ngerman

Lazy Rule Learning

Lazy Rule Learning
Bachelor-Thesis von Nikolaus Korfhage
Januar 2012

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Fachgebiet Knowledge Engineering

Lazy Rule Learning
Lazy Rule Learning

Vorgelegte Bachelor-Thesis von Nikolaus Korfhage

1. Gutachten: Johannes Fürnkranz
2. Gutachten: Frederik Janssen

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 23. Januar 2012

(Nikolaus Korfhage)

Contents

1 Introduction

In the mid-1950's, the field of Artificial intelligence (AI) research was founded. AI is the area in computer science that concerns study and design of machines that humans consider intelligent. Certainly it is hard to decide, when a machine can be considered as intelligent, or if a computer really can be intelligent. Up to now, an artificial general intelligence (strong AI) has not yet been developed, i.e. a machine that can successfully perform any intellectual task that a human can. However, for certain problems, whose solutions require some kind of intelligence, computer programs have been designed that perform equally well or even better than humans. For example, a chess computer that defeats human chess champions shows in a way intelligent behavior for the particular domain of chess playing.

Artificial intelligence is an essential part of many today's technologies and is used in a wide range of fields: Among others, Artificial intelligence applications can be found in medical diagnosis, robot control, and stock trading. It plays an important role in every day applications like email spam filtering or voice recognition. The field of Artificial intelligence can be divided into various subfields concerning different subproblems. These problems include perception, reasoning, knowledge representation, planning, learning, communication and haptic interaction with objects.

1.1 Machine Learning

An important branch of AI is machine learning that is concerned with the design of *learning* algorithms. Such algorithms are capable of evolving intelligent behaviors based on experience. The objective of the learning algorithm is to generalize from its experience.

One well known application of machine learning is spam filtering in emails. For a person it is easy to decide whether a received email is a spam message or not. But since having to do so is annoying and wearying in the long run it is tempting to let a computer learn classifying spam and hereby filter the spam messages out before the user gets to view his incoming messages.

Such a filter has to be trained to work effectively. Particular words occur with certain probabilities in spam but not in desired emails. To learn about these probabilities, the spam filter has to analyze collected samples of spam mail and valid mail. For all words occurring in a training email, the spam filter will adjust the measured probability for that particular word of appearing in spam. After training, the word probabilities are used to compute the probability that an email with a particular set of words in it belongs to either category. The filter will mark the email as spam, if the total of word probabilities exceeds a certain threshold. If the filter performs well on classifying spam, that is if it prevents spam messages from being shown to the user in the majority of cases, it has *learned* how to classify spam.

1.1.1 Supervised Learning

A usual task in machine learning is supervised learning. In supervised learning a function is inferred from a set of supervised (labeled) training data. Each example in the training data is a pair consisting of an input object - a vector of properties - and an output value. The task is now to learn a function from this set of examples that gives a correct output value for a new input object whose output value is unknown. Supervised learning is further divided into *classification* and *regression*: In classification, the unknown objects should be categorized into separate categories (classes). A classification task can for example be to predict, given a set of examined symptoms, whether a patient suffers a specific disease or not. On the other hand, in regression, a real value should be predicted. For example, a regression task can be to predict how high the temperature will be tomorrow. However, in this thesis the focus is on the task of classification.

A descriptive example for a classification task is to let an algorithm decide if it is the right weather to do sports or not, according to some weather measures [1]. The learning algorithm analyzes the given training data and produces a *classifier* that will be used for classifying the unknown instances, i.e. predicting the unknown output value of that instance. That training data consists of a set of instances, each instance divided into two parts; first a vector of observed characteristics and second a label, also called class value. In the weather example, a possible vector of observed characteristics can be:

$$< Outlook = sunny,\ Humidity = 50,\ Wind = FALSE,\ Temperature = hot >$$

In this case, the label of the specific training instance may either be $Sport = yes$ or $Sport = no$. After processing all training instances, the learning phase is complete. Now, the learned classifier is ready to predict classes for unknown instances. Such unknown instances are also called query instances or training instances.

