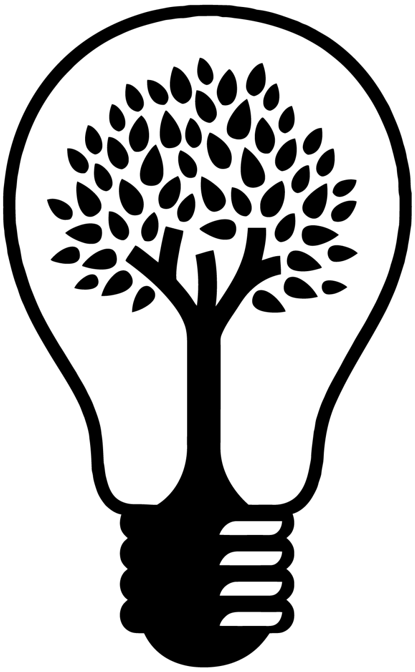
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Decision Trees

An Introduction



Seminar paper for the course *Artificial Intelligence* (KISEM)

Author	Date
Michael Dorner	April 7, 2014

Table of Content

Title page graphic: Copyright by Benni from The Noun Project

1 Introduction

1.1 What is a decision tree?

An economy student with only a very few programming skills shall develop a simple algorithm for sorting three elements A, B, C. He decides to divide this problem in smaller subproblems. First he wonders if A is smaller than B. In the second step it is interesting if B is smaller than C. If $A < B$ and $B < C$ than $A < B < C$. But if B is not greater than C, than a third question is relevant: Is $A < C$?

His head is spinning. Maybe solving this problem graphically is a better idea. He draws a node for each question and an edge for each answer. All leafs represent the correct order. Figure 1 shows the resulting graph:

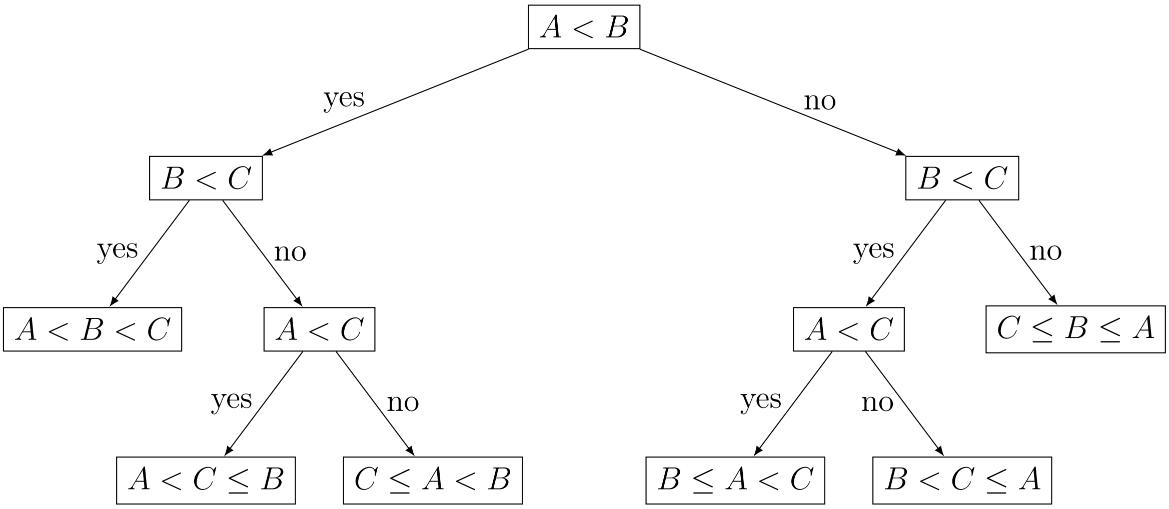


Figure 1: *A decision tree for sorting three values.*

Without further knowledge the student created his first decision tree.

For a right decision two or three if-statements are necessary. So a Python program could look like listing 1.

```
def sort1(A, B, C):
       if (A < B):
              if (B < C):
                     return [A, B, C]
              else:
                     if (A < C):
                            return [A, C, B]
                     else:
                            return [C, A, B]
       else:
              if (B < C):
                     if (A < C):
                            return [B, A, C]
                     else:
                            return [B, C, A]
              else:
                     return [C, B, A]

print( sort1(9,-2,0) ) # >> [-2, 0, 9]
print( sort1(2,0,0) ) # >> [0, 0, 2]
```

Listing 1: *A Python implementation of a decision tree for sorting three elements A, B, C*

Remark. The Python if-statement syntax and intends implies the tree structure in a vertical form, too.

Another way of formatting if-clauses represents a rule set which are first order logical expressions:

```
def sort2(A, B, C):
       if (A < B) and (B < C) : return [A, B, C]
       if (A < B) and not (B < C) and (A < C): return [A, C, B]
       if (A < B) and not (B < C) and not (A < C): return [C, A, B]
       if not (A < B) and (B < C) and (A < C): return [B, A, C]
       if not (A < B) and (B < C) and not (A < C): return [B, C, A]
       if not (A < B) and not (B < C) : return [C, B, A]
```

Listing 2: *A Python reimplementation of the decision tree given in listing 1 as a set of first order logical rules*

Unsurprisingly, we get six rules for 3! = 6 different combinations and the same results as in the algorithm `sort1`.

