1±16.3
500	25	0.985 ± 0.012	0.941 ± 0.028	631.0±25.6	397.8±13.8
500	100	0.984 ± 0.011	0.939 ± 0.027	679.8±37.6	443.0±16.2
500	200	0.985 ± 0.011	0.941 ± 0.025	684.9±42.2	440.6±17.9

Table A.32: $\text{Diterpenes}_{54.3}$, Bagging root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.885 ± 0.065	0.817 ± 0.067	50.4±10.6	102.4±34.5
10	10	0.935 ± 0.044	0.874 ± 0.058	81.8±13.7	194.7±34.8
10	25	0.949 ± 0.036	0.901 ± 0.047	121.4±20.4	264.4±40.9
10	100	0.949 ± 0.029	0.900 ± 0.042	139.1±32.7	288.8±48.8
10	200	0.948 ± 0.032	0.901 ± 0.045	150.2±38.6	289.1±52.7
25	5	0.937 ± 0.040	0.874 ± 0.054	65.6±10.6	113.0±28.0
25	10	0.954 ± 0.032	0.891 ± 0.052	97.7±15.8	191.7±31.5
25	25	0.964 ± 0.026	0.914 ± 0.041	131.8±22.8	269.8±38.3
25	100	0.968 ± 0.026	0.917 ± 0.042	158.4±31.2	290.4±38.4
25	200	0.967 ± 0.026	0.918 ± 0.036	165.1±45.2	286.9±43.2
100	5	0.974 ± 0.022	0.927 ± 0.034	139.4±9.9	108.1±17.1
100	10	0.982 ± 0.017	0.940 ± 0.034	173.3±16.2	193.2±22.8
100	25	0.986 ± 0.013	0.947 ± 0.030	207.5±20.2	274.6±24.8
100	100	0.988 ± 0.012	0.948 ± 0.033	236.0±31.9	286.9±28.3
100	200	0.989 ± 0.011	0.952 ± 0.023	240.3±37.4	292.2±30.7
200	5	0.985 ± 0.013	0.947 ± 0.029	239.2±10.7	107.5±13.0
200	10	0.990 ± 0.010	0.959 ± 0.025	270.0±13.2	194.3±17.3
200	25	0.993 ± 0.009	0.965 ± 0.024	312.4±20.1	271.5±20.2
200	100	0.993 ± 0.009	0.968 ± 0.023	345.9±35.3	281.9±20.5
200	200	0.993 ± 0.008	0.964 ± 0.025	336.6±34.9	286.8±21.9
500	5	0.991 ± 0.011	0.963 ± 0.022	540.1±9.2	106.9±8.0
500	10	0.994 ± 0.008	0.971 ± 0.019	574.6±15.1	195.6±10.3
500	25	0.996 ± 0.006	0.976 ± 0.019	613.3±24.5	270.8±12.7
500	100	0.996 ± 0.005	0.977 ± 0.019	638.0±31.9	288.0±14.0
500	200	0.996 ± 0.005	0.977 ± 0.017	645.6±40.1	287.1±15.0

Table A.33: Musk_1, Bagging root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.727 ± 0.199	0.671 ± 0.156	31.9±4.0	62.8±5.7
10	10	0.745 ± 0.193	0.669 ± 0.169	37.6±5.7	67.9±4.8
10	25	0.728 ± 0.162	0.673 ± 0.154	41.9±8.0	71.9±4.9
10	100	0.751 ± 0.157	0.691 ± 0.144	43.4±10.7	71.7±5.0
10	200	0.725 ± 0.149	0.671 ± 0.138	43.3±9.9	72.3±4.3
25	5	0.811 ± 0.148	0.723 ± 0.129	46.3±3.6	62.1±4.7
25	10	0.791 ± 0.148	0.716 ± 0.150	52.7±4.7	69.2±3.4
25	25	0.767 ± 0.154	0.696 ± 0.136	58.4±8.8	70.9±3.6
25	100	0.785 ± 0.150	0.714 ± 0.156	61.1±13.1	71.4±3.6
25	200	0.792 ± 0.151	0.731 ± 0.145	59.6±13.0	72.1±3.7
100	5	0.876 ± 0.118	0.782 ± 0.128	122.2±4.4	61.7±2.0
100	10	0.860 ± 0.125	0.773 ± 0.123	128.3±4.7	69.0±2.0
100	25	0.890 ± 0.107	0.780 ± 0.120	134.6±9.3	71.1±1.7
100	100	0.871 ± 0.118	0.783 ± 0.126	136.2±12.0	71.4±2.2
100	200	0.872 ± 0.105	0.769 ± 0.125	134.8±9.3	71.8±1.9
200	5	0.892 ± 0.103	0.793 ± 0.116	223.3±4.4	61.4±1.5
200	10	0.890 ± 0.108	0.809 ± 0.118	228.5±5.5	69.1±1.5
200	25	0.880 ± 0.109	0.792 ± 0.122	232.6±9.9	71.4±1.5
200	100	0.896 ± 0.099	0.821 ± 0.109	235.9±11.2	71.6±1.6
200	200	0.894 ± 0.108	0.801 ± 0.119	234.4±10.8	71.5±1.5
500	5	0.918 ± 0.086	0.816 ± 0.106	523.0±4.3	61.6±1.2
500	10	0.912 ± 0.089	0.812 ± 0.110	528.1±6.2	69.2±1.0
500	25	0.920 ± 0.081	0.815 ± 0.105	534.9±9.3	71.4±1.1
500	100	0.923 ± 0.077	0.813 ± 0.110	534.3±10.1	71.7±1.1
500	200	0.922 ± 0.083	0.813 ± 0.114	536.8±11.1	71.6±1.1

Table A.34: Mutagenesis_{All}, Bagging root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.711 ± 0.117	0.694 ± 0.082	39.4±8.7	40.3±10.2
10	10	0.743 ± 0.118	0.716 ± 0.098	63.9±11.9	68.5±12.9
10	25	0.772 ± 0.103	0.730 ± 0.095	121.4±25.6	105.2±13.6
10	100	0.770 ± 0.100	0.724 ± 0.095	308.0±54.7	142.1±7.8
10	200	0.765 ± 0.111	0.721 ± 0.096	478.2±85.5	145.8±6.8
25	5	0.749 ± 0.111	0.705 ± 0.101	53.3±7.4	41.1±10.0
25	10	0.770 ± 0.103	0.733 ± 0.096	81.0±12.7	67.8±15.1
25	25	0.780 ± 0.097	0.741 ± 0.085	136.5±23.0	109.1±11.5
25	100	0.778 ± 0.101	0.738 ± 0.094	332.5±62.0	140.6±7.3
25	200	0.778 ± 0.098	0.738 ± 0.087	495.7±73.7	146.0±6.4
100	5	0.801 ± 0.100	0.741 ± 0.086	129.0±8.5	41.2±5.8
100	10	0.791 ± 0.101	0.749 ± 0.089	153.7±12.7	69.8±8.2
100	25	0.813 ± 0.092	0.760 ± 0.081	213.1±24.9	109.4±9.3
100	100	0.806 ± 0.099	0.759 ± 0.092	392.8±50.7	140.6±5.4
100	200	0.798 ± 0.095	0.751 ± 0.084	564.9±69.3	145.3±5.8
200	5	0.797 ± 0.098	0.744 ± 0.088	230.7±8.7	40.5±4.5
200	10	0.809 ± 0.099	0.761 ± 0.085	255.7±13.4	69.5±6.9
200	25	0.809 ± 0.091	0.773 ± 0.085	312.8±22.2	108.3±6.5
200	100	0.816 ± 0.100	0.767 ± 0.098	497.4±56.0	141.0±5.0
200	200	0.813 ± 0.097	0.760 ± 0.092	662.7±85.4	145.7±4.7
500	5	0.805 ± 0.094	0.741 ± 0.079	534.4±8.4	39.9±2.8
500	10	0.816 ± 0.091	0.774 ± 0.078	559.5±12.9	69.0±4.1
500	25	0.822 ± 0.094	0.780 ± 0.087	618.3±21.1	108.1±4.8
500	100	0.824 ± 0.089	0.774 ± 0.079	791.5±63.1	141.3±3.6
500	200	0.820 ± 0.095	0.768 ± 0.081	969.9±93.1	146.6±3.2

Table A.35: Mutagenesis$_{RF}$, Bagging root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.768 ± 0.122	0.727 ± 0.087	37.3±7.8	35.4±10.4
10	10	0.807 ± 0.105	0.771 ± 0.094	62.2±13.1	61.2±12.0
10	25	0.824 ± 0.110	0.793 ± 0.079	114.2±27.4	87.5±11.0
10	100	0.824 ± 0.109	0.780 ± 0.102	267.4±48.6	107.4±7.8
10	200	0.829 ± 0.104	0.795 ± 0.096	403.9±78.5	110.2±6.5
25	5	0.807 ± 0.109	0.751 ± 0.089	53.0±7.8	37.0±8.7
25	10	0.815 ± 0.100	0.764 ± 0.090	79.0±11.9	59.6±10.7
25	25	0.833 ± 0.112	0.798 ± 0.094	129.9±23.1	86.2±9.2
25	100	0.841 ± 0.103	0.799 ± 0.090	287.1±51.9	106.9±6.1
25	200	0.851 ± 0.101	0.805 ± 0.088	425.7±81.3	109.9±6.2
100	5	0.851 ± 0.089	0.778 ± 0.085	128.2±8.7	36.6±5.6
100	10	0.859 ± 0.094	0.803 ± 0.098	154.3±14.0	59.6±7.0
100	25	0.871 ± 0.087	0.829 ± 0.079	205.0±22.1	88.8±6.7
100	100	0.865 ± 0.093	0.821 ± 0.090	362.7±56.1	107.4±5.2
100	200	0.865 ± 0.098	0.814 ± 0.101	494.9±72.9	109.4±4.6
200	5	0.857 ± 0.083	0.777 ± 0.083	230.3±8.6	36.4±3.9
200	10	0.871 ± 0.088	0.821 ± 0.087	253.6±11.0	60.2±4.7
200	25	0.885 ± 0.081	0.835 ± 0.085	308.9±23.9	87.4±4.3
200	100	0.882 ± 0.082	0.822 ± 0.095	461.3±52.7	107.3±4.3
200	200	0.884 ± 0.071	0.836 ± 0.070	611.4±78.3	109.9±4.1
500	5	0.863 ± 0.088	0.791 ± 0.085	536.7±9.3	36.4±2.6
500	10	0.882 ± 0.084	0.832 ± 0.081	559.9±13.3	59.7±3.3
500	25	0.891 ± 0.079	0.836 ± 0.080	609.1±24.0	87.9±3.4
500	100	0.890 ± 0.077	0.843 ± 0.075	756.7±51.1	107.5±3.4
500	200	0.888 ± 0.079	0.830 ± 0.077	912.2±74.6	110.3±3.1

Table A.36: Carcinogenesis, Bagging root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.567 ± 0.097	0.554 ± 0.088	38.2±8.4	61.0±21.8
10	10	0.577 ± 0.107	0.577 ± 0.090	68.0±12.1	121.5±27.1
10	25	0.603 ± 0.092	0.581 ± 0.073	127.9±22.2	198.0±18.5
10	100	0.589 ± 0.084	0.578 ± 0.079	316.4±56.2	256.2±9.4
10	200	0.584 ± 0.094	0.566 ± 0.077	533.0±99.6	266.0±8.1
25	5	0.582 ± 0.096	0.574 ± 0.076	53.2±7.1	66.6±21.4
25	10	0.594 ± 0.106	0.571 ± 0.089	81.4±10.1	119.7±26.9
25	25	0.585 ± 0.109	0.567 ± 0.085	141.9±20.6	197.6±19.5
25	100	0.591 ± 0.087	0.578 ± 0.086	338.8±56.9	255.5±8.2
25	200	0.585 ± 0.087	0.563 ± 0.085	541.4±76.9	265.8±7.1
100	5	0.597 ± 0.079	0.580 ± 0.063	129.6±8.5	65.5±11.9
100	10	0.612 ± 0.093	0.592 ± 0.079	159.0±12.9	119.7±16.5
100	25	0.611 ± 0.088	0.596 ± 0.075	223.4±22.9	196.0±14.9
100	100	0.591 ± 0.096	0.580 ± 0.083	415.5±53.6	255.6±7.0
100	200	0.600 ± 0.091	0.574 ± 0.086	614.2±76.5	265.3±5.9
200	5	0.609 ± 0.091	0.588 ± 0.074	229.2±8.3	65.6±7.5
200	10	0.618 ± 0.087	0.592 ± 0.073	258.4±13.5	120.3±12.0
200	25	0.617 ± 0.087	0.600 ± 0.076	321.0±20.9	199.3±9.4
200	100	0.603 ± 0.088	0.580 ± 0.078	510.1±57.2	255.9±5.7
200	200	0.602 ± 0.091	0.571 ± 0.079	731.2±95.1	266.0±4.6
500	5	0.612 ± 0.091	0.587 ± 0.078	530.4±8.1	65.6±5.4
500	10	0.623 ± 0.088	0.596 ± 0.070	559.4±14.0	122.8±7.3
500	25	0.627 ± 0.081	0.602 ± 0.073	620.4±21.4	198.1±7.3
500	100	0.615 ± 0.094	0.588 ± 0.079	815.0±51.9	255.6±4.1
500	200	0.612 ± 0.080	0.579 ± 0.072	1011.2±97.1	266.1±3.0