[1] The example is based on (Mitchell, 1997) and will be used throughout this thesis.

1.1.2 Rule Learning and Lazy Learning

The major part of this thesis adresses *rule learning*. For now, the concepts of rule learning and lazy learning will be briefly summarized, but will be explained in further detail within the following sections (cf., section 2 and section 4). Rule learning algorithms are a special type of learning algorithms. During the learning process such an algorithm formulates rules. Rules have an if-then structure. For example, a possible representation of a single rule, specified to classify exactly the above example is:

$$\textbf{if } (Outlook = sunny) \wedge (Humidity = 50) \wedge (Wind = FALSE) \wedge (Temperature = hot) \textbf{ then } (Sport = yes)$$

Usually a single rule is not enough to classify the whole set of training instances. The classifier then has to learn a set of rules to classify the *whole* set of training instances.

Learning algorithms distinguish between eager and lazy learning. In eager learning a classifier (in the case of rule learning a set of rules) is learned for the *whole* training data and afterwards used for classification of unknown instances. Whereas in lazy learning at first, nothing is learned from the training data: The classification is postponed until a specific query instance has to be classified.

1.2 Objective of this thesis

This thesis will first give an overview about two particular not yet related subareas in machine learning, namely *rule learning* and *lazy learning*. In both sections basic concepts and common algorithms will be introduced.

The main objective of this thesis though is, to present a solution that combines both approaches. A motivation to learn rules a lazy way, is to obtain simple and comprehensible rules, each of them tailored to classify exactly one instance. In contrast, learning sets of rules, as eager rule learning does, yields a set of rules in which only the first rule is without context (i.e. independent of other rules). Especially in semantic web applications (cf., section 8) this kind of lazy learned rules can be of interest.

The centerpiece of this thesis is a new algorithm that combines both approaches within one algorithm. First, the algorithm is examined theoretically and then applied evaluated. An existing framework (Janssen and Fürnkranz, 2010a) is used as base for its implementation and evaluation. Furthermore, the algorithm provides an interface for evaluation in the Weka suite (Hall et al., 2009). For the proposed lazy rule learning algorithm possible improvements are examined. A small set of configurations of the algorithm that emerged to be superior to other variants is then selected for further evaluation. Finally, the algorithm is compared to other learning algorithms, in particular to lazy learning and rule learning algorithms.

2 Rule Learning

Rule learning is the fundamental concept of the algorithm presented later. For this reason, the terminology introduced in the previous section will be explained in greater detail. In order to estimate a rule's quality, the heuristic value (cf., 2.2) has to be computed. Heuristics are important for the following class of separate-and-conquer rule learning algorithms (cf., section 2.3) as well as for the new lazy rule learning algorithm (cf., section 4).

2.1 Terminology

In order to learn a rule or a set of rules, training data is needed. This training data consists of instances (also called examples) whose classifications are known. An instance consists of attributes (or features) that have a particular value assigned. The attribute may either be *nominal* or *numeric*. Possible values of a nominal attribute are a limited set of distinct values. An example for a nominal attribute is the attribute $Outlook$, that may assign the values $sunny$, $cloudy$ or $rainy$. To a numeric attribute a real value is assigned. Such an attribute is for example the attribute $Temperature$ that may assign any real value that reasonably describes a temperature. In classification learning, one nominal attribute of the instance is identified as class attribute. The possible values of this class attribute are the classes. While in the training data the value of the class attribute is known, the class of a test instance (the instance to be classified by the classifier, also called query instance) is unknown. For better understanding the procedure of rule learning, consider the following example. Figure 1 shows four training examples on which a rule or a set of rules should be learned. Such a small dataset would actually not allow to learn a useful classifier from, but for exemplifying the definitions and terms of rule learning it is suited well.

```
@relation weather
@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
@attribute sport {yes, no}
@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
sunny,72,95,FALSE,no
overcast,83,86,FALSE,yes
```

Figure 1: Four training examples, modified excerpt of the dataset *weather.arff*, in Attribute-Relation File Format (ARFF) (Hall et al., 2009).

