Data Mining Algorithms for Classification

BSc Thesis Artificial Intelligence
Author: Patrick Ozer
Radboud University Nijmegen

January 2008

Supervisor:
Dr. I.G. Sprinkhuizen-Kuyper
Radboud University Nijmegen

Abstract

Data Mining is a technique used in various domains to give meaning to the available data. In classification tree modeling the data is classified to make predictions about new data. Using old data to predict new data has the danger of being too fitted on the old data. But that problem can be solved by pruning methods which degeneralizes the modeled tree. This paper describes the use of classification trees and shows two methods of pruning them. An experiment has been set up using different kinds of classification tree algorithms with different pruning methods to test the performance of the algorithms and pruning methods. This paper also analyzes data set properties to find relations between them and the classification algorithms and pruning methods.

1 Introduction

The last few years Data Mining has become more and more popular. Together with the information age, the digital revolution made it necessary to use some heuristics to be able to analyze the large amount of data that has become available. Data Mining has especially become popular in the fields of forensic science, fraud analysis and healthcare, for it reduces costs in time and money.

One of the definitions of Data Mining is; “Data Mining is a process that consists of applying data analysis and discovery algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns (or models) over the data” [4]. Another , sort of pseudo definition; “The induction of understandable models and patterns from databases” [6]. In other words, we initially have a large (possibly infinite) collection of possible models (patterns) and (finite) data. Data Mining should result in those models that describe the data best, the models that fit (part of the data).

Classification trees are used for the kind of Data Mining problem which are concerned with prediction. Using examples of cases it is possible to construct a model that is able to predict the class of new examples using the attributes of those examples. An experiment has been set up to test the performance of two pruning methods which are used in classification tree modeling. The pruning methods will be applied on different types of data sets. The remainder of this thesis is as follows. Section 2 explains classification trees and different pruning methods. In Section 3 our experiments are described. The results are given in Section 4 and we end with a discussion in Section 5.

2 Classification Trees

In Data Mining one of the most common tasks is to build models for the prediction of the *class* of an *object* on the basis of its *attributes* [8]. Here the object can be seen as a customer, patient, transaction, e-mail message or even a single character. Attributes of such objects can be, for example for the patient object, hearth rate, blood pressure, weight and gender, whereas the class of the patient object would most commonly be positive/negative for a certain disease.

This section considers the problem of learning a classification tree model using the data we have about such objects. In Section 2.1 a simple example will be presented. In Section 2.2 the steps of constructing such a model will be discussed in more detail.

2.1 Example

An example will be walked through to give a global impression of Classification Trees. The example concerns the classification of a credit scoring data set and is obtained from [5]. To be able to walk through a whole classification tree we only consider a part of the data. Figure 1 shows the tree that is constructed from the data in Table 1.

The objects in this data set are loan applicants. The class values *good* and *bad* correspond to respectively accepted and denied for getting a loan for a loan applicant with the same attribute values as someone who already tried to get a loan in the past. The data set contains five attributes; *age*, *ownhouse?*, *married?*, *income* and *gender*. With this information we can make a classification model of the applicants who were accepted or denied for a loan. The model derived from that classification can then predict if a new applicant can get a loan according to his/her attributes.

As can be seen in Table 1 the instances each have a value for all the attributes and the class label. When we construct the tree structure, we

Record	*age*	*ownhouse?*	*married?*	*income*	*gender*	*class*
1	22	no	no	28,000	male	bad
2	46	no	yes	32,000	female	bad
3	24	yes	yes	24,000	male	bad
4	25	no	no	27,000	male	bad
5	29	yes	yes	32,000	female	bad
6	45	yes	yes	30,000	female	good
7	63	yes	yes	58,000	male	good
8	36	yes	no	52,000	male	good
9	23	no	yes	40,000	female	good
10	50	yes	yes	28,000	female	good

Table 1: Credit scoring data [5]

will try to split the data in such manner that it will result in the best *split*. The splitting is done according to the attributes. The quality of a split is computed from the correctly classified number of cases in each of the resulting nodes of that split. Finding the best split continues until all nodes are *leaf* nodes or when no more splits are possible (a *leaf* node is a node which contains cases of a single class only).

In computer science, tree structures can have binary or n-ary branches. This is the same for classification trees. However, most of the times a tree structure in classification trees will have binary branches, because splitting in two ways will result in a better separation of the data. When we present the impurity measures for the splitting criteria we will consider the possibility for n-ary splits. In the examples given later on, binary splits will be used because those are easier to explain.