Table A.37: Diterpenes$_{52.3}$, Bagging root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.866 ± 0.063	0.786 ± 0.075	50.5±9.9	118.8±40.9
10	10	0.920 ± 0.038	0.851 ± 0.047	79.6±13.5	231.5±43.9
10	25	0.936 ± 0.037	0.881 ± 0.047	116.6±18.9	320.5±42.9
10	100	0.939 ± 0.032	0.894 ± 0.042	151.4±35.7	342.2±40.2
10	200	0.934 ± 0.035	0.885 ± 0.044	158.2±48.3	344.8±43.5
25	5	0.902 ± 0.053	0.825 ± 0.068	65.0±10.8	115.0±31.4
25	10	0.941 ± 0.036	0.875 ± 0.051	94.7±15.2	230.2±41.5
25	25	0.948 ± 0.028	0.890 ± 0.040	135.7±22.4	329.4±34.3
25	100	0.954 ± 0.025	0.897 ± 0.038	165.6±36.2	344.3±42.5
25	200	0.956 ± 0.026	0.898 ± 0.038	172.9±45.1	345.5±40.2
100	5	0.964 ± 0.024	0.907 ± 0.041	139.2±11.1	124.4±21.6
100	10	0.975 ± 0.018	0.926 ± 0.035	169.5±12.5	231.5±28.3
100	25	0.980 ± 0.015	0.931 ± 0.032	208.1±22.9	324.9±29.1
100	100	0.979 ± 0.015	0.930 ± 0.032	249.2±35.1	346.7±27.6
100	200	0.982 ± 0.012	0.936 ± 0.031	244.6±42.6	346.4±29.6
200	5	0.975 ± 0.019	0.925 ± 0.034	238.9±10.1	121.5±14.1
200	10	0.985 ± 0.013	0.942 ± 0.031	272.2±13.0	230.8±17.9
200	25	0.987 ± 0.010	0.947 ± 0.024	310.6±22.7	323.7±18.7
200	100	0.990 ± 0.009	0.953 ± 0.025	348.1±34.3	349.3±21.5
200	200	0.987 ± 0.011	0.950 ± 0.027	346.9±38.5	348.0±20.5
500	5	0.983 ± 0.014	0.946 ± 0.027	539.0±9.1	120.4±9.8
500	10	0.991 ± 0.008	0.957 ± 0.023	571.5±13.7	230.2±12.1
500	25	0.994 ± 0.007	0.965 ± 0.021	616.0±19.1	326.5±12.7
500	100	0.994 ± 0.007	0.965 ± 0.022	647.4±38.2	346.4±12.2
500	200	0.994 ± 0.007	0.966 ± 0.021	648.5±39.4	345.3±13.9

Table A.38: Diterpenes$_{52.54}$, Bagging root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.811 ± 0.071	0.746 ± 0.065	52.8±12.2	124.4±44.1
10	10	0.864 ± 0.057	0.793 ± 0.064	87.3±17.0	258.7±51.2
10	25	0.894 ± 0.044	0.834 ± 0.048	134.0±22.0	391.2±40.0
10	100	0.894 ± 0.039	0.832 ± 0.046	189.6±41.7	437.8±47.1
10	200	0.895 ± 0.045	0.837 ± 0.044	196.5±50.0	442.0±45.4
25	5	0.840 ± 0.063	0.760 ± 0.062	68.4±11.9	126.7±34.1
25	10	0.881 ± 0.059	0.806 ± 0.064	104.4±15.7	254.9±44.5
25	25	0.907 ± 0.040	0.836 ± 0.048	151.3±23.5	391.1±36.2
25	100	0.913 ± 0.041	0.846 ± 0.048	211.6±45.8	436.4±43.3
25	200	0.911 ± 0.043	0.845 ± 0.054	213.2±53.3	440.4±45.2
100	5	0.907 ± 0.045	0.826 ± 0.054	142.5±10.5	123.4±21.1
100	10	0.941 ± 0.030	0.868 ± 0.043	179.1±15.1	257.4±30.9
100	25	0.948 ± 0.029	0.879 ± 0.044	227.3±23.0	393.9±26.5
100	100	0.955 ± 0.025	0.887 ± 0.042	275.3±37.4	433.0±29.1
100	200	0.952 ± 0.026	0.883 ± 0.042	290.5±52.0	437.6±31.7
200	5	0.942 ± 0.029	0.868 ± 0.046	243.2±11.9	125.3±16.1
200	10	0.962 ± 0.025	0.899 ± 0.040	277.0±15.3	257.3±23.7
200	25	0.973 ± 0.017	0.917 ± 0.032	323.9±22.5	393.6±20.0
200	100	0.971 ± 0.019	0.913 ± 0.037	381.0±50.2	435.4±22.5
200	200	0.974 ± 0.017	0.916 ± 0.035	386.4±52.9	432.1±24.9
500	5	0.963 ± 0.024	0.895 ± 0.037	543.3±11.3	125.7±12.1
500	10	0.978 ± 0.015	0.927 ± 0.031	579.4±15.4	256.4±16.1
500	25	0.984 ± 0.013	0.938 ± 0.030	627.5±23.9	395.4±14.5
500	100	0.985 ± 0.011	0.938 ± 0.028	679.2±46.3	433.2±18.9
500	200	0.985 ± 0.011	0.940 ± 0.026	682.9±48.1	433.2±12.5

Table A.39: $\text{Diterpenes}_{54.3}$, Bagging root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.896 ± 0.054	0.821 ± 0.061	50.2±11.3	109.1±33.0
10	10	0.936 ± 0.040	0.878 ± 0.052	83.3±16.1	200.2±33.9
10	25	0.947 ± 0.033	0.901 ± 0.045	117.4±23.7	268.4±42.5
10	100	0.955 ± 0.030	0.912 ± 0.042	146.1±27.7	269.7±51.5
10	200	0.950 ± 0.031	0.905 ± 0.043	150.5±42.0	273.7±51.0
25	5	0.920 ± 0.048	0.848 ± 0.067	66.1±10.8	110.0±30.0
25	10	0.956 ± 0.030	0.899 ± 0.046	91.1±13.3	189.1±34.2
25	25	0.966 ± 0.028	0.914 ± 0.042	131.1±22.7	254.4±37.6
25	100	0.970 ± 0.026	0.922 ± 0.039	162.2±37.1	270.4±44.2
25	200	0.965 ± 0.025	0.919 ± 0.035	161.9±38.7	281.6±42.5
100	5	0.979 ± 0.018	0.935 ± 0.038	138.1±10.2	105.3±15.6
100	10	0.981 ± 0.018	0.937 ± 0.036	170.2±12.4	190.7±22.3
100	25	0.984 ± 0.014	0.942 ± 0.034	208.6±22.6	269.2±25.4
100	100	0.986 ± 0.011	0.948 ± 0.027	234.5±37.1	282.6±33.2
100	200	0.989 ± 0.010	0.953 ± 0.024	235.5±41.3	281.6±28.2
200	5	0.986 ± 0.013	0.951 ± 0.028	240.1±8.9	106.9±13.0
200	10	0.990 ± 0.013	0.959 ± 0.025	272.3±14.3	191.5±17.7
200	25	0.993 ± 0.009	0.966 ± 0.023	308.1±22.7	263.4±19.0
200	100	0.993 ± 0.009	0.963 ± 0.025	334.7±35.0	279.4±22.4
200	200	0.993 ± 0.009	0.964 ± 0.023	338.1±39.5	279.6±26.0
500	5	0.991 ± 0.010	0.962 ± 0.024	540.3±9.7	105.3±7.3
500	10	0.994 ± 0.008	0.973 ± 0.019	573.7±15.8	189.9±10.9
500	25	0.996 ± 0.005	0.979 ± 0.016	602.7±19.3	263.8±13.2
500	100	0.996 ± 0.006	0.980 ± 0.015	635.3±36.6	278.9±12.1
500	200	0.996 ± 0.005	0.974 ± 0.018	639.5±40.4	278.4±13.6

Table A.40: Musk_1, Bagging root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.738 ± 0.177	0.677 ± 0.152	31.2±4.1	58.9±6.3
10	10	0.754 ± 0.163	0.689 ± 0.140	38.0±5.4	65.2±5.5
10	25	0.746 ± 0.161	0.678 ± 0.158	41.4±7.8	67.4±5.3
10	100	0.732 ± 0.171	0.680 ± 0.152	45.2±10.9	66.9±5.1
10	200	0.740 ± 0.159	0.682 ± 0.147	44.5±11.7	68.3±4.8
25	5	0.785 ± 0.168	0.731 ± 0.153	46.9±4.0	58.7±3.6
25	10	0.810 ± 0.147	0.737 ± 0.134	52.6±5.8	64.9±3.4
25	25	0.787 ± 0.135	0.719 ± 0.137	58.4±9.3	67.2±3.8
25	100	0.771 ± 0.158	0.704 ± 0.156	58.0±11.5	67.5±3.9
25	200	0.787 ± 0.153	0.706 ± 0.132	59.5±11.4	67.7±3.7
100	5	0.879 ± 0.116	0.789 ± 0.128	122.0±4.2	58.9±2.0
100	10	0.887 ± 0.101	0.784 ± 0.111	128.1±6.2	65.5±2.0
100	25	0.871 ± 0.124	0.785 ± 0.127	133.2±8.1	67.4±2.5
100	100	0.872 ± 0.112	0.783 ± 0.115	135.2±12.0	67.3±2.3
100	200	0.862 ± 0.128	0.776 ± 0.134	136.8±13.2	67.9±2.0
200	5	0.901 ± 0.103	0.802 ± 0.108	221.9±3.9	58.4±1.6
200	10	0.897 ± 0.086	0.800 ± 0.116	227.6±5.3	65.3±1.3
200	25	0.900 ± 0.107	0.797 ± 0.127	234.2±9.1	67.3±1.5
200	100	0.887 ± 0.103	0.793 ± 0.121	234.7±11.1	67.4±1.6
200	200	0.906 ± 0.096	0.807 ± 0.126	235.1±12.7	67.7±1.6
500	5	0.926 ± 0.085	0.827 ± 0.111	523.3±4.4	58.4±1.2
500	10	0.925 ± 0.084	0.825 ± 0.113	528.2±5.6	65.3±1.0
500	25	0.926 ± 0.083	0.826 ± 0.109	534.1±8.8	67.3±1.2
500	100	0.922 ± 0.072	0.811 ± 0.098	536.6±11.4	67.5±1.3
500	200	0.917 ± 0.084	0.823 ± 0.098	534.4±10.2	67.6±1.0