The first part of the file that contains the training data, defines the attributes and its domain values, i.e. the set of possible values. For the nominal attribute *outlook* these are *sunny*, *overcast* and *rainy*. The attribute *humidity* is a numeric attribute and may assign a real number. Attribute *sport* will be considered as the class attribute because it is the last attribute defined in the list of attributes. The second part consists of four training examples. The values of each instance appear in the same order as the attributes were defined in the head of the file and represent the particular assignments of the attributes, whereas the last attribute consequently specifies the class of the instance. It may also happen, that the value of a specific attribute is unknown for an instance. This instance then contains a *missing value* for this attribute, denoted as "?" instead of a proper value. It depends on the learning algorithm, how missing values are treated. If a value is required, the most likely value is assumed, in other algorithms the attribute with the missing value can simply be ignored.

Classifiers can be represented as a set of rules. A classifier may consist of one or more rules. One single rule consists of a rule head and a rule body. The rule head and rule body consist of conditions. A condition is an attribute with a value assigned to it. The head of a rule always contains a single condition that is a class value assigned to the class attribute. On the other hand, the rule's body might contain an arbitrary number of conditions according to the attributes of the dataset. Rules without any conditions in their body are called *empty rules*. Figure 4 shows a set of four rules that classifies the *whole* training data correctly, while figure 2 shows that two rules are enough to classify all instances correctly. In contrast, the rule in figure 3 would classify only two examples correctly but one wrongly as $sport = no$.

The head and body of a rule are separated by the string ":-" while the first part is the rule's head and the second part its body. Conditions in the body are separated by comma and the rule's end is marked by a dot. If all conditions in the body hold true the condition in the rule's head must hold true as well.

```
sport = yes :- outlook = overcast.
sport = no :- outlook = sunny.
```

Figure 2: Single rule classifying exactly one instance of the data in figure 1. All instances are covered and correctly classified. Here both, the rule's head and the rule's body, each contain one condition.

```
sport = no :- windy = FALSE.
```

Figure 3: Single rule classifying two instances of the training data in figure 1 correctly. One instance is wrongly covered by the rule and one wrongly stays uncovered.

Here it is assumed that the class attribute has only two values, *yes* and *no*. If a class only has two possible values, it is a *binary* class. An example is then either a *positive* example (here *Sport* = *yes*) or *negative* example (here *Sport* = *no*). The training data in figure 1 thus contains three negative and one positive example. A rule *covers* an example, if all conditions of the rule's body hold true. Examples that are not covered by a rule are called *uncovered examples* respectively.

For example, the rule in figure 3 covers three examples. However, the second example is not covered. Two of the covered examples, namely the first and the third, are classified correctly. They are said to be *true positives*. Although the last example is covered by the rule's body too, it is wrongly classified as *Sport* = *no*. Such a wrongly covered example is called a *false positive*. Accordingly, if the conditions of the rule's body do not hold true for the example and the example is classified correctly as negative, it is called a *true negative*. In contrast, the second training example is a *false negative* because it is not covered by the rule i.e. not considered as *Sport* = *yes*, but its class value is actually the same like in the condition of the rule's head.