Now we will focus on classifying new applicants according to a model generated from the data in Table 1. When a new applicant arrives he/she is “dropped down” the tree until we arrive in a leaf node, where we assign the associated class to the applicant. Suppose an applicant arrives and fills in the following information on the application form:

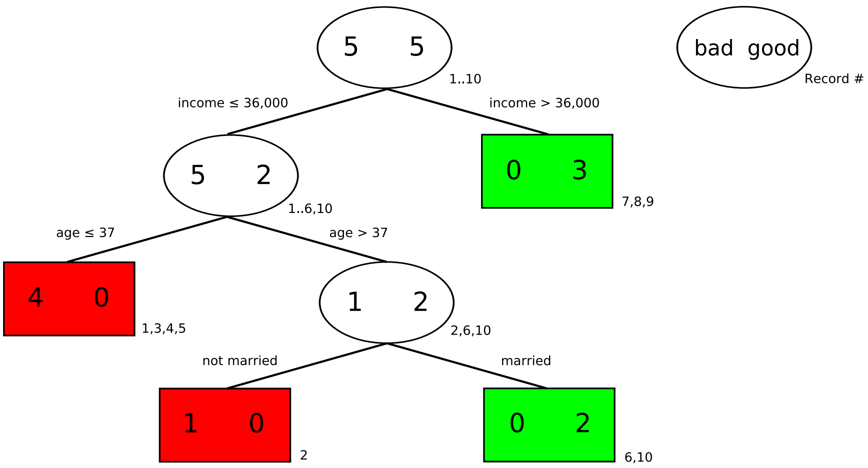


Figure 1: Tree built for loan applicants [5]

age: 42, married?: no, own house?: yes, income: 30,000, gender: male

Then we can say, according to the model shown in Figure 1, that this applicant will be rejected for the loan: starting from the root of the tree, we first perform the associated test on income. Since the value of income for this applicant is below 36,000 we go to the left and we arrive at a new test on age. Age is above 37, thus we go to the right. Here we arrive at the final test on marital status. Since the applicant is not married he is sent to the left and we arrive at a leaf node with class label bad. We predict that the applicant will not qualify for a loan and therefore is rejected.

Besides this example, there are numerous domains in which these kinds of predictions can be done. Classification trees are favoured as Data Mining tool because they are easy to interpret, they select attributes automatically and they are able to handle numeric as well as categorical values.

2.2 Building Classification Trees

In the previous section we saw that the construction of a classification tree starts with performing good splits on the data. In this section we define what such a good split is and how we can find such a split.

Three impurity measures, resubstitution-error, gini-index and the entropy, for splitting data will be discussed in Section 2.2.1. The actual splitting and tree construction according to these splits will be explained in Sections 2.2.2 & 2.2.3. And finally in Section 2.2.4 the need for pruning will be explained accompanied by two well known pruning methods, cost-complexity pruning and reduced-error pruning.

2.2.1 Impurity Measures

As shown in the example in the previous section we should strive towards a split that will separate the data as much as possible in accordance with the class labels. So the objective is to obtain nodes that contain cases of a single class only as mentioned before. We define *impurity* as a function of the relative frequencies of the classes in that node:

$$i(t) = \phi(p_1, p_2, ..., p_J) \tag{1}$$

with p_j $(j = 1, ..., J)$ as the relative frequencies of the J different classes in that node [1].

To compare all the possible splits of the data you have, a quality of a split as the reduction of impurity that the split achieves must be defined. In the example later on the following equation for the impurity reduction of split s in node t will be used:

$$\Delta i(s,t) = i(t) - \sum_j \pi(j) i(j) \tag{2}$$

where $\pi(j)$ is the proportion of cases sent to branch j by s, and $i(j)$ is the impurity of the node of branch j. Because different algorithms of tree construction use different impurity measures, we will discuss three of them and give a general example later on.

Resubstitution error This is a measure for the impuroty defined by the fraction of the cases in a node that is classified incorrectly if we assign every case to the majority class in that node:

$$i(t) = 1 - \max_j p(j|t) \tag{3}$$

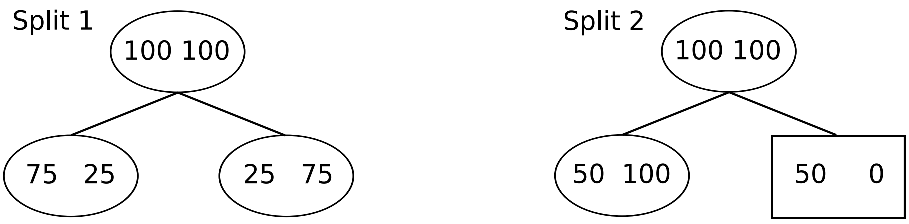
where $p(j|t)$ is the relative frequency of class j in node t. The resubstitution error gives a score to a split according to the incorrectly classified cases in a node. It can recognize a better split if it has less errors in that node. But the resubstitution error has one major disadvantage: it does not recognize a split as a better one if one of its resulting nodes is pure. So it does not prefer Split 2 over Split 1 in Figure 2. In such a case we want a split with a pure node to be preferred.

Gini index The Gini index does recognize such a split. Its impurity measure is defined as follows:

$$i(t) = \sum_j p(j|t)(1 - p(j|t)) \tag{4}$$

The Gini index prefers Split 2 over Split 1 in Figure 2, since it results in the highest impurity reduction.