Table A.41: Mutagenesis$_{All}$, Bagging root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.727 ± 0.111	0.692 ± 0.089	37.9±8.1	41.2±11.6
10	10	0.739 ± 0.115	0.715 ± 0.098	66.5±14.6	70.2±12.7
10	25	0.762 ± 0.095	0.733 ± 0.088	122.7±25.8	106.1±12.4
10	100	0.780 ± 0.100	0.735 ± 0.096	306.9±52.0	140.6±8.4
10	200	0.778 ± 0.098	0.739 ± 0.090	476.0±84.8	145.0±8.3
25	5	0.750 ± 0.109	0.703 ± 0.092	52.9±7.5	40.4±9.8
25	10	0.765 ± 0.099	0.721 ± 0.089	80.2±12.8	67.7±13.2
25	25	0.786 ± 0.094	0.737 ± 0.086	137.5±27.0	108.4±12.1
25	100	0.790 ± 0.106	0.747 ± 0.094	311.1±54.4	141.4±7.6
25	200	0.783 ± 0.105	0.737 ± 0.087	487.6±75.7	147.3±7.2
100	5	0.790 ± 0.098	0.731 ± 0.088	128.5±7.7	40.4±5.5
100	10	0.799 ± 0.099	0.754 ± 0.090	157.0±13.5	69.5±9.0
100	25	0.802 ± 0.094	0.758 ± 0.091	212.3±24.7	109.1±8.9
100	100	0.799 ± 0.097	0.749 ± 0.084	400.9±58.2	140.4±6.4
100	200	0.800 ± 0.099	0.753 ± 0.084	573.9±87.2	145.9±5.5
200	5	0.800 ± 0.098	0.743 ± 0.080	230.8±8.4	40.3±3.8
200	10	0.810 ± 0.091	0.765 ± 0.081	256.8±13.3	68.6±5.7
200	25	0.814 ± 0.091	0.776 ± 0.085	317.6±26.2	107.6±6.5
200	100	0.811 ± 0.088	0.756 ± 0.079	497.3±49.9	140.1±4.8
200	200	0.813 ± 0.092	0.761 ± 0.082	676.6±72.2	145.6±5.2
500	5	0.804 ± 0.100	0.742 ± 0.081	533.6±7.6	40.4±3.1
500	10	0.812 ± 0.094	0.773 ± 0.084	555.8±13.6	69.0±4.5
500	25	0.813 ± 0.089	0.776 ± 0.085	620.2±24.8	107.8±4.7
500	100	0.825 ± 0.088	0.774 ± 0.085	799.0±53.2	140.5±3.6
500	200	0.822 ± 0.089	0.770 ± 0.081	976.1±100.8	145.6±3.8

Table A.42: Mutagenesis$_{RF}$, Bagging root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.779 ± 0.105	0.735 ± 0.087	37.4±7.6	36.6±10.2
10	10	0.805 ± 0.113	0.766 ± 0.087	63.9±11.8	60.8±10.5
10	25	0.813 ± 0.106	0.783 ± 0.087	118.1±25.2	86.9±10.4
10	100	0.817 ± 0.108	0.790 ± 0.099	272.5±54.2	107.7±7.1
10	200	0.822 ± 0.099	0.776 ± 0.091	403.4±76.5	110.7±8.1
25	5	0.798 ± 0.126	0.751 ± 0.089	54.0±9.9	36.7±8.5
25	10	0.824 ± 0.103	0.772 ± 0.084	77.4±13.1	58.7±10.8
25	25	0.839 ± 0.094	0.804 ± 0.089	126.0±23.3	87.3±9.4
25	100	0.844 ± 0.096	0.807 ± 0.083	286.2±53.6	107.1±7.4
25	200	0.849 ± 0.089	0.811 ± 0.084	424.4±76.8	109.2±6.3
100	5	0.848 ± 0.105	0.773 ± 0.087	131.4±8.3	36.0±4.0
100	10	0.860 ± 0.084	0.803 ± 0.085	154.4±12.8	59.7±6.5
100	25	0.872 ± 0.088	0.817 ± 0.079	204.4±24.4	87.9±6.9
100	100	0.874 ± 0.080	0.817 ± 0.082	356.0±50.7	106.7±4.9
100	200	0.869 ± 0.087	0.817 ± 0.069	529.9±81.2	109.0±5.3
200	5	0.864 ± 0.086	0.787 ± 0.095	229.5±8.2	36.4±3.8
200	10	0.872 ± 0.086	0.824 ± 0.090	256.2±13.7	60.3±5.7
200	25	0.883 ± 0.086	0.838 ± 0.082	312.7±24.0	87.5±5.2
200	100	0.889 ± 0.077	0.830 ± 0.086	465.0±52.9	106.9±4.2
200	200	0.882 ± 0.081	0.824 ± 0.074	617.4±88.5	109.5±3.5
500	5	0.864 ± 0.084	0.803 ± 0.086	534.3±8.7	36.3±2.5
500	10	0.878 ± 0.084	0.828 ± 0.078	557.4±11.7	60.0±3.1
500	25	0.891 ± 0.078	0.834 ± 0.079	611.4±22.8	88.3±3.3
500	100	0.891 ± 0.079	0.841 ± 0.081	758.6±54.2	107.4±2.9
500	200	0.890 ± 0.080	0.837 ± 0.080	916.1±88.6	109.5±2.8

Table A.43: Carcinogenesis, Unique root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.563 ± 0.093	0.553 ± 0.082	40.6±7.8	83.0±32.4
10	10	0.576 ± 0.101	0.566 ± 0.084	68.8±12.2	160.7±42.2
10	25	0.574 ± 0.088	0.568 ± 0.082	134.0±25.1	277.5±29.5
10	100	0.570 ± 0.100	0.565 ± 0.093	325.8±55.1	383.1±19.9
10	200	0.557 ± 0.085	0.548 ± 0.073	546.6±93.5	405.1±13.8
25	5	0.575 ± 0.109	0.571 ± 0.077	58.5±9.1	82.6±23.1
25	10	0.580 ± 0.094	0.577 ± 0.080	92.9±14.2	158.4±35.3
25	25	0.593 ± 0.096	0.582 ± 0.083	157.6±24.1	280.6±28.7
25	100	0.577 ± 0.089	0.567 ± 0.079	355.8±56.6	384.9±14.2
25	200	0.587 ± 0.097	0.569 ± 0.088	582.7±98.7	406.3±13.6
100	5	0.600 ± 0.088	0.590 ± 0.077	133.1±8.1	81.5±12.0
100	10	0.598 ± 0.099	0.587 ± 0.087	168.7±15.4	158.4±24.5
100	25	0.597 ± 0.079	0.582 ± 0.075	250.0±22.3	276.0±22.2
100	100	0.607 ± 0.084	0.577 ± 0.076	526.0±59.0	384.6±12.0
100	200	0.605 ± 0.100	0.578 ± 0.085	785.2±110.4	404.4±8.9
200	5	0.599 ± 0.090	0.588 ± 0.072	233.9±8.6	81.1±10.9
200	10	0.602 ± 0.083	0.584 ± 0.069	268.4±13.0	154.7±19.7
200	25	0.607 ± 0.091	0.588 ± 0.080	351.2±24.9	281.0±18.9
200	100	0.603 ± 0.084	0.573 ± 0.074	626.6±52.7	384.5±10.8
200	200	0.602 ± 0.085	0.572 ± 0.072	942.6±97.0	404.9±7.2
500	5	0.610 ± 0.090	0.592 ± 0.075	534.5±9.5	80.7±5.7
500	10	0.623 ± 0.090	0.592 ± 0.077	568.9±13.6	160.8±10.5
500	25	0.622 ± 0.080	0.598 ± 0.067	648.1±21.5	279.0±11.6
500	100	0.613 ± 0.084	0.572 ± 0.070	943.8±59.9	382.7±8.1
500	200	0.609 ± 0.087	0.588 ± 0.075	1259.2±95.3	405.5±7.2

Table A.44: $\text{Diterpenes}_{52.3}$, Unique root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.853 ± 0.071	0.779 ± 0.074	50.4±10.9	146.2±52.3
10	10	0.900 ± 0.046	0.845 ± 0.054	82.4±14.3	297.3±63.3
10	25	0.909 ± 0.042	0.874 ± 0.041	133.9±24.0	466.3±69.1
10	100	0.902 ± 0.043	0.878 ± 0.039	181.1±44.0	508.0±69.9
10	200	0.900 ± 0.043	0.877 ± 0.045	182.1±45.0	505.9±69.4
25	5	0.901 ± 0.051	0.822 ± 0.058	66.5±11.8	138.7±33.0
25	10	0.926 ± 0.036	0.859 ± 0.047	101.0±16.3	291.1±62.8
25	25	0.935 ± 0.035	0.885 ± 0.041	150.0±19.4	459.9±54.4
25	100	0.929 ± 0.035	0.888 ± 0.042	194.0±46.0	516.6±65.5
25	200	0.936 ± 0.034	0.890 ± 0.045	194.8±46.6	513.5±65.3
100	5	0.955 ± 0.027	0.889 ± 0.045	144.3±10.7	144.7±27.4
100	10	0.967 ± 0.020	0.911 ± 0.038	183.6±14.9	297.2±37.5
100	25	0.974 ± 0.019	0.919 ± 0.037	236.4±24.5	466.6±38.2
100	100	0.974 ± 0.020	0.924 ± 0.034	278.7±44.5	507.9±41.6
100	200	0.977 ± 0.019	0.928 ± 0.035	287.9±49.9	507.0±43.4
200	5	0.972 ± 0.022	0.920 ± 0.040	244.1±11.4	145.8±17.8
200	10	0.981 ± 0.014	0.938 ± 0.030	283.8±14.2	303.3±26.4
200	25	0.986 ± 0.011	0.947 ± 0.022	342.4±21.9	460.9±29.4
200	100	0.987 ± 0.012	0.948 ± 0.030	411.0±40.8	514.8±38.5
200	200	0.989 ± 0.010	0.953 ± 0.023	424.5±44.3	509.0±32.9
500	5	0.982 ± 0.013	0.938 ± 0.029	544.8±9.5	145.0±12.0
500	10	0.990 ± 0.010	0.954 ± 0.026	584.6±15.7	299.4±19.0
500	25	0.993 ± 0.007	0.963 ± 0.022	649.0±27.9	462.6±21.8
500	100	0.994 ± 0.006	0.963 ± 0.021	774.3±43.5	509.7±23.1
500	200	0.995 ± 0.005	0.968 ± 0.020	870.2±46.5	511.1±20.3

Table A.45: Diterpenes$_{52.54}$, Unique root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.787 ± 0.082	0.723 ± 0.077	52.1±12.5	143.4±51.3
10	10	0.843 ± 0.061	0.775 ± 0.064	91.9±15.7	333.4±68.1
10	25	0.852 ± 0.052	0.814 ± 0.054	147.9±24.7	561.5±64.5
10	100	0.856 ± 0.048	0.831 ± 0.048	213.7±49.9	650.6±77.6
10	200	0.859 ± 0.047	0.837 ± 0.050	219.5±55.1	631.7±76.6
25	5	0.823 ± 0.068	0.748 ± 0.068	68.3±12.8	154.8±46.4
25	10	0.863 ± 0.058	0.787 ± 0.062	110.9±17.9	326.2±60.8
25	25	0.885 ± 0.047	0.828 ± 0.051	159.0±23.6	556.0±60.5
25	100	0.886 ± 0.049	0.838 ± 0.052	231.7±42.4	646.4±63.7
25	200	0.884 ± 0.050	0.836 ± 0.051	242.4±57.0	647.0±71.5
100	5	0.909 ± 0.051	0.830 ± 0.058	147.3±11.5	151.0±30.3
100	10	0.931 ± 0.035	0.860 ± 0.045	189.5±16.6	329.5±38.3
100	25	0.940 ± 0.029	0.871 ± 0.045	260.9±29.1	553.6±43.5
100	100	0.944 ± 0.032	0.873 ± 0.047	315.9±39.3	645.7±52.3
100	200	0.948 ± 0.029	0.883 ± 0.038	329.8±59.6	640.5±51.3
200	5	0.933 ± 0.037	0.852 ± 0.050	247.6±12.2	154.3±18.8
200	10	0.952 ± 0.026	0.888 ± 0.041	290.9±17.0	328.5±33.6
200	25	0.965 ± 0.020	0.906 ± 0.038	366.8±28.0	555.1±30.2
200	100	0.969 ± 0.017	0.913 ± 0.032	459.5±51.9	639.5±34.1
200	200	0.970 ± 0.018	0.909 ± 0.031	479.9±59.3	642.5±42.3
500	5	0.960 ± 0.025	0.889 ± 0.041	549.1±12.4	152.3±13.1
500	10	0.976 ± 0.016	0.924 ± 0.035	589.7±13.8	331.4±20.5
500	25	0.983 ± 0.013	0.937 ± 0.027	659.9±25.2	558.2±26.0
500	100	0.983 ± 0.011	0.937 ± 0.028	806.8±49.2	641.2±30.0
500	200	0.987 ± 0.009	0.945 ± 0.025	909.6±55.9	638.3±22.3