Consider a classifier that classifies the whole set of training data correctly, i.e. there is not any wrongly classified instance in the training data, but when unknown instances are passed to the classifier, it performs rather poor and frequently predicts the wrong class for the new instances. This may happen when the classifier is adapted too much to the possibly noisy training data[2] and is called *overfitting* of the classifier to the training data. An extreme example of an overfitted classifier is the rule set in figure 4, where the rules are directly derived from the training set and only the examples of the training with *sport* = *no* would be classified correctly but not any instance that does not occur in the training data but have also assigned the class value *no* to the class attribute *sport*. Pruning (cf., section 5.7.2) is a way to handle the problem of overfitted classifiers.

```
sport = no :- outlook = sunny, temperature = 85, humidity = 85, windy = FALSE.
sport = no :- outlook = sunny, temperature = 80, humidity = 90, windy = TRUE.
sport = no :- outlook = sunny, temperature = 72, humidity = 95, windy = FALSE.
sport = yes :- outlook = overcast, temperature = 83, humidity = 86, windy = FALSE.
```

Figure 4: Three rules classifying the training data in figure 1 correctly. The rule set only covers the four examples of the training examples. New examples with *sport* = *no* that differ from the training examples would not be covered.

In the preceding example it was assumed that the class attribute is a binary one i.e. that it has only two possible class values for the class attribute *sport*, namely *yes* and *no*. The task to decide between more than two class values is called a *multi-class problem*. One solution is to learn a rule set in a certain order and to always use the first rule that fires for an instance i.e. whose rule body applies to the instance. This is called *decision list*. Another solution is to treat each class separately. The multi-class problem is then viewed as a series of binary (two-class) problems. In *one-against-all binarization* each class is once considered to be positive while all other classes are considered to be negative. The rule that performs best is used for classification. To decide which of two rules is better, a rule's performance can be determined by heuristic measures (see section 2.2).

2.2 Search Heuristics

A heuristic is a strategy using some available information to solve a problem that ignores whether the solution can be proven to be correct, but which usually produces a good solution. A usual rule learning heuristic computes its heuristic value from basic properties of the particular rule. Basic properties can be for example the number of positives covered

[2] Noisy data are data that have been input erroneously, measured incorrectly or corrupted in some processing step.

by a rule (true positives) or the number of negatives covered by a rule (false positives), the total number of positive or negative examples or the number of conditions in a rule. The Laplace estimate for a rule r is

$$\frac{p+1}{p+n+2}$$

where p are the positive examples covered by the rule and n are the negative examples. Particularly rules with a high coverage of negative examples are penalized by this heuristic. If the rule does not cover any example the Laplace estimate returns 0.5. In table 5 a short overview of all heuristics applicable in the algorithm's implementation is given.

2.3 Separate-and-Conquer Rule Learning

A big group of rule learning algorithms are those that follow a separate-and-conquer strategy. In Fürnkranz (1999) a variety of separate-and-conquer algorithms was summarized into a single framework and analyzed. A simple separate-and-conquer algorithm first searches for a rule that covers many positive examples of the training data. Rules that cover positive examples but also negative examples are refined until they do not cover any negative example. The covered positive examples are then separated from the training examples and the remaining examples are conquered by subsequently searching rules that cover many positive examples and then adding them to the rule set, until no positive examples remain. In order to avoid overfitting, the requirements that no negative examples may be covered (*consistency*) and all positive examples have to be covered (*completeness*) are relaxed by many separate-and-conquer algorithms (Fürnkranz, 1999). The generic separate-and-conquer rule learning algorithm can be viewed in algorithm 1.

The algorithms are characterized by three dimensions. The first dimension is the *language bias*. It defines in which *hypothesis* or *representation language* the conditions are represented. For example in the weather example above, the conditions are represented by simple selectors that relate an attribute value to one of its domain values. But a separate-and-conquer algorithm might also use more complex conditions. The representation language can be searched for acceptable rules. The *search bias* identifies how the algorithm searches this hypothesis space. It may for example employ a *hill-climbing search*, that considers a single rule and improves it until no further improvement is possible. Another approach is to employ a *beam search* (cf., section 5.7.1) which in contrast to the hill-climbing search does not only consider a single rule for improvement but keeps a (usually small) set of candidates for further improvement. Rules can be either specialized (top-down) or generalized (bottom-up) or both. In the course of searching the best rules, the rule's quality can be measured by several heuristic evaluation functions (cf., section 2.2). The third dimension is the *overfitting avoidance bias*. Some kind of preprocessing or postprocessing is used by several algorithms to generalize rules or rule sets to avoid possible overfitting to the training examples. Preprocessing is performed before or during building the classifier, postprocessing afterwards.