Entropy Finally we have the entropy measure which is used in well-known classification tree algorithms like ID3 and C4.5. The advantage of the entropy measure over the gini-index is that it will reach a minimum faster if more instances of the child nodes belong to the same class. The entropy measure is defined as:



Resubstitution error: $\Delta(s_1,t) = \frac{1}{2} - (\frac{1}{2}\cdot\frac{1}{4}) - (\frac{1}{2}\cdot\frac{1}{4}) = \frac{1}{4}$

Gini index: $\Delta(s_1,t) = \frac{1}{2} - (\frac{1}{2}\cdot\frac{3}{8}) - (\frac{1}{2}\cdot\frac{3}{8}) = \frac{1}{8}$

Resubstitution error: $\Delta(s_2,t) = \frac{1}{2} - (\frac{3}{4}\cdot\frac{1}{3}) - (\frac{1}{4}\cdot 0) = \frac{1}{4}$

Gini index: $\Delta(s_2,t) = \frac{1}{2} - (\frac{3}{4}\cdot\frac{4}{9}) - (\frac{1}{4}\cdot 0) = \frac{1}{6}$

Figure 2: According to resubstitution error these splits are equally good for the resubstitution error. The Gini index prefers the second split.

$$i(t) = -\sum_j p(j|t)\log(p(j|t) \tag{5}$$

For all three measures, it holds that they reach a maximum when all classes are evenly distributed in the nodes and they will be at a minimum if all instances in the nodes belong to one class. To show the differences in how fast the measures reach their minimum, Figure 3 is given.

2.2.2 Splits to consider

We have looked at different impurity criteria for computing the quality of a split. In this section we look at which splits are considered and how we select the best split (for binairy splits only). The attributes can have numerical or categorical values. In the case of numerical values, all the values of the attributes occurring in the training set are considered. The possible splits are made between two consecutive numerical values occuring in the training set. If the attribute is categorical with N categories, then $2^{N-1}-1$ splits are considered. There are 2^N-2 non-empty proper subsets of a set of N elements. The empty set and the complete set do not count. Furthermore a split of the N categories into S and S^c, the complement of S, is the same split as the

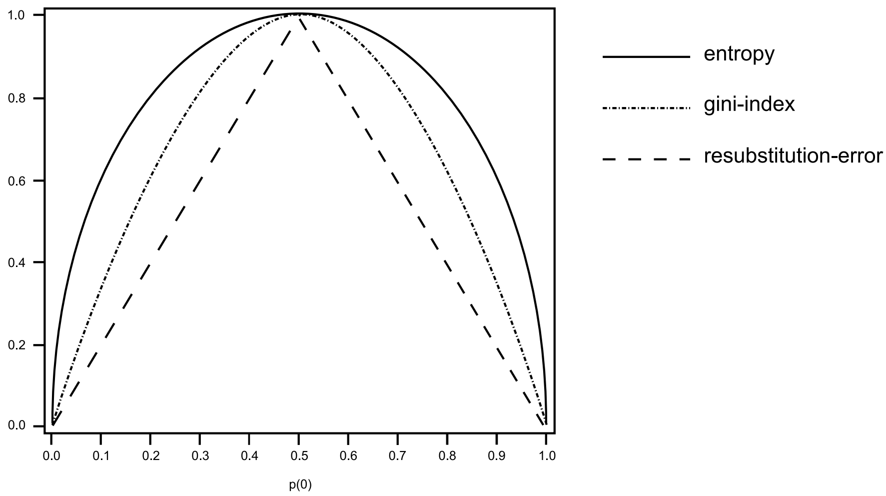


Figure 3: Graphs of entropy, Gini index and resubstitution error for a two-class problem [5]

split into S^c and S. Then we can count $\frac{1}{2}(2^N - 2) = 2^{-1}2^N - 1 = 2^{N-1} - 1$ different splits.

Example To compute the best split on income, a numerical attribute, the impurity reduction is computed for all possible values of income. In Table 2 the data are ordered on the value of income. Using the Gini index with the split after the first value on income we get the following computation with Equation 4 filled into Equation 2:

$$\Delta i(s,t) = \frac{1}{2} - \frac{1}{10} \cdot 0 - \frac{9}{10} \cdot 2 \cdot (\frac{4}{9})(\frac{5}{9}) = \frac{1}{18} \tag{6}$$

First the impurity of the parent node is computed. Then the impurity of each branch node is multiplied with the proportion of cases send to each corresponding direction, to subtract this from the impurity of the parent node. For the rest of the computations, see Table 2.

income	*class*	*impurity reduction of split : income*
24,000	B	$(1/2) - 1/10 \cdot 0 - 9/10 \cdot 2 \cdot (4/9)(5/9) = 0.056$
27,000	B	$(1/2) - 2/10 \cdot 0 - 8/10 \cdot 2 \cdot (3/8)(5/8) = 0.125$
28,000	B,G	$(1/2) - 4/10 \cdot 2 \cdot (1/4)(3/4) - 6/10 \cdot 2 \cdot (4/6)(2/6) = 0.083$
30,000	G	$(1/2) - 5/10 \cdot 2 \cdot (2/5)(3/5) - 5/10 \cdot 2 \cdot (3/5)(2/5) = 0.02$
32,000	B,B	$(1/2) - 7/10 \cdot 2 \cdot (2/7)(5/7) - 3/10 \cdot 2 \cdot 0 = 0.214$
40,000	G	$(1/2) - 8/10 \cdot 2 \cdot (3/8)(5/8) - 2/10 \cdot 0 = 0.125$
52,000	G	$(1/2) - 9/10 \cdot 2 \cdot (4/9)(5/9) - 1/10 \cdot 0 = 0.056$
58,000	G	