Table A.46: Diterpenes$_{54.3}$, Unique root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.875 ± 0.058	0.808 ± 0.064	51.0±11.5	129.0±45.7
10	10	0.915 ± 0.047	0.859 ± 0.058	82.3±14.5	254.1±53.5
10	25	0.922 ± 0.038	0.892 ± 0.042	128.8±20.8	386.1±62.7
10	100	0.918 ± 0.041	0.894 ± 0.048	163.9±40.0	409.6±60.2
10	200	0.923 ± 0.040	0.901 ± 0.043	170.1±42.6	416.1±59.0
25	5	0.916 ± 0.048	0.846 ± 0.062	67.8±11.1	123.6±27.7
25	10	0.933 ± 0.044	0.876 ± 0.056	99.1±15.7	249.8±53.7
25	25	0.947 ± 0.032	0.896 ± 0.047	144.4±23.9	376.8±58.0
25	100	0.948 ± 0.034	0.907 ± 0.042	184.8±42.9	406.9±63.2
25	200	0.947 ± 0.032	0.911 ± 0.041	197.9±49.6	419.3±62.0
100	5	0.969 ± 0.021	0.915 ± 0.041	145.2±10.3	129.1±20.0
100	10	0.980 ± 0.017	0.937 ± 0.034	184.0±17.5	251.0±35.2
100	25	0.984 ± 0.016	0.943 ± 0.030	238.1±26.0	375.7±37.1
100	100	0.984 ± 0.013	0.941 ± 0.032	277.6±44.3	413.1±43.7
100	200	0.986 ± 0.012	0.946 ± 0.030	279.7±45.4	414.9±43.4
200	5	0.982 ± 0.014	0.939 ± 0.032	245.9±12.2	126.4±15.7
200	10	0.989 ± 0.011	0.953 ± 0.027	285.2±18.1	251.5±23.2
200	25	0.992 ± 0.009	0.960 ± 0.024	348.5±24.1	375.5±29.0
200	100	0.993 ± 0.010	0.965 ± 0.024	409.2±41.6	419.5±31.8
200	200	0.993 ± 0.007	0.961 ± 0.021	424.4±46.4	411.9±34.0
500	5	0.990 ± 0.011	0.963 ± 0.021	544.5±12.7	130.6±11.5
500	10	0.993 ± 0.010	0.967 ± 0.022	583.9±16.8	255.5±15.5
500	25	0.996 ± 0.007	0.977 ± 0.018	642.7±23.6	377.2±18.4
500	100	0.996 ± 0.006	0.974 ± 0.019	764.1±38.8	414.7±23.0
500	200	0.997 ± 0.005	0.978 ± 0.016	862.9±43.5	415.1±19.9

Table A.47: Musk_1, Unique root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.724 ± 0.158	0.665 ± 0.136	34.2±4.9	84.8±12.2
10	10	0.701 ± 0.171	0.654 ± 0.157	42.7±5.8	103.6±8.2
10	25	0.680 ± 0.175	0.644 ± 0.170	51.3±9.7	109.4±9.3
10	100	0.717 ± 0.190	0.667 ± 0.170	53.3±14.1	109.1±8.6
10	200	0.714 ± 0.168	0.668 ± 0.151	51.1±13.9	108.6±9.4
25	5	0.732 ± 0.185	0.683 ± 0.151	48.6±4.4	85.1±8.8
25	10	0.736 ± 0.183	0.667 ± 0.168	56.9±5.7	102.8±6.9
25	25	0.746 ± 0.153	0.695 ± 0.144	65.1±10.6	109.4±7.5
25	100	0.752 ± 0.172	0.682 ± 0.171	70.3±17.3	107.7±8.2
25	200	0.752 ± 0.164	0.689 ± 0.151	70.7±13.7	108.2±7.6
100	5	0.856 ± 0.126	0.770 ± 0.142	127.1±5.6	84.0±4.5
100	10	0.853 ± 0.131	0.771 ± 0.122	136.7±6.3	102.5±4.1
100	25	0.843 ± 0.112	0.761 ± 0.121	146.2±10.3	108.7±4.2
100	100	0.850 ± 0.129	0.763 ± 0.121	147.1±14.4	110.1±4.6
100	200	0.829 ± 0.134	0.749 ± 0.143	147.6±13.3	109.9±4.0
200	5	0.896 ± 0.103	0.801 ± 0.128	228.5±5.1	83.9±3.5
200	10	0.878 ± 0.103	0.772 ± 0.113	241.6±5.8	101.6±2.9
200	25	0.893 ± 0.112	0.811 ± 0.122	253.0±11.1	108.7±2.9
200	100	0.894 ± 0.119	0.801 ± 0.125	258.1±14.1	109.5±3.2
200	200	0.905 ± 0.093	0.812 ± 0.113	259.2±16.6	109.6±3.2
500	5	0.917 ± 0.092	0.824 ± 0.110	529.0±4.9	84.6±1.9
500	10	0.920 ± 0.086	0.826 ± 0.113	542.0±6.0	101.8±2.0
500	25	0.919 ± 0.087	0.819 ± 0.114	565.4±10.8	109.0±2.0
500	100	0.909 ± 0.094	0.825 ± 0.109	618.8±18.3	109.4±2.1
500	200	0.916 ± 0.082	0.813 ± 0.108	617.5±20.9	109.6±2.0

Table A.48: Mutagenesis$_{All}$, Unique root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.736 ± 0.095	0.702 ± 0.091	41.3±9.6	48.6±13.3
10	10	0.753 ± 0.111	0.719 ± 0.094	66.6±13.6	87.2±20.4
10	25	0.746 ± 0.106	0.726 ± 0.091	124.2±23.0	147.9±19.4
10	100	0.737 ± 0.094	0.715 ± 0.084	305.2±52.8	200.6±13.0
10	200	0.745 ± 0.092	0.721 ± 0.088	474.4±66.0	208.4±12.1
25	5	0.768 ± 0.105	0.723 ± 0.089	58.5±8.4	53.6±12.8
25	10	0.766 ± 0.104	0.746 ± 0.089	90.3±13.9	91.1±17.5
25	25	0.780 ± 0.113	0.747 ± 0.089	153.1±25.0	145.5±17.0
25	100	0.773 ± 0.102	0.731 ± 0.083	347.0±65.8	199.8±10.8
25	200	0.774 ± 0.089	0.735 ± 0.087	524.8±91.5	210.2±11.1
100	5	0.794 ± 0.100	0.731 ± 0.080	134.0±9.9	52.9±6.9
100	10	0.784 ± 0.098	0.750 ± 0.091	165.7±12.7	90.9±11.7
100	25	0.795 ± 0.091	0.749 ± 0.075	247.2±24.3	147.9±13.3
100	100	0.795 ± 0.095	0.750 ± 0.092	515.6±59.8	199.6±8.2
100	200	0.807 ± 0.092	0.757 ± 0.096	756.7±76.1	209.0±7.9
200	5	0.799 ± 0.094	0.746 ± 0.080	234.2±8.3	52.9±6.3
200	10	0.809 ± 0.093	0.762 ± 0.088	265.6±14.6	89.8±9.1
200	25	0.815 ± 0.096	0.766 ± 0.084	342.7±26.6	146.6±12.5
200	100	0.800 ± 0.091	0.755 ± 0.092	617.6±55.2	199.7±8.6
200	200	0.804 ± 0.091	0.757 ± 0.084	893.3±81.9	209.8±6.9
500	5	0.805 ± 0.096	0.752 ± 0.086	534.8±8.7	52.7±4.0
500	10	0.812 ± 0.101	0.763 ± 0.082	566.5±14.0	90.8±5.6
500	25	0.820 ± 0.091	0.773 ± 0.087	643.0±23.1	147.4±9.0
500	100	0.814 ± 0.092	0.763 ± 0.086	913.9±53.5	199.6±6.6
500	200	0.809 ± 0.096	0.755 ± 0.093	1207.8±89.4	210.5±6.2

Table A.49: Mutagenesis$_{RF}$, Unique root, Normal leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.758 ± 0.127	0.741 ± 0.086	40.1±8.6	48.8±11.9
10	10	0.809 ± 0.103	0.779 ± 0.084	64.3±12.6	80.4±14.2
10	25	0.808 ± 0.120	0.786 ± 0.099	115.8±22.5	121.4±15.7
10	100	0.817 ± 0.109	0.797 ± 0.098	271.4±53.1	151.9±11.8
10	200	0.807 ± 0.103	0.785 ± 0.085	415.9±83.7	156.0±12.0
25	5	0.793 ± 0.116	0.755 ± 0.089	57.5±7.5	48.1±10.2
25	10	0.845 ± 0.093	0.791 ± 0.082	86.2±11.6	79.9±14.0
25	25	0.844 ± 0.107	0.808 ± 0.089	138.5±26.6	122.6±13.8
25	100	0.831 ± 0.086	0.786 ± 0.083	297.3±48.5	151.6±9.7
25	200	0.844 ± 0.091	0.798 ± 0.086	441.7±79.3	156.0±10.1
100	5	0.842 ± 0.094	0.781 ± 0.096	132.3±8.1	49.8±7.0
100	10	0.863 ± 0.097	0.808 ± 0.097	162.7±13.1	78.8±9.4
100	25	0.880 ± 0.082	0.822 ± 0.088	233.0±23.3	120.9±8.6
100	100	0.875 ± 0.084	0.817 ± 0.082	471.5±58.4	152.1±7.5
100	200	0.873 ± 0.088	0.820 ± 0.079	694.2±80.8	156.6±5.9
200	5	0.860 ± 0.096	0.804 ± 0.089	233.5±7.8	48.0±5.3
200	10	0.881 ± 0.079	0.822 ± 0.083	261.2±11.9	79.3±6.8
200	25	0.884 ± 0.086	0.823 ± 0.091	339.2±25.5	118.5±8.2
200	100	0.894 ± 0.079	0.839 ± 0.076	574.5±54.9	152.4±6.4
200	200	0.878 ± 0.083	0.813 ± 0.080	821.6±80.8	157.1±5.5
500	5	0.868 ± 0.080	0.811 ± 0.082	531.6±7.5	48.5±2.9
500	10	0.886 ± 0.083	0.823 ± 0.081	564.5±14.0	79.5±4.8
500	25	0.898 ± 0.078	0.844 ± 0.079	635.9±25.2	121.1±5.0
500	100	0.893 ± 0.079	0.838 ± 0.081	879.3±46.6	152.3±5.6
500	200	0.886 ± 0.078	0.832 ± 0.078	1140.6±75.0	156.5±4.7

Table A.50: Carcinogenesis, Unique root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.581 ± 0.091	0.563 ± 0.075	40.9±8.9	90.7±29.5
10	10	0.589 ± 0.090	0.573 ± 0.076	69.6±14.9	172.9±40.2
10	25	0.574 ± 0.091	0.570 ± 0.085	131.8±22.9	297.2±31.8
10	100	0.570 ± 0.087	0.563 ± 0.077	332.8±54.0	392.5±15.7
10	200	0.573 ± 0.091	0.563 ± 0.080	527.4±95.2	408.4±12.7
25	5	0.569 ± 0.107	0.559 ± 0.082	59.2±7.8	89.7±27.4
25	10	0.586 ± 0.087	0.578 ± 0.071	90.7±15.6	166.7±39.2
25	25	0.596 ± 0.089	0.579 ± 0.075	153.6±27.1	290.8±28.0
25	100	0.583 ± 0.091	0.572 ± 0.083	345.0±56.2	389.8±13.8
25	200	0.575 ± 0.090	0.569 ± 0.084	555.7±104.5	407.4±12.7
100	5	0.597 ± 0.102	0.581 ± 0.081	134.8±9.1	87.9±15.2
100	10	0.608 ± 0.088	0.582 ± 0.086	169.0±14.0	167.5±20.4
100	25	0.604 ± 0.089	0.578 ± 0.072	251.6±21.5	287.0±21.6
100	100	0.611 ± 0.083	0.578 ± 0.080	511.8±61.5	390.1±11.2
100	200	0.585 ± 0.091	0.563 ± 0.084	744.4±115.5	407.7±9.0
200	5	0.599 ± 0.086	0.574 ± 0.070	235.7±9.6	88.8±11.5
200	10	0.606 ± 0.092	0.587 ± 0.085	269.6±14.8	170.5±18.7
200	25	0.604 ± 0.084	0.588 ± 0.072	351.0±23.5	290.4±18.6
200	100	0.603 ± 0.093	0.581 ± 0.082	630.5±61.0	391.0±9.3
200	200	0.599 ± 0.088	0.575 ± 0.079	933.5±100.6	408.5±7.6
500	5	0.607 ± 0.086	0.580 ± 0.066	534.0±8.8	88.0±7.8
500	10	0.625 ± 0.080	0.592 ± 0.072	570.2±13.4	169.5±12.4
500	25	0.627 ± 0.089	0.601 ± 0.080	650.8±21.0	292.1±11.7
500	100	0.605 ± 0.078	0.588 ± 0.067	913.6±53.3	388.8±7.3
500	200	0.602 ± 0.081	0.566 ± 0.071	1261.2±100.9	409.6±6.6