This introductory example shows four main things: Decision trees

1. work very well for classification and data mining.
2. are intuitive and self-explanatory.
3. They are easy to implement.
4. can be even used by economists.

Remark. Decision trees are used to model all comparison sorts like mergesort or quicksort. The reader may notice that the decision tree in figure 1 represents the insertion sort algorithm [**6**, p. 208].

1.2 Taxonomy

Before we approach the theory behind decision trees, a small but general overview of the taxonomy shall be given.

Decision tree classification is very often used in the context of data mining and machine learning. These keywords are no synonyms – although used as one very often. Machine learning cannot be seen as a true subset of data mining, as it also contains other fields, not utilized for data mining (e.g. theory of learning, computational learning theory, and reinforcement learning).

Figure 2 shows the machine learning context for decision trees.

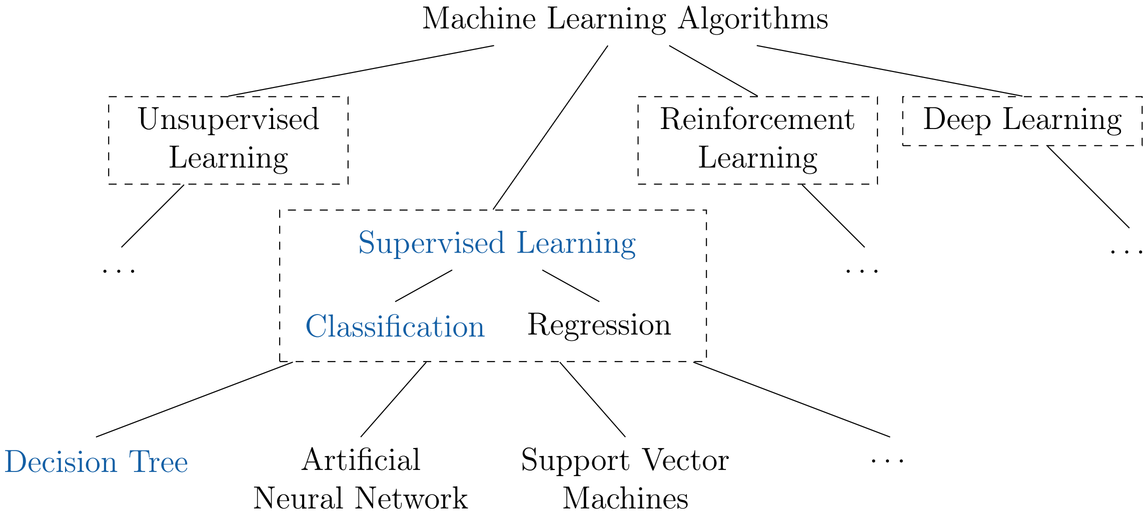


Figure 2: *The context of decision trees in machine learning with in this paper visited topics.*

Remark. The context shown in figure 2 is not intended for being complete, e.g. there is mixture of unsupervised and supervised learning, so called semi-supervised learning. Also not every decision tree can handle continuos values for regression analysis as we will see later.

Decision trees have a sibling, called regression trees. They have a common parent: prediction trees [**10**]. The basic idea is to use trees to model functions though each end point will result in the same predicted value, a constant for that point. Thus a regression tree is like a classification tree except that the end point will be a predicted function value rather than a predicted classification.

Although, there is a wide field of applications for regression trees, the focus of this paper in only on classification trees.

1.3 About this paper

This project work emerges in the context of the course *Artificial Intelligence* in the winter semester 2013/2014. Beside this seminar paper, an introductory presentation was conducted and an implementation for decision tree was developed.

In the scope of this seminar paper, a small introduction to theory and application of decision trees shall be given.

After this short introduction a theoretical consideration shall guide to a practical part, which shall clarify the theoretical part by examples. The last part shall summarize and compare the introduced algorithm and shall give a small outlook to not tackled research fields of decision trees.

On the contrary to the presentation during the seminar, this seminar paper expects a basic knowledge about graph theory, complexity, and machine learning. Instead of an introduction to these underlaying topics, a deeper look inside four decision tree algorithm families shall be given: CHAID, CART, ID3, and C4.5.

The focus of all Python implementation is on classification. This limitation is not owed to the insufficient importance of regression calculating, but a wider look would push the boundaries of this seminar paper.

Remark. Remark boxes like this one shall help to see the bigger pictures. It contains information that will not be explained any further, but which are a starting point for further investigations.

2 Theory of Decision Trees

In this section a (computer) scientific base for the informal introduction of decision trees shall be given. As mentioned in the introduction a basic knowledge in graph theory, machine learning and computer science in general is assumed.