Table 2: Computation of the impurity reduction for the Gini index for the splits on income [5]

2.2.3 Tree construction

Building a classification tree starts at the top of the tree with all the data. For all the attributes the best split of the data must be computed. Then the best splits for each of the attributes are compared. The attribute with the best split wins. The split will be executed on the attribute with the best value of the best split (again we consider binary trees). The data is now separated to the corresponding branches and from here the computation on the rest of the nodes will continue in the same manner. Tree construction will finish when there is no more data to separate or no more attributes to sperate them by.

2.2.4 Overfitting and Pruning

If possible we continue splitting until all leaf nodes of the tree contain examples of a single class. But unless the problem is deterministic, this will not result in a good tree for prediction. We call this overfitting. The tree will be focused too much on the training data. To prevent overfitting we can use *stopping rules*; stop expanding nodes if the impurity reduction of the best split is below some threshold. A major disadvantage of stopping rules is that sometimes, first a weak (not weaker) split is needed to be able to follow up with a good split. This can be seen in building a tree for the XOR problem [**?**]practDM. Another solution is pruning. First grow a maximum-size tree on the training sample and then prune this large tree. The objective is to select the pruned subtree that has the lowest true error rate. The problem is, how to find this pruned subtree?

There are two pruning methods we will use in the tests, cost-complexity pruning [1] and [5] and reduced-error pruning [3]. In the next two paragraphs we will explain how the two pruning methods work and finish with a concrete example of the pruning process.

Cost-complexity pruning The basic idea of cost-complexity pruning is not to consider all pruned subtrees, but only those that are the "best of their kind" in a sense to be defined below. Let $R(T)$ (T stands for the complete tree) denote the fraction of cases in the training sample that is misclassified by the tree T ($R(T)$ is the weighted summed error of the leafs of tree T). Total cost $C_\alpha(T)$ of tree T is defined as:

$$C_\alpha(T) = R(T) + \alpha|\tilde{T}| \tag{7}$$

The total cost of tree T then consists of two components: summed error of the leafs $R(T)$, and a penalty for the complexity of the tree $\alpha|\tilde{T}|$. In this expression $\tilde{T}$ stands for the set of leaf nodes of T, $|\tilde{T}|$ the number of leaf nodes and α is the parameter that determines the complexity penalty: when the number of leaf nodes increases by one (one additional split in a binary tree), then the total cost (if R remains equal) increases with α [5]. The value of α can make a complex tree with no errors have a higher total cost than a small tree making a number of errors. For every value of α there is a smallest minimizing subtree. We state the complete tree by T_{max}. For a fixed value of α there is a unique smallest minimizing subtree $T(\alpha)$ of T_{max} that fulfills the following conditions, proven in [1]:

1. $C_\alpha(T(\alpha)) = min_{T \subseteq T_{max}} C_\alpha(T)$
2. If $C_\alpha(T) = C_\alpha(T(\alpha))$ then $T(\alpha) \subseteq T$

These two conditions say the following. The first says that there is no subtree of T_{max} with lower cost than $T(\alpha)$ at that α value. And the second says that if more than one tree achieves the same minimum, we select the smallest tree. Since T_{max} is finite, there is a finite number of different subtrees $T(\alpha)$ of T_{max}. A decreasing sequence of α values for subtrees of T_{max} would then look like

$$T_1 \supset T_2 \supset T_3 \supset ... \supset \{t_1\}$$

with t_1 as the root node of T and T_n is the smallest minimizing subtree for $\alpha \in [\alpha_n, \alpha_{n+1})$. This means that the next tree can be computed by pruning the current one, which will be shown in the example later on.

Reduced-error pruning At each node in a tree it is possible to establish the number of instances that are misclassified on a training set by propagating errors upwards from leaf nodes. This can be compared to the error rate if the node was replaced by the most common class resulting from that node. If the difference is a reduction in error, then the subtree below the node can be considered for pruning. This calculation is performed for all nodes in the tree and the one which has the highest reduced-error rate is pruned. The procedure is then iterated over the freshly pruned tree until there is no possible reduction in error rate at any node. The error is computed by using a pruning set, a part of the test set. This has the disadvantage of needing larger amounts of data, but the advantage of resulting in a more accurate classification tree [3].