Table A.51: $\text{Diterpenes}_{52.3}$, Unique root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.881 ± 0.059	0.809 ± 0.063	52.2±11.1	167.4±56.7
10	10	0.908 ± 0.044	0.852 ± 0.052	83.2±16.1	322.7±63.0
10	25	0.929 ± 0.038	0.894 ± 0.040	124.6±20.4	452.1±56.8
10	100	0.921 ± 0.036	0.889 ± 0.042	159.2±36.0	494.0±62.6
10	200	0.920 ± 0.039	0.889 ± 0.045	161.2±35.4	506.6±70.7
25	5	0.909 ± 0.044	0.837 ± 0.053	66.3±9.9	174.3±47.8
25	10	0.939 ± 0.034	0.879 ± 0.044	102.1±15.1	332.0±53.7
25	25	0.947 ± 0.029	0.894 ± 0.041	140.3±22.3	466.8±55.2
25	100	0.944 ± 0.031	0.890 ± 0.041	176.7±37.1	508.3±64.5
25	200	0.937 ± 0.041	0.889 ± 0.045	173.9±40.5	498.9±66.3
100	5	0.964 ± 0.023	0.909 ± 0.040	144.0±8.6	172.1±31.8
100	10	0.975 ± 0.016	0.925 ± 0.031	177.9±12.0	320.3±38.2
100	25	0.979 ± 0.018	0.933 ± 0.034	220.4±22.9	466.9±37.7
100	100	0.981 ± 0.016	0.935 ± 0.030	251.2±38.3	493.0±42.0
100	200	0.977 ± 0.017	0.926 ± 0.035	256.6±43.6	506.9±42.4
200	5	0.976 ± 0.017	0.929 ± 0.032	245.6±12.0	162.2±21.4
200	10	0.986 ± 0.011	0.944 ± 0.027	282.5±14.3	320.7±26.4
200	25	0.990 ± 0.009	0.953 ± 0.027	331.5±26.7	468.1±29.5
200	100	0.989 ± 0.010	0.951 ± 0.025	374.7±41.3	502.4±35.6
200	200	0.989 ± 0.009	0.953 ± 0.025	376.6±49.7	493.5±30.4
500	5	0.985 ± 0.011	0.949 ± 0.025	546.3±10.4	166.9±12.7
500	10	0.992 ± 0.008	0.961 ± 0.022	585.7±17.2	322.7±16.7
500	25	0.994 ± 0.007	0.966 ± 0.021	639.3±20.6	467.3±18.6
500	100	0.994 ± 0.006	0.968 ± 0.019	758.8±43.5	503.1±20.3
500	200	0.996 ± 0.005	0.969 ± 0.019	764.1±47.1	503.6±20.5

Table A.52: Diterpenes$_{52.54}$, Unique root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.809 ± 0.067	0.738 ± 0.065	54.9±11.6	176.8±61.6
10	10	0.864 ± 0.057	0.802 ± 0.062	92.7±16.0	358.5±65.6
10	25	0.874 ± 0.047	0.831 ± 0.048	140.7±24.4	564.7±64.5
10	100	0.872 ± 0.050	0.837 ± 0.050	191.7±38.8	643.2±73.0
10	200	0.870 ± 0.044	0.832 ± 0.044	195.3±43.0	649.4±69.3
25	5	0.840 ± 0.066	0.766 ± 0.068	71.0±13.4	166.9±48.0
25	10	0.879 ± 0.046	0.806 ± 0.057	106.6±17.9	359.0±61.9
25	25	0.900 ± 0.042	0.839 ± 0.051	158.6±27.4	571.4±61.1
25	100	0.901 ± 0.041	0.847 ± 0.048	210.4±42.9	634.6±66.0
25	200	0.901 ± 0.040	0.840 ± 0.045	216.7±49.2	640.7±56.7
100	5	0.918 ± 0.043	0.833 ± 0.056	149.3±13.8	167.2±30.9
100	10	0.941 ± 0.034	0.872 ± 0.046	187.7±16.4	368.3±44.2
100	25	0.950 ± 0.027	0.884 ± 0.041	241.9±25.9	567.5±41.7
100	100	0.946 ± 0.032	0.879 ± 0.047	283.3±41.4	637.4±52.4
100	200	0.950 ± 0.025	0.880 ± 0.037	290.8±45.6	639.2±54.3
200	5	0.949 ± 0.028	0.873 ± 0.047	249.0±12.9	176.7±23.1
200	10	0.965 ± 0.021	0.901 ± 0.036	291.4±15.9	364.8±34.2
200	25	0.972 ± 0.018	0.916 ± 0.037	348.3±24.9	569.5±29.7
200	100	0.973 ± 0.016	0.918 ± 0.031	404.3±41.4	627.1±34.6
200	200	0.970 ± 0.020	0.915 ± 0.038	411.0±57.3	635.8±39.7
500	5	0.964 ± 0.022	0.893 ± 0.040	550.0±11.1	172.2±14.4
500	10	0.984 ± 0.012	0.937 ± 0.025	592.6±19.3	366.5±20.1
500	25	0.984 ± 0.012	0.939 ± 0.026	661.2±25.5	574.7±21.7
500	100	0.986 ± 0.011	0.939 ± 0.025	784.2±37.7	636.1±22.5
500	200	0.987 ± 0.010	0.943 ± 0.024	819.4±55.7	635.9±22.4

Table A.53: Diterpenes$_{54.3}$, Unique root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.899 ± 0.067	0.837 ± 0.078	51.9±10.3	145.1±45.2
10	10	0.934 ± 0.039	0.886 ± 0.046	85.7±14.7	271.1±58.7
10	25	0.940 ± 0.037	0.905 ± 0.043	121.9±20.9	371.1±64.3
10	100	0.943 ± 0.032	0.911 ± 0.039	151.0±37.2	400.1±63.6
10	200	0.936 ± 0.036	0.906 ± 0.038	145.0±32.7	404.0±63.8
25	5	0.933 ± 0.041	0.863 ± 0.062	69.4±10.8	153.6±42.5
25	10	0.953 ± 0.033	0.899 ± 0.050	103.8±15.9	259.0±49.5
25	25	0.960 ± 0.029	0.917 ± 0.041	137.0±26.2	372.3±52.6
25	100	0.954 ± 0.030	0.914 ± 0.038	167.1±32.2	412.4±59.2
25	200	0.954 ± 0.031	0.910 ± 0.039	170.2±43.3	410.5±66.3
100	5	0.977 ± 0.019	0.934 ± 0.035	145.5±10.6	144.8±22.5
100	10	0.983 ± 0.016	0.941 ± 0.033	180.0±15.9	264.9±28.6
100	25	0.987 ± 0.013	0.948 ± 0.033	219.5±23.4	378.4±40.0
100	100	0.987 ± 0.010	0.948 ± 0.028	247.3±34.5	403.1±46.2
100	200	0.987 ± 0.012	0.946 ± 0.029	248.5±39.1	402.0±44.4
200	5	0.987 ± 0.014	0.956 ± 0.028	248.1±11.1	145.0±17.8
200	10	0.991 ± 0.010	0.962 ± 0.023	285.1±14.8	271.8±24.9
200	25	0.994 ± 0.007	0.965 ± 0.024	332.9±23.6	378.7±27.8
200	100	0.993 ± 0.009	0.968 ± 0.022	363.0±34.6	400.7±29.2
200	200	0.994 ± 0.007	0.967 ± 0.023	358.2±36.6	402.6±33.4
500	5	0.991 ± 0.010	0.963 ± 0.023	549.8±12.4	146.4±12.2
500	10	0.995 ± 0.008	0.974 ± 0.019	585.3±15.0	269.3±16.8
500	25	0.997 ± 0.005	0.977 ± 0.018	637.0±20.9	377.2±19.0
500	100	0.997 ± 0.004	0.979 ± 0.018	741.2±36.8	400.6±22.9
500	200	0.997 ± 0.005	0.979 ± 0.016	774.2±41.4	403.8±19.8

Table A.54: Musk_1, Unique root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.727 ± 0.183	0.679 ± 0.155	34.4±4.6	94.0±9.6
10	10	0.749 ± 0.162	0.686 ± 0.152	40.1±5.6	104.3±8.7
10	25	0.721 ± 0.162	0.674 ± 0.142	44.7±7.8	109.9±8.0
10	100	0.730 ± 0.171	0.661 ± 0.162	47.3±13.5	109.2±8.4
10	200	0.739 ± 0.165	0.694 ± 0.143	46.7±9.3	110.2±8.7
25	5	0.811 ± 0.144	0.744 ± 0.142	49.4±4.2	92.7±6.9
25	10	0.767 ± 0.164	0.718 ± 0.145	56.0±6.0	105.8±6.1
25	25	0.822 ± 0.142	0.749 ± 0.136	60.8±8.7	108.6±5.8
25	100	0.781 ± 0.155	0.723 ± 0.135	62.2±12.2	110.8±6.2
25	200	0.784 ± 0.147	0.713 ± 0.132	63.2±11.4	108.1±7.0
100	5	0.865 ± 0.117	0.774 ± 0.129	124.8±4.9	92.9±3.6
100	10	0.856 ± 0.105	0.765 ± 0.124	130.8±6.6	104.9±3.4
100	25	0.872 ± 0.101	0.775 ± 0.110	135.0±9.3	109.7±3.4
100	100	0.871 ± 0.126	0.796 ± 0.132	140.3±11.5	109.5±3.6
100	200	0.858 ± 0.127	0.779 ± 0.132	136.8±10.7	109.5±3.6
200	5	0.903 ± 0.094	0.810 ± 0.117	224.5±4.9	92.8±3.0
200	10	0.902 ± 0.106	0.813 ± 0.116	230.8±6.3	105.5±2.8
200	25	0.902 ± 0.093	0.803 ± 0.116	236.0±8.3	109.2±2.9
200	100	0.905 ± 0.090	0.816 ± 0.110	238.6±12.3	109.4±2.9
200	200	0.883 ± 0.103	0.783 ± 0.117	237.0±8.6	109.8±2.9
500	5	0.922 ± 0.073	0.815 ± 0.107	525.3±4.4	92.8±2.2
500	10	0.917 ± 0.082	0.820 ± 0.116	530.3±5.7	105.2±1.9
500	25	0.923 ± 0.079	0.828 ± 0.112	536.2±8.4	109.2±1.8
500	100	0.911 ± 0.080	0.823 ± 0.096	538.6±14.8	109.4±2.0
500	200	0.921 ± 0.081	0.811 ± 0.106	537.2±11.0	109.5±2.0

Table A.55: Mutagenesis_{All}, Unique root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.737 ± 0.106	0.705 ± 0.081	40.0±9.7	51.8±14.1
10	10	0.736 ± 0.116	0.716 ± 0.096	70.4±13.1	94.3±17.7
10	25	0.756 ± 0.108	0.735 ± 0.092	128.0±27.0	149.6±22.0
10	100	0.747 ± 0.096	0.720 ± 0.088	300.3±52.2	200.2±13.2
10	200	0.743 ± 0.096	0.724 ± 0.084	487.0±88.4	210.3±11.3
25	5	0.745 ± 0.120	0.702 ± 0.095	58.7±8.4	55.2±14.2
25	10	0.776 ± 0.092	0.738 ± 0.081	90.1±13.3	90.4±16.6
25	25	0.776 ± 0.096	0.744 ± 0.087	155.5±27.8	145.8±17.6
25	100	0.769 ± 0.095	0.725 ± 0.090	334.0±60.9	201.1±11.7
25	200	0.763 ± 0.098	0.727 ± 0.085	515.4±81.6	209.0±10.9
100	5	0.781 ± 0.104	0.731 ± 0.087	134.1±7.8	51.8±7.4
100	10	0.803 ± 0.098	0.751 ± 0.087	165.1±12.7	92.6±12.6
100	25	0.801 ± 0.090	0.759 ± 0.085	242.7±24.4	147.4±14.4
100	100	0.808 ± 0.095	0.752 ± 0.088	511.4±52.2	200.9±8.0
100	200	0.807 ± 0.088	0.752 ± 0.080	727.8±83.5	210.3±7.1
200	5	0.798 ± 0.103	0.750 ± 0.087	235.7±9.1	52.5±5.1
200	10	0.812 ± 0.090	0.768 ± 0.077	266.0±12.8	91.3±9.1
200	25	0.809 ± 0.088	0.767 ± 0.075	345.2±26.6	149.1±11.2
200	100	0.809 ± 0.090	0.756 ± 0.094	614.7±50.0	200.9±8.7
200	200	0.804 ± 0.092	0.757 ± 0.084	903.9±91.0	210.0±6.8
500	5	0.806 ± 0.095	0.754 ± 0.086	535.0±8.9	53.7±4.1
500	10	0.817 ± 0.090	0.770 ± 0.085	566.7±13.1	91.7±6.2
500	25	0.819 ± 0.091	0.778 ± 0.085	640.0±23.7	147.8±6.9
500	100	0.824 ± 0.087	0.771 ± 0.083	911.9±49.8	200.3±6.7
500	200	0.815 ± 0.091	0.766 ± 0.086	1217.0±82.5	209.8±6.4