2.1 Definitions

Definition 1. A **tree** is a directed, connected graph with one root node. Every other node has a single predecessor (**parent**) and no or more successors (**children**). Nodes without successors are called **leaves**. All nodes are connected by **edges**. The **depth** of a node is the number of edges on the path to the root. The **height** of the whole tree is the number of edges on the longest path from the root to any leaf.

Remark. This very rough definition focussed on trees shall not hide the fact that graph theory is complex and enormous mathematical field. For a deeper look e.g. [**6**] is recommended.

Definition 2. A **decision tree** is a tree with following equivalents:

Tree	Decision tree equivalent
Root	Initial decision node
Node	Internal decision node for testing on an attribute
Edge	Rule to follow
Leaf	Terminal node represents the resulting classification

As mentioned in subsection 1.2, machine learning is a set of algorithms that extract models representing patterns from data and then evaluate those models. Let us define three relevant terms, which are important for understanding the following algorithms descriptions: instance, attribute, class, and dataset:

Definition 3. The input of a machine learning algorithm consists of a set of **instances** (e.g. rows, examples or observations). Each instance is described by a fixed number of **attributes** (i.e. columns), which are assumed to be either nominal or numeric, and a label which is called **class** (in case of a classification task). The set of all instances is called **dataset**.

Instance	Attribute			Class
	$A < B$	$B < C$	$A < C$	
1	yes	yes	yes	$A < B < C$
2	yes	yes	no	$A < B < C$
3	yes	no	yes	$A < C \leq B$
4	yes	no	no	$C \leq A < B$
5	no	yes	yes	$B \leq A < C$
6	no	yes	no	$B < C \leq A$
7	no	no	yes	$C \leq B \leq A$
8	no	no	no	$C \leq B \leq A$

Table 1: *Dataset table for the sorting example from subsection 1.1*

Following this definition we get a table containing the dataset: Each decision becomes an attribute (all binary relations), all leaves are classes, while each row represents an instance of the dataset (see table 1).

Normally, the transformation is vice versa: the data is collected in table form (e.g. databases) and a decision tree has to be generated.

The reason why there are now eight instead of six classes is simple: it does not matter for instances 1, 2 and 3, 4, if $A < B$ or not; the result is the same class. This effect of removing irrelevant branches of a tree is called pruning and is also part of this paper (see 2.2.3).

2.2 Decision Tree Learning

In this subsection the question how to generate a decision tree from a given dataset in general shall be answered.

A founding idea of tree-based classification is based in the Concept Learning System [**19**]. All algorithms introduced in the next section are based on a simple but very powerful algorithm called TDIDT which stands for *Top-Down Induction of Decision Trees* [**19**]. This algorithm framework consists of two methods, growing and pruning a decision tree, which are introduced in the next two pseudocode listings and follows the idea of divide and conquer [**6**, p. 33].

Remark. σ is the relational operator for selection. See e.g. [**21**, p. 145] for further operators and more detailed information.

```
Algorithm 1: Tree Growing treeGrowing
Input : Training set $X$, attribute set $A$, target feature $y$
Output: Decision tree
Create a new tree $T$ with a single root node.
if stoppingCriterion($X$) then
    Mark $T$ as a leaf with the most common value in $X$ as a label.
else
    $\forall a_i \in A$ find $a$ that obtain the best splittingCriterion($a_i, X, y$).
    Label $n$ with $a$.
    for each outcome $v_i$ of $a$ do
        Set subtree $t_i$ = treeGrowing($\sigma_{a=v_i} X, A, y$).
        Connect the root node of $n_t$ to the subtree $t_i$ with an edge that is labelled as $v_i$.
return treePruning($S, T, y$)
```

A simplified decision tree 3 created by this algorithmic framework shall clarify how it works. The colors correspond to the variables in the pseudocode.

This framework gives three positions to adjust the framework: the splitting and the stopping criterion, as well as the tree is going to be pruned. The following parts shall give a quick overview of these positions.

```
Algorithm 2: Tree Pruning treePruning
Input  : Training set $X$, tree to be pruned $T$, target feature $y$
Output: Decision tree
repeat
    Select a node $n$ in $T$ such that pruning this node $n$ maximally improves some
    evaluation criteria.
    if $n \neq \emptyset$ then
        $T =$ pruned($T, n$)
until $t = \emptyset$;
return $T$
```