Example *cost-complexity* pruning We will give an example of how to find the pruning sequence with the tree T_1 given in Figure 4. We are looking

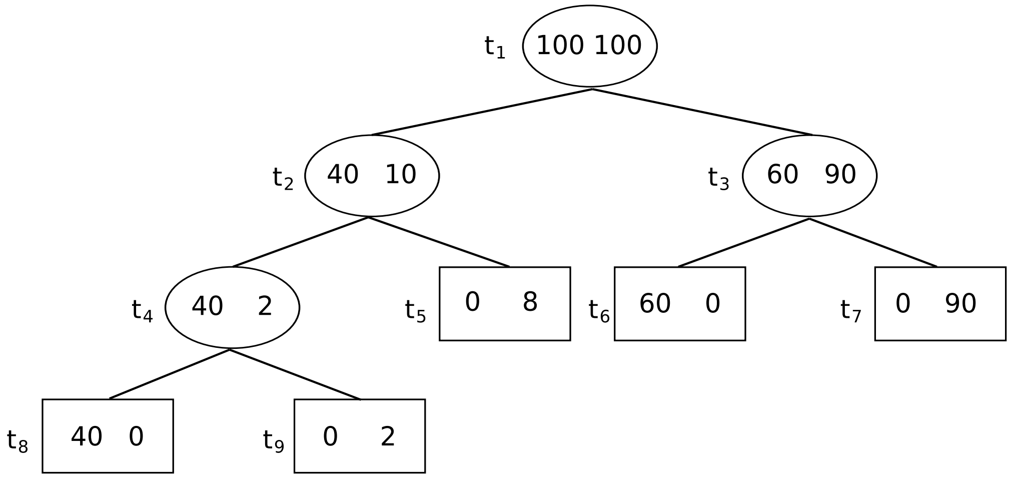


Figure 4: Complete tree: T_1

for the value of α where $T_1 - T_t$ becomes better than T_1. In [1] it is shown that using

$$\alpha = \frac{R(t) - R(T_t)}{(|\tilde{T_t}| - 1)} \tag{8}$$

the sequence of α values can be computed. In Equation 8 T_t stands for the branch of T with root node t. By calculating the α value for each node in tree T_1, it can be determined which node should be pruned. This process is continued until there are no more nodes to prune. As stated, we first calculate the corresponding α values for each node in the tree. For the complete tree T_1 this is:

$$\alpha(T_1(t_1)) = \frac{\frac{1}{2} - 0}{5 - 1} = \frac{1}{80} \tag{9}$$

$$\alpha(T_1(t_2)) = \frac{\frac{1}{20} - 0}{3 - 1} = \frac{1}{40} \tag{10}$$

$$\alpha(T_1(t_3)) = \frac{\frac{60}{200} - 0}{2 - 1} = \frac{3}{10} \tag{11}$$

$$\alpha(T_1(t_4)) = \frac{\frac{2}{200} - 0}{2 - 1} = \frac{1}{100} \tag{12}$$

The node in t_4 has the lowest alpha value, so we prune the tree below that node. This results in the tree $T_2 : T_1(t_4)$, seen in Figure 5. We use $T_1(t_i)$ for: T_1 is pruned below t_i. In tree T_2 the node in t_2 has the lowest alpha value, so

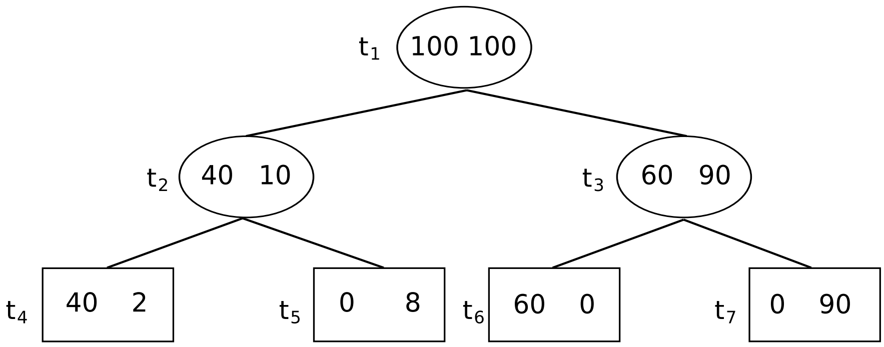


Figure 5: The tree T_2, obtained by pruning T_1 below node t_4

now we prune the tree in that node. This results in the tree $T_3 : T_1(t_2)$, see Figure 6. Now the node in t_1 has the lowest alpha value, so now we prune

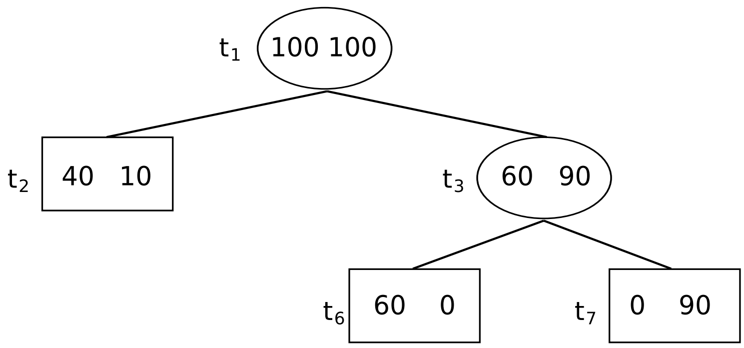


Figure 6: The tree T_3, obtained by pruning T_1 below node t_2

the tree in that node. This results in the tree $T_4 : T_1(t_1)$, see Figure 7.