Table A.56: Mutagenesis$_{RF}$, Unique root, Random leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.773 ± 0.125	0.747 ± 0.101	41.2±7.8	46.8±10.5
10	10	0.807 ± 0.114	0.773 ± 0.091	63.7±13.3	80.3±18.0
10	25	0.829 ± 0.101	0.800 ± 0.092	118.5±19.9	121.8±12.7
10	100	0.810 ± 0.107	0.795 ± 0.093	261.8±49.8	152.6±12.3
10	200	0.805 ± 0.109	0.787 ± 0.090	398.7±66.4	156.9±10.6
25	5	0.819 ± 0.106	0.758 ± 0.089	57.2±7.6	49.8±9.5
25	10	0.841 ± 0.093	0.788 ± 0.082	86.3±15.0	83.4±13.3
25	25	0.857 ± 0.093	0.812 ± 0.080	149.1±23.4	122.7±12.3
25	100	0.835 ± 0.107	0.798 ± 0.088	305.1±61.2	150.6±10.1
25	200	0.824 ± 0.109	0.792 ± 0.095	441.9±87.9	155.8±8.6
100	5	0.843 ± 0.102	0.767 ± 0.089	133.3±8.6	47.9±6.3
100	10	0.867 ± 0.092	0.813 ± 0.076	161.5±11.9	80.4±8.9
100	25	0.874 ± 0.082	0.821 ± 0.073	230.7±22.1	122.8±9.5
100	100	0.877 ± 0.090	0.822 ± 0.087	460.3±49.6	153.3±5.9
100	200	0.876 ± 0.087	0.813 ± 0.087	667.6±87.7	157.6±5.8
200	5	0.863 ± 0.091	0.795 ± 0.093	232.4±7.7	48.5±4.8
200	10	0.884 ± 0.078	0.814 ± 0.081	263.5±13.6	81.3±6.6
200	25	0.887 ± 0.077	0.828 ± 0.079	330.9±21.5	121.3±7.8
200	100	0.882 ± 0.076	0.831 ± 0.081	573.9±47.5	151.8±5.5
200	200	0.873 ± 0.087	0.817 ± 0.083	831.0±79.6	156.3±5.4
500	5	0.875 ± 0.079	0.814 ± 0.084	532.8±8.0	49.2±2.8
500	10	0.888 ± 0.078	0.832 ± 0.081	564.4±12.0	80.6±4.9
500	25	0.898 ± 0.073	0.833 ± 0.077	633.1±24.8	122.0±4.9
500	100	0.891 ± 0.077	0.837 ± 0.077	876.5±60.9	152.4±5.1
500	200	0.886 ± 0.077	0.827 ± 0.071	1147.2±69.7	156.7±5.4

Table A.57: Carcinogenesis, Unique root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.557 ± 0.098	0.555 ± 0.080	40.3±9.3	84.1±29.1
10	10	0.576 ± 0.095	0.566 ± 0.085	70.8±14.0	168.5±39.5
10	25	0.566 ± 0.102	0.562 ± 0.084	136.0±23.2	288.6±33.1
10	100	0.578 ± 0.091	0.565 ± 0.083	314.1±57.1	388.1±15.6
10	200	0.564 ± 0.091	0.552 ± 0.081	528.2±83.5	406.6±16.2
25	5	0.572 ± 0.091	0.555 ± 0.077	59.5±9.2	89.5±25.7
25	10	0.576 ± 0.099	0.565 ± 0.086	91.0±13.1	167.9±30.9
25	25	0.584 ± 0.094	0.572 ± 0.078	150.4±21.7	293.0±28.5
25	100	0.583 ± 0.103	0.572 ± 0.088	336.7±58.5	388.7±16.7
25	200	0.583 ± 0.088	0.566 ± 0.081	563.4±95.6	407.9±13.8
100	5	0.600 ± 0.083	0.581 ± 0.068	134.1±8.6	88.1±17.4
100	10	0.606 ± 0.086	0.584 ± 0.079	170.2±14.0	175.6±21.6
100	25	0.597 ± 0.088	0.578 ± 0.080	251.1±22.6	291.8±23.4
100	100	0.601 ± 0.085	0.576 ± 0.082	497.0±54.7	388.4±12.3
100	200	0.603 ± 0.091	0.566 ± 0.074	705.5±96.9	406.5±10.5
200	5	0.617 ± 0.092	0.593 ± 0.075	234.5±8.3	88.9±10.9
200	10	0.609 ± 0.083	0.588 ± 0.076	270.4±14.2	170.5±16.6
200	25	0.610 ± 0.089	0.582 ± 0.087	349.8±23.2	292.3±16.9
200	100	0.601 ± 0.086	0.573 ± 0.076	631.6±65.2	388.5±9.3
200	200	0.620 ± 0.090	0.585 ± 0.081	943.7±104.4	407.7±7.6
500	5	0.608 ± 0.085	0.582 ± 0.069	533.3±8.4	88.1±7.6
500	10	0.619 ± 0.082	0.588 ± 0.073	568.5±14.3	167.5±11.6
500	25	0.616 ± 0.085	0.585 ± 0.076	653.3±24.2	290.1±11.1
500	100	0.603 ± 0.084	0.576 ± 0.074	933.3±53.9	389.3±6.7
500	200	0.601 ± 0.083	0.566 ± 0.076	1263.1±104.6	407.1±6.4

Table A.58: $\text{Diterpenes}_{52.3}$, Unique root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.882 ± 0.057	0.815 ± 0.062	50.4±9.2	168.8±52.8
10	10	0.911 ± 0.041	0.855 ± 0.048	83.3±14.1	319.3±60.6
10	25	0.921 ± 0.040	0.886 ± 0.045	118.9±22.1	449.7±65.9
10	100	0.907 ± 0.040	0.883 ± 0.041	155.3±34.4	485.8±59.5
10	200	0.910 ± 0.036	0.883 ± 0.040	156.1±40.9	494.3±63.5
25	5	0.906 ± 0.045	0.831 ± 0.055	66.5±10.9	170.9±51.8
25	10	0.936 ± 0.031	0.879 ± 0.041	98.3±14.4	314.7±59.1
25	25	0.938 ± 0.032	0.892 ± 0.040	138.3±20.7	455.7±61.9
25	100	0.940 ± 0.033	0.893 ± 0.037	166.8±35.9	491.9±64.3
25	200	0.939 ± 0.035	0.893 ± 0.038	172.2±41.8	484.8±58.1
100	5	0.963 ± 0.024	0.900 ± 0.040	143.8±12.4	163.8±27.9
100	10	0.975 ± 0.020	0.927 ± 0.035	174.9±14.2	315.0±35.3
100	25	0.977 ± 0.017	0.928 ± 0.039	215.9±24.7	457.3±41.1
100	100	0.976 ± 0.015	0.927 ± 0.030	259.7±43.6	495.2±48.1
100	200	0.979 ± 0.015	0.929 ± 0.033	262.2±48.3	483.4±43.3
200	5	0.976 ± 0.017	0.930 ± 0.035	244.4±10.9	166.0±20.7
200	10	0.986 ± 0.012	0.946 ± 0.030	283.1±14.6	315.3±27.0
200	25	0.988 ± 0.011	0.952 ± 0.026	326.3±20.5	457.9±30.0
200	100	0.988 ± 0.010	0.950 ± 0.025	361.8±36.8	494.8±34.1
200	200	0.988 ± 0.011	0.951 ± 0.025	364.2±39.4	493.0±32.8
500	5	0.985 ± 0.013	0.944 ± 0.027	547.5±10.8	162.3±13.5
500	10	0.991 ± 0.008	0.957 ± 0.024	581.7±13.6	316.7±20.1
500	25	0.994 ± 0.007	0.967 ± 0.020	637.3±21.9	460.7±20.5
500	100	0.995 ± 0.005	0.969 ± 0.019	738.8±32.8	489.3±24.3
500	200	0.995 ± 0.005	0.972 ± 0.018	752.1±48.4	494.3±22.5

Table A.59: $\text{Diterpenes}_{52.54}$, Unique root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.803 ± 0.069	0.735 ± 0.066	54.7±11.3	172.1±57.4
10	10	0.858 ± 0.058	0.796 ± 0.061	89.8±16.6	371.8±67.9
10	25	0.870 ± 0.042	0.833 ± 0.046	136.9±23.0	558.0±54.9
10	100	0.868 ± 0.049	0.839 ± 0.044	188.9±42.9	625.7±66.7
10	200	0.860 ± 0.045	0.833 ± 0.044	202.3±45.4	645.3±66.0
25	5	0.839 ± 0.062	0.767 ± 0.060	70.4±11.4	176.1±50.8
25	10	0.888 ± 0.050	0.820 ± 0.053	105.1±17.0	357.7±67.5
25	25	0.901 ± 0.045	0.844 ± 0.046	158.1±25.5	551.3±58.6
25	100	0.890 ± 0.047	0.845 ± 0.052	206.3±43.0	637.8±64.4
25	200	0.895 ± 0.038	0.840 ± 0.041	208.1±47.1	617.1±65.3
100	5	0.919 ± 0.044	0.842 ± 0.056	144.8±12.0	177.3±30.2
100	10	0.937 ± 0.033	0.869 ± 0.044	182.2±15.2	362.3±40.6
100	25	0.950 ± 0.029	0.883 ± 0.044	234.2±22.1	559.3±40.9
100	100	0.950 ± 0.030	0.887 ± 0.041	281.4±40.1	621.1±48.4
100	200	0.947 ± 0.030	0.876 ± 0.041	293.7±54.7	633.4±49.7
200	5	0.947 ± 0.031	0.873 ± 0.046	247.0±10.7	174.5±22.8
200	10	0.967 ± 0.020	0.905 ± 0.037	289.6±15.4	361.9±31.1
200	25	0.971 ± 0.021	0.918 ± 0.036	353.4±23.0	559.6±30.8
200	100	0.972 ± 0.017	0.918 ± 0.033	392.8±38.0	628.9±37.3
200	200	0.973 ± 0.018	0.913 ± 0.037	405.0±50.0	625.3±34.3
500	5	0.968 ± 0.021	0.908 ± 0.037	549.4±11.3	175.5±13.7
500	10	0.981 ± 0.014	0.934 ± 0.027	592.3±16.0	362.2±22.8
500	25	0.984 ± 0.012	0.938 ± 0.027	655.8±26.6	565.0±22.0
500	100	0.986 ± 0.011	0.943 ± 0.025	775.3±41.9	626.0±23.9
500	200	0.987 ± 0.011	0.944 ± 0.029	814.7±54.3	627.9±21.8