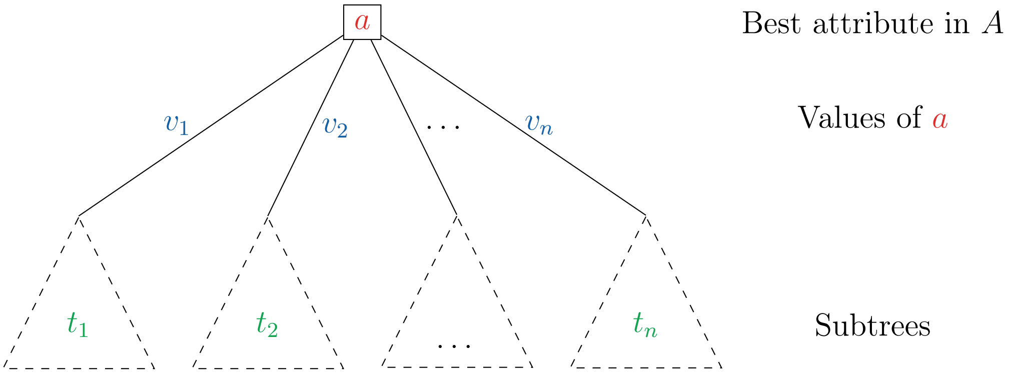


Figure 3: *The basic structure of a decision tree created by the algorithmic framework.*

2.2.1 Splitting Criterion

All introduced criteria are based on impurity measure. This defines how well classes are separated.

Remark. For further information about impurity based criteria [**22**, p. 53 ff.] is highly recommended.

Of course, it is not possible to list all criteria and ideas for splitting in the scope of this short paper. The selected criteria are needed for the presented algorithms in section 2.3.

Comparison of splitting criteria is a frequently visited research topic (e.g. [**3**], [**5**], [**16**], [**8**], [**26**]). Although, there is no extraordinary difference, each splitting criteria is superior in some cases and inferior in others; a general, scientifically sound statement which one is “better” is not possible.

For the different node splitting criteria we introduce some notation used throughout this section.

We assume there are total number of C classes denoted by $\Omega = \omega_1, \omega_2, \ldots, \omega_C$. Let there be N training examples represented by

$$\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(N)}, y^{(N)}\right) \tag{1}$$

where $x^{(i)}$ is a vector of n attributes and $y^i \in \Omega$ is the class label corresponding to the input $x^{(i)}$. Of these N examples, N_{ω_k} belong to the class ω_k, while $\sum_k N_{\omega_k} = N$. The decision rule splits these examples into P partitions, or P child nodes, each of which has $N^(p)$ examples. The number of examples in a particular partition p is denoted by $N^{(p)}_{\omega_k}$, while $\sum_k N^{(v)}_{\omega_k} = N^{(p)}$.

Entropy & Information Gain The idea of the information gain is based on the information theory which was introduced by Claude Elwood Shannon in 1948 [**25**]. For the computation we need two magnitudes:

Definition 4. The **entropy** and the **information gain** are defined as

$$G(a_j) = \left(\sum_{k=1}^{C} -\frac{N_{\omega_k}}{N} \log_2 \frac{N_{\omega_k}}{N}\right) - \left(\sum_{p}^{P} \frac{N^{(p)}}{N} \cdot \sum_{k=1}^{C} -\frac{N^{(p)}_{\omega_k}}{N^{(P)}} \log_2 \frac{N^{(p)}_{\omega_k}}{N^{(P)}}\right) \tag{2}$$

while the first term is the entropy and the second term the weighted entropy of the child nodes. The difference thus reflects the decrease in entropy or the information gained from the use of attribute a_j.

Remark. There is a detailed example in the presentation slides which should clarify the application of the formula.

One of the problems that arises is that the information gain criterion as defined in equation 2 favors large number of partition P. So the an improvement was suggested by [**18**], which is now used in the C4.5: instead of the information gain $G(a_j)/g$ is used, where

$$g = \sum_{p=1}^{P} \frac{N^{(p)}}{N} \log_2 \frac{N^{(p)}}{N} \tag{3}$$

Gini Index & Gini Gain

Definition 5. The **Gini index** measures the divergences between the probability distributions of the target attributes values and is defined as

$$D(a_j) = \frac{1}{N}\left(\sum_{k=1}^{C}\sum_{p=1}^{P}\frac{\left(N_{\omega_k}^{(p)}\right)^2}{N^{(p)}} - \sum_{k=1}^{C}\frac{(N_{\omega_k})^2}{N}\right) \tag{4}$$

The goal is to find a node which is the most "pure" one, i.e. has instances of a single class. Similar to the decrease in entropy and gain information used in the information gain based criterion, the impurity as given in 5 is used. The chosen attribute is one that has the largest decrease in impurity.

Twoing Criterion

Definition 6. The binary twoing criterion maximizes the function

$$\frac{P(t_L)\cdot P(t_R)}{4}\cdot\left(\sum_{c\in a_i}|P(c|t_L)-P(c|t_R)|\right)^2 \tag{5}$$

for a node t, where $P(t_L)$ and $P(t_R)$ are the probability of going left or right, respectively, and $P(c|t_L)$ and $P(c|t_R)$ are the proportions of data points in t_L and t_R which belong to class c.