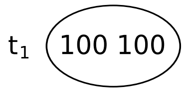


Figure 7: The tree T_4, obtained by pruning T_1 below node t_1

3 Experiment

This section describes our experiments investigating how the different pruning methods and properties of the data sets influence the results. To this end we selected some characteristic data sets from the UCI Machine Learning repository [2] to test the algorithms with. Besides checking the performance on the datasets we are also interested in whether there are properties of those data sets that influence pruning method and algorithm performance performance. To run the experiments, the Data Mining tool WEKA [9] was used. In Section 3.1 a global description of WEKA is given. The data sets are discussed in Section 3.2. The section will end with the experiment description and an overview of the used classification tree algorithms.

3.1 WEKA

WEKA is a collection of machine learning algorithms for Data Mining tasks. It contains tools for data preprocessing, classification, regression, clustering, association rules, and visualization [9]. For our purpose the classification tools were used. There was no preprocessing of the data.
WEKA has four different modes to work in.

- Simple CLI; provides a simple command-line interface that allows direct execution of WEKA commands.
- Explorer; an environment for exploring data with WEKA.
- Experimenter; an environment for performing experiments and conduction of statistical tests between learning schemes.

- Knowledge Flow; presents a “data-flow” inspired interface to WEKA. The user can select WEKA components from a tool bar, place them on a layout canvas and connect them together in order to form a “knowledge flow” for processing and analyzing data.

For most of the tests, which will be explained in more detail later, the explorer mode of WEKA is used. But because of the size of some data sets, there was not enough memory to run all the tests this way. Therefore the tests for the larger data sets were executed in the simple CLI mode to save working memory.

3.2 Data sets

The data sets used for the tests come from the UCI Machine Learning repository [2]. We are dealing with classification tasks, thus we have selected sets of which the class values are nominal [5], [9]. Selection of the sets further depended on their size, larger data sets generally means higher confidence. We choose different kinds of data sets, because we also wanted to test if the performance of an algorithm depended on the kind of set that is used. For instance, is the performance of an algorithm better on sets in different categories, like medical or recognition databases? Is performance influenced by the attribute type, numerical or nominal? Or can we find other properties of data sets that influence performance?

For the WEKA tool the data sets need to be in the ARFF format. An example ARFF files for the credit scoring data is given in Figure 8. Lines beginning with a % sign are comments. Following the comments at the beginning of the file are the name of the *relation* (credit scoring) and a block defining the *attributes* (age, ownhouse?, married?, income, gender), all preceded with an “@”. Nominal attributes are followed by the set of values they can take, enclosed in curly braces. Numeric values are followed by the keyword *numeric*.

```
% ARFF file for the credit scoring data with
% numeric and nominal attributes

@relation creditscoring

@attribute age numeric
@attribute ownhouse? {yes, no}
@attribute married? {yes, no}
@attribute income numeric
@attribute gender {male, female}
@attribute loan? {bad, good}

% 10 attributes

@data
22, no, no, 28.000, male, bad
46, no, yes, 32.000, female, bad
24, yes, yes, 24.000, male, bad
25, no, no, 27.000, male, bad
29, yes, yes, 32.000, female, bad
45, yes, yes, 30.000, female, good
63, yes, yes, 58.000, male, good
36, yes, no, 52.000, male, good
23, no, yes, 40.000, female, good
50, yes, yes, 28.000, female, good
```

Figure 8: ARFF data file for credit scoring data set.

3.3 Experiment description

For the tests we selected fifteen data sets; Arrhythmia, Cylinder-band, Hypothyroid, Kr-vs-Kp, Credit-g, Letter, Mushroom, Nursery, OptiDigits, Page-block, Segment, Sick, Spambase and Waveform5000. All of these data sets have their own properties like the domain of the data set, the kind of attributes it contains, and tree size after training. All of these properties were recorded in Table 3 and 4 .

Data set	**Domain**	**Inst.**	**Attrib.**	**Num.**	**Cat.**	**Classes**
Arrythmia	Medical	452	279	206	73	16
Cylinder-band		512	40	20	20	2
Thyroid disease	Medical	3772	30	7	23	4
Chess game	Game	3196	36	0	36	2
German Credit	Finance	1000	20	7	13	2
Letter	Recognition	20000	16	16	0	26
Mushroom	Nature	8124	22	0	22	2
Nursery	Medical	12960	8	0	8	5
Optical Digits	Recognition	5620	64	64	0	10
Page Blocks		5473	10	10	0	5
Segment	Recognition	2310	19	19	0	7
Sick	Medical	3772	30	7	23	2
Spam	Recognition	4601	58	58	0	2
Waveform		5000	40	40	0	3

Table 3: Dataset properties 1

We tested each data set with four different classification tree algorithms: J48, REPTree, RandomTree and Logistical Model Trees. For each algorithm both the test options *percentage split* and *cross-validation* were used. With percentage split, the data set is divided in a training part and a test part. For the training set 66% of the instances in the data set is used and for the test set the remaining part. Cross-validation is especially used when the amount of data is limited. Instead of reserving a part for testing, cross-validation