Table A.60: Diterpenes$_{54.3}$, Unique root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.908 ± 0.049	0.835 ± 0.064	51.6±10.8	143.6±35.3
10	10	0.928 ± 0.047	0.882 ± 0.057	80.9±15.7	265.0±49.6
10	25	0.937 ± 0.034	0.905 ± 0.037	119.2±22.1	362.0±65.9
10	100	0.937 ± 0.036	0.910 ± 0.041	149.5±36.7	389.3±64.9
10	200	0.929 ± 0.039	0.902 ± 0.039	148.4±43.1	404.7±70.4
25	5	0.926 ± 0.043	0.861 ± 0.057	65.1±10.1	149.4±44.8
25	10	0.953 ± 0.032	0.898 ± 0.047	95.7±13.4	263.9±44.9
25	25	0.956 ± 0.028	0.910 ± 0.037	133.0±22.0	379.7±56.3
25	100	0.958 ± 0.028	0.919 ± 0.040	156.4±34.9	393.4±65.2
25	200	0.957 ± 0.032	0.923 ± 0.042	162.3±37.7	387.3±64.7
100	5	0.975 ± 0.020	0.925 ± 0.035	145.5±10.4	141.1±22.4
100	10	0.984 ± 0.014	0.945 ± 0.030	176.5±16.4	267.6±35.1
100	25	0.986 ± 0.013	0.946 ± 0.029	214.6±21.5	363.1±40.1
100	100	0.984 ± 0.016	0.943 ± 0.032	244.2±37.6	394.9±49.6
100	200	0.986 ± 0.013	0.948 ± 0.028	248.7±41.6	395.3±48.4
200	5	0.986 ± 0.014	0.952 ± 0.029	242.3±11.2	140.4±14.3
200	10	0.991 ± 0.010	0.961 ± 0.022	277.0±16.1	261.7±22.2
200	25	0.993 ± 0.008	0.968 ± 0.024	328.1±23.5	364.1±30.1
200	100	0.993 ± 0.009	0.969 ± 0.024	354.3±37.4	390.3±34.7
200	200	0.993 ± 0.008	0.965 ± 0.024	365.8±40.6	390.8±29.2
500	5	0.991 ± 0.011	0.966 ± 0.023	546.3±12.0	144.1±12.0
500	10	0.994 ± 0.008	0.970 ± 0.020	579.1±14.2	265.0±17.3
500	25	0.997 ± 0.005	0.979 ± 0.017	634.0±22.7	369.0±20.0
500	100	0.997 ± 0.004	0.981 ± 0.015	737.0±34.8	389.6±18.4
500	200	0.997 ± 0.005	0.978 ± 0.017	773.1±48.2	390.5±20.3

Table A.61: Musk$_1$, Unique root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.716 ± 0.166	0.658 ± 0.163	33.7±5.2	86.9±10.2
10	10	0.719 ± 0.172	0.671 ± 0.154	38.4±5.3	99.2±7.9
10	25	0.730 ± 0.160	0.680 ± 0.137	45.4±9.4	102.8±8.8
10	100	0.753 ± 0.153	0.681 ± 0.151	45.8±10.0	103.1±9.0
10	200	0.734 ± 0.169	0.681 ± 0.155	45.4±10.7	103.9±9.1
25	5	0.790 ± 0.167	0.699 ± 0.138	48.8±4.2	87.5±7.8
25	10	0.793 ± 0.162	0.717 ± 0.165	54.4±5.7	99.4±6.9
25	25	0.786 ± 0.146	0.700 ± 0.140	60.1±8.2	103.9±6.4
25	100	0.782 ± 0.135	0.733 ± 0.121	63.7±14.5	102.3±7.2
25	200	0.783 ± 0.145	0.706 ± 0.141	59.3±11.8	102.9±6.8
100	5	0.879 ± 0.106	0.777 ± 0.125	124.4±4.7	87.5±3.7
100	10	0.861 ± 0.110	0.787 ± 0.116	129.3±5.9	98.4±4.0
100	25	0.845 ± 0.131	0.784 ± 0.143	136.4±9.1	102.5±3.8
100	100	0.871 ± 0.130	0.783 ± 0.137	137.0±10.6	102.4±3.8
100	200	0.852 ± 0.139	0.782 ± 0.138	136.1±8.3	102.5±3.7
200	5	0.909 ± 0.101	0.815 ± 0.109	224.0±4.4	87.4±3.0
200	10	0.889 ± 0.104	0.802 ± 0.112	230.3±6.2	98.4±2.8
200	25	0.886 ± 0.094	0.787 ± 0.112	234.5±9.8	101.9±2.7
200	100	0.895 ± 0.093	0.824 ± 0.102	235.7±10.7	102.4±3.1
200	200	0.912 ± 0.087	0.800 ± 0.119	236.2±10.8	102.2±2.7
500	5	0.937 ± 0.069	0.836 ± 0.099	524.1±5.0	87.3±2.1
500	10	0.924 ± 0.087	0.815 ± 0.106	530.5±6.4	98.9±2.0
500	25	0.917 ± 0.094	0.832 ± 0.103	536.4±9.6	102.4±2.2
500	100	0.925 ± 0.081	0.831 ± 0.106	536.0±10.6	102.8±2.0
500	200	0.922 ± 0.084	0.825 ± 0.109	537.3±13.1	102.8±2.1

Table A.62: Mutagenesis$_{All}$, Unique root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.717 ± 0.104	0.697 ± 0.086	42.1±8.3	53.7±13.7
10	10	0.739 ± 0.104	0.715 ± 0.088	68.8±14.4	87.5±20.0
10	25	0.758 ± 0.102	0.739 ± 0.087	125.1±24.9	148.4±22.6
10	100	0.764 ± 0.101	0.730 ± 0.094	303.9±58.9	200.6±12.1
10	200	0.748 ± 0.098	0.723 ± 0.092	481.9±87.4	209.2±12.6
25	5	0.764 ± 0.086	0.723 ± 0.076	59.6±9.3	53.0±12.9
25	10	0.769 ± 0.094	0.724 ± 0.090	88.8±12.8	92.1±19.2
25	25	0.765 ± 0.107	0.737 ± 0.088	152.5±24.8	147.8±16.6
25	100	0.767 ± 0.101	0.741 ± 0.093	332.6±55.8	200.3±11.5
25	200	0.775 ± 0.106	0.735 ± 0.094	494.4±84.8	211.3±10.4
100	5	0.792 ± 0.108	0.741 ± 0.094	134.1±8.6	52.8±7.4
100	10	0.789 ± 0.094	0.743 ± 0.088	163.5±10.9	92.6±10.7
100	25	0.808 ± 0.089	0.761 ± 0.085	246.3±26.2	146.6±13.6
100	100	0.800 ± 0.082	0.744 ± 0.078	508.5±60.2	200.6±8.8
100	200	0.804 ± 0.092	0.759 ± 0.088	735.7±88.5	208.8±7.4
200	5	0.798 ± 0.098	0.753 ± 0.083	233.5±9.0	54.5±5.8
200	10	0.809 ± 0.102	0.767 ± 0.081	265.5±13.6	92.9±8.5
200	25	0.815 ± 0.088	0.771 ± 0.090	345.5±26.2	147.5±13.6
200	100	0.810 ± 0.096	0.757 ± 0.087	608.4±52.1	200.6±7.1
200	200	0.807 ± 0.095	0.756 ± 0.091	886.5±87.3	209.8±6.5
500	5	0.812 ± 0.093	0.759 ± 0.087	533.2±9.4	53.5±3.7
500	10	0.824 ± 0.092	0.777 ± 0.081	565.8±13.6	92.6±6.2
500	25	0.823 ± 0.089	0.778 ± 0.084	640.1±22.0	148.7±7.8
500	100	0.811 ± 0.092	0.764 ± 0.088	905.8±47.3	200.7±7.4
500	200	0.812 ± 0.090	0.760 ± 0.087	1211.8±84.1	210.3±5.9

Table A.63: Mutagenesis$_{RF}$, Unique root, Info leaves

Forest size	MFC	AUC	Accuracy	Rules generated	Tree size
10	5	0.787 ± 0.126	0.747 ± 0.104	40.1±8.1	49.2±13.1
10	10	0.806 ± 0.117	0.768 ± 0.100	64.6±13.6	81.8±15.1
10	25	0.820 ± 0.115	0.802 ± 0.095	116.8±23.0	121.4±15.0
10	100	0.805 ± 0.104	0.785 ± 0.087	271.0±50.3	152.5±10.2
10	200	0.805 ± 0.098	0.794 ± 0.087	412.7±72.5	155.6±11.9
25	5	0.805 ± 0.116	0.767 ± 0.089	58.4±8.7	48.6±11.8
25	10	0.831 ± 0.099	0.797 ± 0.084	87.2±14.3	82.4±12.2
25	25	0.838 ± 0.105	0.801 ± 0.089	145.0±24.0	121.4±13.6
25	100	0.851 ± 0.094	0.799 ± 0.079	302.4±59.8	152.2±12.0
25	200	0.841 ± 0.098	0.801 ± 0.083	439.9±80.5	157.2±10.2
100	5	0.851 ± 0.091	0.789 ± 0.085	133.4±9.0	50.0±6.5
100	10	0.863 ± 0.094	0.808 ± 0.091	165.2±12.5	80.2±8.4
100	25	0.884 ± 0.085	0.820 ± 0.091	230.6±23.2	120.9±10.9
100	100	0.872 ± 0.089	0.816 ± 0.083	470.4±57.5	151.5±6.1
100	200	0.877 ± 0.084	0.826 ± 0.080	687.5±85.0	156.3±5.7
200	5	0.867 ± 0.085	0.800 ± 0.090	232.5±7.3	49.1±5.1
200	10	0.879 ± 0.085	0.815 ± 0.081	265.1±14.5	81.1±7.3
200	25	0.887 ± 0.084	0.831 ± 0.077	332.7±22.7	120.6±8.5
200	100	0.889 ± 0.072	0.837 ± 0.077	585.4±54.0	151.6±6.6
200	200	0.884 ± 0.080	0.819 ± 0.083	824.3±76.9	156.2±6.0
500	5	0.874 ± 0.085	0.808 ± 0.083	533.7±9.0	48.8±3.3
500	10	0.892 ± 0.078	0.825 ± 0.082	563.6±12.3	80.5±5.4
500	25	0.897 ± 0.075	0.836 ± 0.077	631.6±21.2	121.2±5.9
500	100	0.898 ± 0.073	0.828 ± 0.084	879.1±54.7	151.9±5.2
500	200	0.884 ± 0.084	0.826 ± 0.081	1143.9±93.2	157.0±5.3

Table A.64: Unique root, Track leaves, 500 trees

Dataset	MRC	AUC	Accuracy	Rules generated	Tree size
Carcinogenesis	25	0.635 ± 0.082	0.592 ± 0.070	1429.6±113.9	239.3±5.2
Carcinogenesis	50	0.634 ± 0.082	0.595 ± 0.077	2531.8±213.7	246.2±4.5
$\text{Diterpenes}_{52.3}$	25	0.996 ± 0.005	0.974 ± 0.019	972.7±51.5	179.7±11.2
$\text{Diterpenes}_{52.3}$	50	0.997 ± 0.004	0.975 ± 0.018	1331.6±84.4	136.3±10.4
$\text{Diterpenes}_{52.54}$	25	0.994 ± 0.007	0.963 ± 0.023	1074.0±57.1	242.6±20.7
$\text{Diterpenes}_{52.54}$	50	0.995 ± 0.005	0.966 ± 0.021	1507.5±92.3	180.3±11.9
$\text{Diterpenes}_{54.3}$	25	0.998 ± 0.003	0.983 ± 0.017	920.0±49.3	127.2±9.7
$\text{Diterpenes}_{54.3}$	50	0.999 ± 0.002	0.985 ± 0.015	1232.8±74.9	95.6±8.2
Musk_1	25	0.958 ± 0.059	0.866 ± 0.089	806.3±35.4	39.1±1.6
Musk_1	50	0.958 ± 0.052	0.865 ± 0.106	1055.0±56.4	32.5±1.2
Mutagenesis_{All}	25	0.830 ± 0.088	0.791 ± 0.085	1109.0±67.2	120.1±5.2
Mutagenesis_{All}	50	0.830 ± 0.091	0.780 ± 0.085	1836.8±123.1	126.7±4.9
Mutagenesis_{RF}	25	0.902 ± 0.073	0.850 ± 0.076	1028.0±64.1	88.9±4.0
Mutagenesis_{RF}	50	0.905 ± 0.070	0.850 ± 0.077	1566.2±111.7	86.0±4.1

Table A.65: Bagging root, Track leaves, 500 trees

Dataset	MRC	AUC	Accuracy	Rules generated	Tree size
Carcinogenesis	50	0.646 ± 0.084	0.612 ± 0.074	2157.7±158.1	163.5±2.5
$\text{Diterpenes}_{52.3}$	50	0.996 ± 0.005	0.973 ± 0.019	1245.7±75.5	104.1±7.5
$\text{Diterpenes}_{52.54}$	50	0.994 ± 0.006	0.962 ± 0.024	1442.2±105.5	136.5±10.0
$\text{Diterpenes}_{54.3}$	50	0.999 ± 0.002	0.984 ± 0.015	1175.6±67.6	75.5±5.1
Musk_1	50	0.945 ± 0.070	0.853 ± 0.108	970.0±43.7	24.3±0.6
Mutagenesis_{All}	50	0.839 ± 0.088	0.793 ± 0.080	1669.8±112.9	89.3±2.7
Mutagenesis_{RF}	50	0.906 ± 0.068	0.860 ± 0.078	1454.2±87.9	61.7±2.5

Bibliography

[1] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In *Neural Information Processing Systems*, pages 561–568, 2002.