Figure 4 visualizes the used variables:

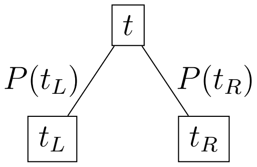


Figure 4: *Twoing criterion is a measure of the difference in probability that a category appears in the left descendant rather than the right descendant node.*

When the target attribute is binary the Gini and twoing criterion are equivalent [**26**]. The towing rule is more appropriate for data, which has a large number of different classes. For multi-class problems the twoing criteria prefers attributes with evenly divided splits [**22**, p. 57].

Chi-Squared Statistic Criterion The chi-squared statistic (χ^2) criterion is based on comparing the obtained values of the frequency of a class because of the split to the *a priori* frequency of the class.

Definition 7. The formula for computing the χ^2 value is

$$\chi^2 = \sum_{k=1}^{C} \sum_{p=1}^{P} \frac{\left(\left| N_{\omega_k}^{(p)} \right| - \left| \tilde{N}_{\omega_k}^{(p)} \right| \right)^2}{\left| \tilde{N}_{\omega_k}^{(p)} \right|} \tag{6}$$

where $\tilde{N}_k^{(p)}$ is the expected frequency of the samples N in k.

A larger value of χ^2 indicates that the split is more homogeneous, i.e. has a greater frequency of instances from a certain class. The attribute is chosen by the largest value of χ^2.

2.2.2 Stopping Criterion

The growing phase continues until a stopping criterion is triggered. The following conditions are common stopping rules [**22**, p. 63]:

- All instances in the training set belong to a single value of y.
- The maximum tree depth has been reached.
- The number of cases in the terminal node is less than the minimum number of cases for parent nodes.
- If the node were split, the number of cases in one or more child nodes would be less than the minimum number of cases for child nodes.
- The best splitting criteria is not greater than a certain threshold.

Remark. The last stopping criterion is used in the implementation for pruning after the growing phase.

2.2.3 Tree Pruning

Using a tight stopping criteria tends to create small and underfitted decision trees. On the other hand, using loose stopping criteria tends to generate large decision trees that are overfitted to the training set.

To avoid both extremes, the idea of pruning was developed: A loose stopping criterion is used and after the growing phase, the overfitted tree is cut back into a smaller tree by removing sub-branches that are not contributing to the generalization accuracy.

Many approaches were developed and even the in the following presented ones have different versions with improvements. However, the focus is on the basic ideas.

Reduced Error Pruning Reduced error pruning is a basic pruning approach introduced by Ross Quinlan in 1987 [**20**].

```
Algorithm 3: Reduced Error Pruning
Input  : Training set $X$, attribute set $A$, target feature $y$
Output: Pruned decision tree
Subdivide $X$ into a training set $X_T$ and validation set $X_V$.
Build a tree $T$ = treeGrowing($X_T, A, y$).
Pass all of the training examples $X_V$ through the tree $T$ and estimate the error rate of
  each node $n$ using $X_V$.
Convert a node to a leaf if it would have lower estimated error than the sum of the
  errors of its children.
return $T$
```

However the splitting into two sets is not welcome, because it reduces the size of training set.

Cost-Complexity Pruning Proposed by Leo Breiman in 1984, the pruning method is also known as weakest link pruning or error-complexity pruning. Contrary to the reduced error pruning this pruning approach is not straight forward [**22**, p. 64].

The idea is to consider the size and the estimated error rate of the tree. The total cost $C_\alpha(T)$ of tree T is defined as

$$C_\alpha(T) = R(T) + \alpha|\tilde{T}|, \qquad \alpha \geq 0 \tag{7}$$

where $R(T)$ is the weighted summed error of the leafs of tree T and α is penalty for the complexity of the tree $\tilde{T}$ (so called complexity parameter), while $\tilde{T}$ stands for all leaves in the tree T.

For a fixed value of α there is a unique smallest minimizing subtree $T(\alpha)$ of the complete tree $T_{\max}$, that fulfills the following two conditions [**16**]:

$$① \qquad C_\alpha\left(T(\alpha)\right) = \min_{T \subseteq T_{\max}} C_\alpha(T) \tag{8}$$

$$② \qquad \text{If } C_\alpha(T) = C_\alpha\left(T(\alpha)\right) \text{ then } T(\alpha) \subseteq T \tag{9}$$

The first condition says that there is no subtree of $T_{\max}$ with lower costs than $T(\alpha)$ at that α. The second condition says that if more than one tree achieves the same minimum, the cost-complexity pruning selects 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 for subtrees of $T_{\max}$ would look like

$$T_1 \supseteq T_2 \supseteq T_3 \supseteq \cdots \supseteq \{n\}$$

with n as the root node of T and T_n is the smallest subtree for $\alpha \in [\alpha_n, \alpha_{n+1})$.

Error-Based Pruning The goal is to improve the estimate of error on unseen data using only training set data. Hence if a node does not increase estimated error, we prune it.

The error estimate for a subtree is the weighted (based on how many instances of each there are) sum of the error estimates for all its leaves, defined by

$$\varepsilon(T,S) = \varepsilon(T,S) + Z_\alpha \cdot \sqrt{\frac{\varepsilon(T,S) \cdot (1 - \varepsilon(T,S))}{|S|}} \tag{10}$$

where $\varepsilon(T,S)$ denotes the misclassification rate of the tree T on the training set S; Z is the inverse of the standard normal cumulative distribution; and α is the desired significance level.