Data set	**J48** tree size	**REP** tree size	**Random** tree size	**LMT** tree size
Arrythmia	41	29	475	1
Cylinder-band	30	430	7239	no mem
Thyroid disease	21	17	517	11
Chess game	43	67	2091	15
German Credit	64	96	1279	1
Letter	1423	1257	no mem	no mem
Mushroom	30	38	284	1
Nursery	369	518	2401	155
Optical Digits	247	207	2809	
Page Blocks	71	45	529	3
Segment	59	43	543	5
Sick	45	41	427	24
Spam	161	95	1681	39
Waveform	187	181	3005	1

Table 4: Dataset properties 2

repeats the training and testing process several times with different random samples. The standard for this is 10-fold cross-validation. The data is divided randomly into 10 parts in which the classes are represented in approximately the same proportions as in the full dataset (stratification). Each part is held out in turn and the algorithm is trained on the nine remaining parts; then its error rate is calculated on the holdout set. Finally, the 10 error estimates are averaged to yield an overall error estimate. Further reading about this topic and an explanation for the preference for 10-fold cross-validation can be found in [9].

To end this section we will discuss the four classification tree algorithms we will use in our experiment setting.

The J48 algorithm is WEKA's implementation of the C4.5 decision tree learner. The algorithm uses a greedy technique to induce decision trees for classification and uses reduced-error pruning [8].

REPTree is a fast decision tree learner. Builds a decision/regression tree using entropy as impurity measure and prunes it using reduced-error pruning. Only sorts values for numeric attributes once [9].

RandomTree is an algorithm for constructing a tree that considers K random features at each node. Performs no pruning [9].

LMT is a combination of induction trees and logistic regression. A combination of learners that rely on simple regression models if only little and/or noisy data is available and add a more complex tree structure if there is enough data to warrant such a structure. LMT uses cost-complexity pruning. This algorithm is significantly slower than the other algorithms [7].

For J48, REPTree and RandomTree all the tests were run with ten different random seeds. For the LMT algorithm we restricted the experiments to five runs, because of the time it costs to run that algorithm. For the first three algorithms one run took less than a minute, but for the LMT algorithm one run could take several hours. Choosing the different random seeds is done to average out statistical variations. Then for all computed percentages their mean and variance is computed. All data gathered by these tests are available on CD. Our results are described in the next section.

4 Results

First we will discuss what kind of results we can expect before we run the tests. The test option 10-fold validation should result in better performance than using percentage split, especially with smaller data sets, since more data is used for training. For larger data sets this difference will decrease. Using a pruning method in a classification tree algorithm will result in higher performance than when mining the data without pruning. We could also expect a larger data set to have a higher performance on testing. But we have to keep in mind that a large data set, especially when having many attributes, will produce a large decision tree, which will result in longer computation time.

		J48		REPTree		RandomTr		LMT	
		$10-fold$	$66\%split$	$10-fold$	$66\%split$	$10-fold$	$66\%split$	$10-fold$	$66\%split$
Arrhythmia	Mean	69,6	66,08	67,1	65,89	45,79	43,61	**75*** **	70,26
	Variance	1,4	6,77	3,32	13,1	9,6	19,03	6,08	11,05
Cylinder-band	Mean	57,53	56,11	58,7	56,31	**65,13***	63,46	-	-
	Variance	5,34	14,61	6,46	16,5	10,47	22,93	-	-
Hypothyroid	Mean	99,46	99,22	99,51	99,26	95,48	94,99	**99,54****	99,38
	Variance	0,01	0,05	0,04	0,05	1,38	1,42	0	0,03
Krvskp	Mean	99,08	98,78	99,16	98,66	87,38	86,19	99,65	**99,67*****
	Variance	0,01	0,26	0,12	0,13	0,79	8,27	0,01	0,03
Credit-G	Mean	72,35	71,45	72,2	72	65,8	64,39	**75,48*****	74,76
	Variance	1,38	2,54	0,9	3,57	11,03	4,86	0,74	5,25
Letter	Mean	**84,41**	82,88	84,01	82,53	-	-	-	-
	Variance	0,89	0,8	0,54	0,67	-	-	-	-
Mushroom	Mean	100	100	99,96	99,96	99,95	99,93	99,99	100
	Variance	0	0	0	0	0	0,01	0	0
Nursery	Mean	96,13	95,31	95,89	95,07	75,54	76,32	**98,81****	98,23
	Variance	0,12	0,15	0,04	0,3	9,35	25,49	0,07	0,15
Optidigits	Mean	88,48	87,32	88,8	87,32	74,05	73,15	**97,18*****	96,93
	Variance	0,05	0,54	0,55	1,27	1,96	3,19	0,02	0,14
Pageblocks	Mean	96,83	96,72	96,79	96,67	95,76	95,19	**96,9*****	96,81
	Variance	0,02	0,21	0,02	0,05	0,14	0,29	0,01	0,03
Segment	Mean	95,42	96,19	95,31	94,28	89,54	88,75	95,96	**96,34**
	Variance	0,24	0,53	0,16	1,9	1,54	4,71	0,21	1,42
Sick	Mean	98,52	98,31	98,62	98,49	96,08	95,54	98,92	**98,96*****
	Variance	0,03	0,17	0,04	0,2	0,59	2,29	0,01	0,09
Spambase	Mean	91,9	91,67	92,28	91,73	88,77	88,4	**93,43*****	93,11
	Variance	0,06	0,99	0,07	0,79	0,53	0,91	0,05	0,23
Waveform5000	Mean	76,09	75,94	76,46	76,15	62,35	61,32	86,68	**86,89*****
	Variance	1,11	1,17	0,34	1,05	0,58	3,88	0,46	0,4