[2] A. Van Assche. *Improving the Applicability of Ensemble Methods in Data Mining*. PhD thesis, Katholieke Universiteit Leuven, January 2008.

[3] A. Van Assche, C. Vens, H. Blockeel, and S. Džeroski. First order random forests: Learning relational classifiers with complex aggregates. *Machine Learning*, 64(1-3):149–182, 2006.

[4] P. Berka. Guide to the financial data set. In A. Siebes and P. Berka, editors, *The ECML/PKDD 2000 Discovery Challenge.*, 2000.

[5] H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[6] H. Blockeel, L. De Raedt, N. Jacobs, and B.Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3(1):59–93, 1999.

[7] H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In J. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning*, pages 55–63. Morgan Kaufmann, 1998.

[8] A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[9] I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[10] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[11] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[12] R. Caruana and R. Niculescu-mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning (ICML06*, pages 161–168, 2006.

[13] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[14] Y. Chevaleyre, N. Bredeche, and J.-D. Zucker. Learning rules from multiple instance data : Issues and algorithms. In *Proceedings of the 9th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU02)*, 2002.

[15] Y. Chevaleyre and J.-D. Zucker. A framework for learning rules from multiple instance data. In *12th European Conference on Machine Learning*, volume 2167 of *LNCS*, pages 49–60. Springer, 2001.

[16] I. Davidson and A. Satyanarayana. Speeding up k-means clustering by bootstrap averaging. In *Proceedings of the Third IEEE International Conference on Data Mining, Workshop on Clustering Large Data Sets*, pages 16–25, 2003.

[17] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991.

[18] A. P. Dempster, N. M. Laid, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[19] T. G. Dietterich. Ensemble methods in machine learning. In *MCS '00: Proceedings of the First International Workshop on Multiple Classifier Systems*, pages 1–15, London, UK, 2000. Springer-Verlag.

[20] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Perez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[21] I. Dutra, D. Page, V. Costa, and J. Shavlik. An empirical evaluation of bagging in inductive logic programming. In *Proceedings of the Twelfth International Conference on Inductive Logic Programming*. Springer-Verlag, 2002.

[22] S. Džeroski. Multi-relational data mining: an introduction. *SIGKDD Exploration Newsletter*, 5(1):1–16, 2003.

[23] S. Džeroski, S. Schulze-Kremer, K. R. Heidtke, K. Siems, and D. Wettschereck. Applying ILP to diterpene structure elucidation from 13C NMR spectra. In *ILP '96: Selected Papers from the 6th International Workshop on Inductive Logic Programming*, pages 41–54, London, UK, 1997. Springer-Verlag.

[24] W. Emde and D. Wettschereck. Relational instance based learning. In Lorenza Saitta, editor, *Machine Learning - Proceedings 13th International Conference on Machine Learning*, pages 122 – 130. Morgan Kaufmann Publishers, 1996.

[25] V. Faber. Clustering and the continuous k-means algorithm. *Los Alamos Science*, 22:138–144, 1994.

[26] P. A. Flach and N. Lachiche. Naïve Bayesian classification of structured data. *Machine Learning*, 57(3):233–269, 2004.

[27] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT '95: Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, UK, 1995. Springer-Verlag.

[28] T. Gärtner, J. W. Lloyd, and P. A. Flach. Kernels and distances for structured data. *Machine Learning*, 57(3):205–232, 2004.

[29] T. B. Ho, S. Kawasaki, and J. Granat. Creative environments. In Wierzbicki and Nakamori, editors, *Knowledge Acquisition by Machine Learning and Data Mining*, chapter 4, pages 69–91. Springer-Verlag, 2007.

[30] T. K. Ho. Random decision forests. In *ICDAR '95: Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1)*, page 278, Washington, DC, USA, 1995. IEEE Computer Society.

[31] S. Hoche and S. Wrobel. A comparative evaluation of feature set evolution strategies for multirelational boosting. In *Proceedings of the 13th International Conference on Inductive Logic Programming*. Springer-Verlag, 2003.

[32] T. Horváth, S. Wrobel, and U. Bohnebeck. Relational instance-based learning with lists and terms. *Machine Learning*, 43(1-2):53–80, 2001.

[33] A. Hutchinson. Metrics on terms and clauses. In *ECML '97: Proceedings of the 9th European Conference on Machine Learning*, pages 138–145, London, UK, 1997. Springer-Verlag.

[34] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[35] D. Jensen and J. Neville. Linkage and autocorrelation cause feature selection bias in relational learning. In *Proceedings of the 19th International Conference on Machine Learning*, pages 259–266. Morgan Kaufmann, 2002.

[36] R. D. King and A. Srinivasan. Prediction of rodent carcinogenicity bioassays from molecular structure using inductive logic programming. *Environmental Health Perspectives*, 104(5):1031–1040, 1996.

[37] M. Kirsten and S. Wrobel. Relational distance-based clustering. In Fritz Wysotzki, Peter Geibel, and Christina Schädler, editors, *Proc. Fachgruppentreffen Maschinelles Lernen (FGML-98)*, pages 119 – 124, 10587 Berlin, 1998. Technical University of Berlin, Technischer Bericht 98/11.

[38] E. M. Kleinberg. On the algorithmic implementation of stochastic discrimination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(5):473–490, 2000.

[39] P. Komarek and A. Moore. Fast robust logistic regression for large sparse datasets with binary outputs. In C. M. Bishop and B. J. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics, 2003.

[40] S. Kramer, N. Lavrač, and P. Flach. Propositionalization approaches to relational data mining. pages 262–286. Springer-Verlag New York, Inc., New York, NY, USA, 2000.

[41] M. Krogel, S. Rawles, F. Železný, P. Flach, N. Lavrač, and S. Wrobel. Comparative evaluation of approaches to propositionalization. In *Proceedings of the 13th Int. Conference on Inductive Logic Programming*. Springer-Verlag, 2003.

[42] M.-A. Krogel and T. Scheffer. Multi-relational learning, text mining, and semi-supervised learning for functional genomics. *Machine Learning*, 57(1-2):61–81, 2004.

[43] M. A. Krogel and S. Wrobel. Transformation-based learning using multirelational aggregation. In *ILP '01: Proceedings of the 11th International Conference on Inductive Logic Programming*, pages 142–155, London, UK, 2001. Springer-Verlag.

[44] N. Landwehr, K. Kersting, and L. De Raedt. Integrating naïve Bayes and FOIL. *Machine Learning*, 8:481–507, 2007.

[45] N. Landwehr, A. Passerini, L. De Raedt, and P. Frasconi. kfoil: Learning simple relational kernels. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*. AAAI Press, 2006.

[46] J. Larson and R. S. Michalski. Inductive inference of VL decision rules. *SIGART Bulletin*, (63):38–44, 1977.

[47] S. le Cessie and J.C. van Houwelingen. Ridge estimators in logistic regression. *Applied Statistics*, 41(1):191–201, 1992.

[48] P. Ling, Z. Wang, and C. Zhou. A spectrum-based support vector algorithm for relational data semi-supervised classification. In *ICONIP (1)*, pages 801–810, 2006.

[49] H. Lodhi and S. Muggleton. Is mutagenesis still challenging? In *ILP-05 Late-Breaking Papers*, pages 35–40, 2005.

[50] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathemtical Statistics and Probability*, pages 281–297, 1967.

[51] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[52] A. Mpagouli and I. Hatzilygeroudis. Converting first order logic into natural language: A first level approach. In Theodore S. Papatheodorou, Dimitris N. Christodoulakis, and Nikitas N. Karanikolas, editors, *Current Trends in Informatics: 11th Panhellenic Conference on Informatics, PCI 2007*, volume A, pages 517–526. New Technologies Publications, 2007.

[53] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[54] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[55] C. Nadeau and Y. Bengio. Inference for the generalization error. *Machine Learning*, 52(3):239–281, 2003.

[56] S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, chapter 9, pages 162–177. Springer, 1997.

[57] C. W. Olofson. Competitive analysis - worldwide RDBMS 2006 vendor shares. IDC survey, April 2007.

[58] M. J. Pazzani and D. F. Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9:57–94, 1992.

[59] B. Pfahringer, C. Leschi, and P. Reutemann. Scaling up semi-supervised learning: An efficient and effective LLGC variant. In Zhi-Hua Zhou, Hang Li, and Qiang Yang, editors, *PAKDD*, volume 4426 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 2007.

[60] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

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

[62] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[63] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[64] J. Ross Quinlan and R. Mike Cameron-Jones. Induction of logic programs: Foil and related systems. *New Generation Computing*, 13(3&4):287–312, 1995.

[65] L. De Raedt. Attribute-value learning versus inductive logic programming: The missing links (extended abstract). In *ILP '98: Proceedings of the 8th International Workshop on Inductive Logic Programming*, pages 1–8, London, UK, 1998. Springer-Verlag.

[66] S. Ray and M. Craven. Supervised versus multiple instance learning: an empirical comparison. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 697–704, New York, NY, USA, 2005. ACM.

[67] G. Riccardi. *Principles of Database Systems with Internet and Java Applications*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[68] G. Ridgeway, D. Madigan, and T. Richardson. Interpretable boosted Naïve Bayes classification. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 101–104, 1998.

[69] K. Ross, D. Ashwin, and S. Dehaspe. Warmr: A data mining tool for chemical data. *Journal of Computer Aided Molecular Design*, 15:173–181, 2001.

[70] P. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(1):53–65, 1987.

[71] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(2):210–229, April 1959. Reprinted in E. A. Feigenbaum and J. Feldman (Eds.) 1963, *Computers and Thought*, McGraw-Hill, New York.

[72] D. E. Smith and M. R. Genesereth. Ordering conjunctive queries. *Artificial Intelligence*, 26(2):171–215, 1985.

[73] A. Srinivasan. Aleph manual. URL: `http://web2.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph`, 2007.

[74] A. Srinivasan, R. King, and S. Muggleton. The role of background knowledge: using a problem from chemistry to examine the performance of an ILP program. Technical report, Oxford University, Oxford, 1999.

[75] A. Srinivasan, R. D. King, S. Muggleton, and M. J. E. Sternberg. Carcinogenesis predictions using ILP. In Sašo Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297, pages 273–287. Springer-Verlag, 1997.

[76] A. Srinivasan, S. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237, pages 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[77] K. M. Ting and B. T. Low. Model combination in the multiple-data-batches scenario. In *Proceedings of the 9th European Conference on Machine Learning*, pages 250–265, 1997.

[78] V. Tresp. Scaling kernel-based systems to large data sets. *Data Mining and Knowledge Discovery*, 5(3):197–211, 2001.

[79] A. M. Turing. Computing Machinery and Intelligence. *Mind*, 59(236):433–460, 1950.

[80] F. Železný and N. Lavrač. Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, 62(1-2):33–63, 2006.

[81] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2nd edition, 2005.

[82] A. Woznica, A. Kalousis, and M. Hilario. Kernels over relational algebra structures. In Tu Bao Ho, David Wai-Lok Cheung, and Huan Liu, editors, *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, 2005, Proceedings*, volume 3518 of *Lecture Notes in Computer Science*, pages 588–598. Springer, 2005.

[83] A. Woznica, A. Kalousis, and M. Hilario. Distances and (indefinite) kernels for sets of objects. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 1151–1156, Washington, DC, USA, 2006. IEEE Computer Society.

[84] Q. Zhang and S. Goldman. EM-DD: An improved multiple-instance learning technique. In *Neural Information Processing Systems*, volume 14, 2001.

[85] D. Zhou, J. Huang, and B. Schölkopf. Learning from labeled and unlabeled data on a directed graph. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 1036–1043, New York, NY, USA, 2005. ACM.