Remark. Cost-complexity pruning and reduced error pruning tends to over-pruning, while error-based pruning tend to under-pruning [**22**, p.68].

2.3 Selected Algorithms

Before we immerse in the underlying theory, a small historical background of all shall be given.

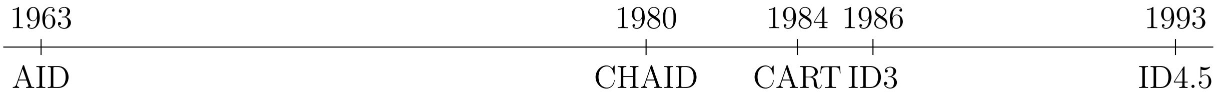


Figure 5: *Timeline of four basic algorithm families: CHAID, CART, ID3 and ID4.5*

Remark. As in the considered parts of decision tree theory, this is not a complete summarization of all decision tree algorithms. Instead, basic and popular algorithms were selected and will be introduced in the following.

2.3.1 Chi-squared Automatic Interaction Detector (CHAID)

In 1964, the statisticians John A. Sonquist and James N. Morgan introduced a first version of tree learning algorithm called Automatic Interaction Detection (AID) [**17**].

Gordon V. Kass proposed a modification to AID in his paper from 1979 [**12**] called *Chi-squared Automatic Interaction Detector* (CHAID).

The basic improvement was splitting criterion: instead of the variance the χ^2-test is used. Therefore, the partitions need not be a bisection.

CHAID handles missing values by treating them as a separate valid category and does not perform any pruning.

2.3.2 Iterative Dichotomiser 3 (ID3)

The ID3 algorithm was developed by Ross Quinlan in 1986. It uses the information gain as splitting criterion and it does not apply any pruning procedure nor does it handle numeric attributes or missing values.

2.3.3 Classification And Regression Tree (CART)

Leo Breiman published his idea of CART in 1984 [**quinlandecisiontreediscovery**]. As the name suggests CART supports also regression trees. This book provides a strong statistic foundation and revived the idea of CLS [**9**, p. 435].

Two splitting criteria can be chosen: the twoing or Gini criterion [**26**, p. 1]. The obtained tree is pruned by cost-complexity pruning.

2.3.4 C4.5

C4.5 is the updated version of ID3 by the same author [**13**]. It uses also the information gain, but extends this by the option of handling missing value and continuos values. This new splitting criterion is called gain ratio (see formula 3).

After the growing phase an error-based pruning is performed.

Remark. There is a commercial successor of C4.5, called C5.0 [**13**], but it has not influenced the decision tree learning in the way Quinlan's two other algorithms did: no scientific papers about the C5.0 itself are findable.

2.4 Discussion

In this part, the usage of decision trees shall be discussed. The pro and con statements are very general and thus not very precise. There are several approaches to avoid or at least minimize the disadvantages and maximize the benefits and advantages, as mentioned in the previous parts.

2.4.1 Advantages

Also a lot of advantages can be recognized [**22**, p. 73ff.]. Decision trees

- are self-explanatory and can be converted into a set of rules, which is easy and fast to interpret, understand, and implement: A path from the root to a leaf is the explanation for the resulting classification which is given by the leaf (terminal node).
- can handle both nominal and numeric input values.
- are capable of handling data sets with missing values.
- are a nonparametric method, which means that they have no assumptions about the space distribution and the classifier structure.

2.4.2 Disadvantages

On the other hand, decision trees have disadvantages such as

- an over-sensitivity to the training set, to irrelevant attributes, and to noise.
- a requirement that the target attribute consists only of discrete values (in case e.g. ID3 or C4.5).

- a tending to perform well if a few highly relevant attributes exist, because decision trees use the divide and conquer method; but less so if many complex interactions are present.

2.5 Outlook

This subsection is about what was not part of the considerations of this seminar paper, but, in the humble opinion of the author, is important or interesting for working with decision trees.

2.5.1 Complexity & Performance

Like for all computer algorithms the time and space complexity is relevant. [**4**] gives a broad overview of the complexity of decision trees in general.

In the following some assumptions contain the wide field of complexity. We suppose that we have n training instances, m attributes, $O(n)$ nodes (up to one leaf per example), and the tree depth of $O(\log n)$. So the complexities are given by table 2.

Building a tree	$O(m \cdot n \log n)$
Subtree replacement	$O(n)$
Subtree raising	$O\left(n \cdot (n \log n)^2\right)$
• Every instance may have to be redistributed at every node between its leaf and the root	$O\left(n \log n\right)$
• Cost for redistribution (on average)	$O\left(\log n\right)$
Total Costs	$O(m \cdot n \log n) + O\left(n \cdot (n \log n)^2\right)$

Table 2: *Complexity overview of a assumed decision tree*

But [**27**] showed that it is possible with a novel algorithm to achieve a time complexity of $O(m \cdot n)$.