Table 5: Overall Results (p<0.1*, p<0.05**, p<0.01***)

Looking at the results in Table 5 we see the following. LMT has got the best overall performance, this is shown in Table 6. On two of the three

J48		REP		Random		LMT	
10-fold	66%	10-fold	66%	10-fold	66%	10-fold	66%
87,56	86,84	87,49	86,74	74,4	73,66	**93,13**	92,61

Table 6: Overall means

data sets, for which other algorithms had a better but not significant better performance (Cylinder-band *p<0.1*, Letter *p>0.1*), the LMT algorithm did not have a score at all because WEKA ran out of memory, even in CLI mode. On the other set (Mushroom), the difference in performance was minimal. On three of the sets (Arrhythmia, Optidigits, Waveform5000), which had the best performance with LMT, the performance was much better than with the other algorithms (*p<0.01*). Interesting was that on those sets the performance of RandomTree was much worse than on the other sets, see Table 7. The performance was overall the lowest for RandomTree. But RandomTree had a remarkable better performance than the other two algorithms on the Cylinder-band set, although not significant (*p<0.1*), see Table 8.

	J48		REP		Random		LMT	
	10-fold	66%	10-fold	66%	10-fold	66%	10-fold	66%
Arrhyth	69,6	66,08	67,1	65,89	45,79	43,61	**75**	70,26
Optid	88,48	87,32	88,8	87,32	74,05	73,15	**97,18**	96,93
Wavef	76,09	75,94	76,46	76,15	62,35	61,32	86,68	**86,89**

Table 7: RandomTree performs much worse, where LMT performs much better

The performance of the J48 and RepTree algorithm were almost the same on all the data sets, but J48 is slightly better (*p>0.1*) on most of them. They are in most cases almost as good as the LMT algorithm, except for the earlier

	J48		REP		Random		LMT	
	10-fold	66%	10-fold	66%	10-fold	66%	10-fold	66%
Cyl-b	57,53	56,11	58,7	56,31	**65,13**	63,46	-	-

Table 8: RandomTree performs better

mentioned three cases. And for almost all the sets the cross-validation test method has a better performance.

Now we will look at the different dataset properties. If we define a data set with more than 1000 instances as a large data set, then we can say that indeed as we expected, large data sets perform better, with the exception of the Credit-g and Waveform set. For tree size we can not say if it influences the performance, because we see small and large data sets, modeled by "small" and large trees with varying performance.

Looking at the tree size of the RandomTree algorithm we see that it builds the largest trees (and has the lowest overall performance), something we would expect because that algorithm does not use pruning. Two algorithms that use reduced-error pruning build approximately equally sized trees. The LMT algorithm builds the smallest trees. This could indicate that cost-complexity pruning prunes down to smaller trees than reduced-error pruning, but it could also indicate that the LMT algorithm, which builds logistic models, does not need to build large trees to classify the data. The LMT algorithm seems to perform better on data sets with many numerical attributes. The three data sets, that performed much better than the rest, clearly have more numerical attributes than the other data sets. Whereas we see that for good performance in general, for all four of the algorithms, data sets with few numerical attributes gave a better performance.

5 Discussion

We see that LMT is the best classification tree algorithm of the four algorithms we used. We also see that not using a pruning method will result in lower performance. When using the same pruning method is seems that using another classification tree algorithm will make no difference.

Our results show, for the four classification tree algorithms we used, that using cost-complexity pruning has a better performance than reduced-error pruning. But as we said in the results section, this could also be caused by the classification algorithm itself. To really see the difference in performance in pruning methods another experiment can be performed for further/future research. Tests could be run with algorithms by enabling and disabling the pruning option and using more different pruning methods. This can be done for various classification tree algorithms which use pruning. Then the increase of performance by enabling pruning could be compared between those classification tree algorithms.

References

[1] L. Breiman, J.H. Friedman, R.A. Olshen, and C.T. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, California, 1984.

[2] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998.

[3] T. Elomaa and M. Kääriäinen. An analysis of reduced error pruning. *Journal of Articial Intelligence Research*, 15:163–187, 2001.

[4] Usama M. Fayyad. Data mining and knowledge discovery: Making sense out of data. *IEEE Expert: Intelligent Systems and Their Applications*, 11(5):20–25, 1996.

[5] A. Feelders. Classification trees. http://www.cs.uu.nl/docs/vakken/adm/trees.pdf.

[6] R. Kruse G. Della Riccia and H. Lenz. *Computational Intelligence in Data Mining*. Springer, New York, NY, USA, 2000.

[7] N. Landwehr, M. Hall, and E. Frank. Logistic model trees, 2003.

[8] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[9] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2005.