Due to the fact that there are many improvements for each algorithm, a general statement is not feasible. But several research approaches compare the performances of decision tree algorithms with each other (e.g. [**15**], [**1**], [**23**]) and decision trees with other classification approaches (e.g. [**11**], [**7**]).

2.5.2 Missing Attributes

It is common, that classification problems have missing attributes during training, during classification, or both. Consider first training a tree classifier despite the fact that some training patterns are missing attributes.

There are several approaches to handle missing values. Numerical values can be initialized by a meaningful value (minimum, maximum, average, −1, etc.). For nonnumerical attributes it might be useful to choose the commonest attribute value.

For a more detailled description of the approaches to handle missing values, e.g. [**31**] and [**22**, p. 59 ff.] are recommended.

2.5.3 Random Forests

A Random forest is a classifier consisting of n decision trees [**28**]. This method combines the *Bagging* (Bootstrap aggregating, [**22**, p. 105ff.]) method and the random selection of features [**22**, p. 87 ff.].

Figure 6 gives a short graphical overview of the data flow and the functional principle of random forests.

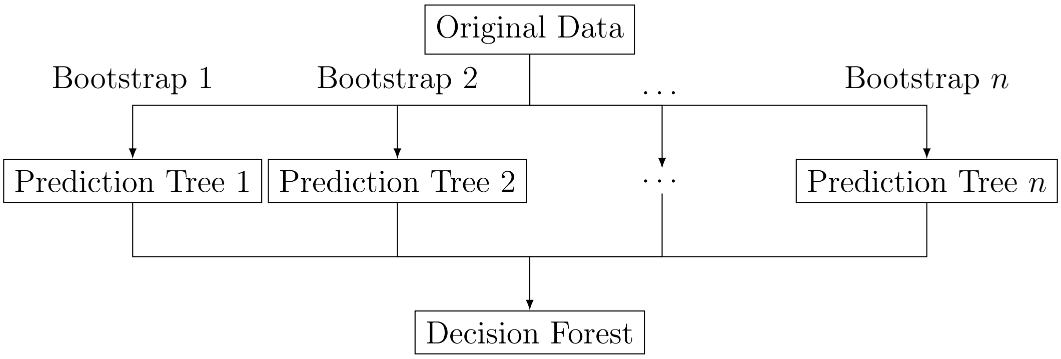


Figure 6: *The random forest schematic*

3 Summary & Conclusion

3.1 Applications

Decision trees have a wide field of applications. In this subsection some examples of the applications are listed.

Astronomy [24] applied decision tree learning to the task of distinguishing between stars and cosmic rays in images collected by the Hubble Space Telescope.

Chemistry The relationship between the research on octane (ROC) number and the molecular substructures were explored in the paper [2].

Medicine In [29] decision trees are applied for the diagnosis of the ovarian cancer.

Economy The results of the research project on decision trees used in stock trading was published in [30].

Geography [14] used classification trees to predict and correct errors in topographical and geological data.

3.2 Programming Example

In the programming example a decision tree induction algorithm was implemented in the programming language Python. The focus during the development was on readability and understanding, less on performance and software architecture.

Two splitting criteria are implemented: Information and Gini gain (ID3 and CART). Also a basic pruning algorithm can be used, which uses a threshold for the pruning decision (see subsection for the idea of stopping criteria). The decision tree itself is implemented as binary tree and does not support regression analysis.

Two examples are enclosed: Tuberculoses/pneumonia and fish iris classification. Both are from real world, while the first example consists of only a handful instances due to the fact that this example was calculated completely by hand in the presentation. The second example origins from Matlab demo files.

The code is well commented and has a demo application for each example.

3.3 Summary

In the scope of this paper, a small introduction to decision trees was given. The introductory example showed the working principle and advantages of decision trees. An overview of machine learning approaches helped to see the bigger picture.

The theory part started with some necessary definitions which are used in the following parts. The basic top-down induction of decision trees algorithm was introduced and the options for improving this framework were described mathematically.

Four decision tree algorithms were selected and presented to the reader: CHAID, ID3, CART, and C4.5. All facts from the previous sections were compared and traded off one against the other. Due to the limited scope of this seminar paper some parts were not considered in detail, but a small outlook on these interesting topics is given.

The last part shows some examples where decision trees find their application in the real world and how many-faceted this field is. The programming example with a small code documentation completes the picture of decision trees.

A Bibliography

[1] Robert E. Banfield et al. “A comparison of decision tree ensemble creation techniques”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29.1 (2007), pp. 173–180.

[2] Edward S. Blurock. “Automatic learning of chemical concepts: Research octane number and molecular substructures”. In: *Computers & chemistry* 19.2 (1995), pp. 91–99.

[3] Leo Breiman. “Technical note: Some properties of splitting criteria”. In: *Machine Learning* 24.1 (1996), pp. 41–47.

[4] Harry Buhrman and Ronald De Wolf. “Complexity measures and decision tree complexity: a survey”. In: *Theoretical Computer Science* 288.1 (2002), pp. 21–43.

[5] Wray Buntine and Tim Niblett. “A further comparison of splitting rules for decision-tree induction”. In: *Machine Learning* 8.1 (1992), pp. 75–85.

[6] Thomas H. Cormen et al. *Introduction to algorithms*. Vol. 2. MIT Press, 2001.

[7] Stephen P. Curram and John Mingers. “Neural networks, decision tree induction and discriminant analysis: An empirical comparison”. In: *Journal of the Operational Research Society* 45.4 (1994), pp. 440–450.

[8] Chris Drummond and Robert C. Holte. “Exploiting the cost (in) sensitivity of decision tree splitting criteria”. In: *ICML*. 2000, pp. 239–246.

[9] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern classification*. John Wiley & Sons, 2012.

[10] Sally Goldman and Yoram Singer. “Self-Pruning Prediction Trees”. In: *WiML* (2013). URL: `http://snowbird.djvuzone.org/2010/abstracts/122.pdf`.

[11] Jin Huang, Jingjing Lu, and Charles X. Ling. “Comparing naive Bayes, decision trees, and SVM with AUC and accuracy”. In: *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*. IEEE. 2003, pp. 553–556.

[12] Gordon V. Kass. “An exploratory technique for investigating large quantities of categorical data”. In: *Applied statistics* (1980), pp. 119–127.

[13] Ron Kohavi and J. Ross Quinlan. “Decision Tree Discovery”. In: *Handbook of Data Mining and Knowledge Discovery*. University Press, 1999, pp. 267–276.

[14] Philippe Lagacherie and Susan Holmes. “Addressing geographical data errors in a classification tree for soil unit prediction”. In: *International Journal of Geographical Information Science* 11.2 (1997), pp. 183–198.

[15] D. Lavanya and K. Usha Rani. “Performance Evaluation of Decision Tree Classifiers on Medical Datasets”. In: *International Journal of Computer Applications* 26 (2011).

[16] John Mingers. “An empirical comparison of selection measures for decision-tree induction”. In: *Machine learning* 3.4 (1989), pp. 319–342.

[17] James N. Morgan and John A. Sonquist. “Problems in the analysis of survey data, and a proposal”. In: *Journal of the American statistical association* 58.302 (1963), pp. 415–434.

[18] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* C4.5 - programs for machine learning / J. Ross Quinlan. Morgan Kaufmann Publishers, 1993.

[19] J. Ross Quinlan. “Induction of decision trees”. In: *Machine learning* 1.1 (1986), pp. 81–106.

[20] J. Ross Quinlan. “Simplifying decision trees”. In: *International journal of man-machine studies* 27.3 (1987), pp. 221–234.

[21] P. Rob, C. Coronel, and K. Crockett. *Database Systems: Design, Implementation & Management.* Cengage Learning, 2008.

[22] L. Rokach. *Data Mining with Decision Trees: Theory and Applications.* Series in machine perception and artificial intelligence. World Scientific Publishing Company, Incorporated, 2008.

[23] S. Rasoul Safavian and David Landgrebe. “A survey of decision tree classifier methodology”. In: *IEEE transactions on systems, man, and cybernetics* 21.3 (1991), pp. 660–674.

[24] Steven Salzberg et al. “Decision trees for automated identification of cosmic-ray hits in Hubble Space Telescope images”. In: *Publications of the Astronomical Society of the Pacific* (1995), pp. 279–288.

[25] Claude Elwood Shannon. “A mathematical theory of communication”. In: *The Bell System Technical Journal* 27.1 (1948), pp. 379–423, 623–656.

[26] Yu-Shan Shih. “Families of splitting criteria for classification trees”. In: *Statistics and Computing* 9.4 (1999), pp. 309–315.

[27] Jiang Su and Harry Zhang. “A fast decision tree learning algorithm”. In: *Proceedings of the National Conference on Artificial Intelligence.* Vol. 21. 1. AAAI Press; MIT Press; 1999. 2006, p. 500.

[28] Weida Tong et al. “Decision forest: combining the predictions of multiple independent decision tree models”. In: *Journal of Chemical Information and Computer Sciences* 43.2 (2003), pp. 525–531.

[29] Antonia Vlahou et al. “Diagnosis of ovarian cancer using decision tree classification of mass spectral data”. In: *BioMed Research International* 2003.5 (2003), pp. 308–314.

[30] Muh-Cherng Wu, Sheng-Yu Lin, and Chia-Hsin Lin. “An effective application of decision tree to stock trading”. In: *Expert Systems with Applications* 31.2 (2006), pp. 270–274.

[31] Shichao Zhang et al. “" Missing is useful": missing values in cost-sensitive decision trees”. In: *Knowledge and Data Engineering, IEEE Transactions on* 17.12 (2005), pp. 1689–1693.