2.4 Pruning

A frequently used method of pre- and postprocessing is pruning. Pruning originates in decision tree learning and refers to removing branches of a decision tree (cf., section 3). The intention of performing pruning is to remove those branches that do not contribute to correct prediction or even counteract it. Pruned trees show a reduced complexity of the final classifier as well as better predictive accuracy. Pruning can either be performed during the learning process to deal with noise or erroneous instances in the training data. It is then called pre-pruning. The tree can also be pruned after processing the training data, and is then called post-pruning. Both techniques are applicable for separate-and-conquer rule learning, too (Fürnkranz, 1997).

```
SeparateAndConquer (Examples)
$Theory = \emptyset$
while Positive (Examples) $\neq \emptyset$
    $Rule$ = FindBestRule(Examples)
    $Covered$ = Cover(Rule, Examples)
    if RuleStoppingCriterion(Theory, Rule, Examples)
        exit while
    $Examples = Examples \setminus Covered$
    $Theory = Theory \cup Rule$
$Theory$ = PostProcess(theory)
return(Theory)
```

```
FindBestRule(Examples)
$InitRule$ = InitializeRule(Examples)
$InitVal$ = EvaluateRule(Examples)
$BestRule =< InitVal, InitRule >$
$Rule = \{BestRule\}$
while $Rules \neq \emptyset$
    $Candidates$ = SelectCandidates(Rules, Examples)
    $Rules = Rules \setminus Candidates$
    for $Candidate \in Candidates$
        $Refinements$ = RefineRule(Candidate, Examples)
        for $Refinement \in Refinements$
            $Evaluation$ = EvaluateRule(Refinement, Examples)
            unless StoppingCriterion(Refinement, Evaluation, Examples)
                $NewRule =< Evaluation, Refinement >$
                $Rules$ = InsertionSort(NewRule, Rules)
                if $NewRule > BestRule$
                    $BestRule = NewRule$
    $Rules$ = FilterRules(Rules, Examples)
return(BestRule)
```

Algorithm 1: Generic separate-and-conquer algorithm SeperateAndConquer (Fürnkranz, 1999). The main part of the algorithm is the while loop that iterates over the training data until all positive examples are covered or the rule set achieved a certain quality (RuleStoppingCriterion). In each iteration a rule provided by FindBestRule is added to the rule set (theory) and the covered examples are removed from the set. In PostProcess postprocessing like pruning is performed. In FindBestRule a list of sorted rules is kept. From this list in each iteration a subset is selected for refinement (SelectCandidates). Each refinement is evaluated and inserted into the list. After this, a subset of the grown list of candidate rules for the next iteration is obtained by applying FilterRules. The best rule, which is the refinement that evaluated best, is returned by FindBestRule.

3 Decision Tree Learning

In this section the method of decision tree learning will be introduced. This is also fundamental for the later introduced lazy decision trees (cf., 4.2) which relate to the eager decision trees described here. Furthermore, basic measures like entropy and information gain will be explained.

Decision tree learning (Breiman et al., 1984; Quinlan, 1986, 1993) is a widely used method in machine learning and is a kind of supervised learning. Figure 5 shows a decision tree according to the weather example in the preceding section, however the whole dataset of 14 instances is considered here. The leaves of the tree represent class labels. Each node corresponds to an attribute of the instances in the dataset and branches to the possible values of the attribute. If the attribute is a numeric one, it needs to be discretized in some way. For example, in figure 5 the numeric attribute $Humidity$ splits into two branches, but could also split into more. The query instance

$$< Outlook = overcast,\ Humidity = 50,\ Wind = FALSE,\ Temperature = hot >$$

would be sorted to the second branch in the node $Outlook$ and therefore be classified as positive, the tree predicts $Sport = yes$ for this instance.

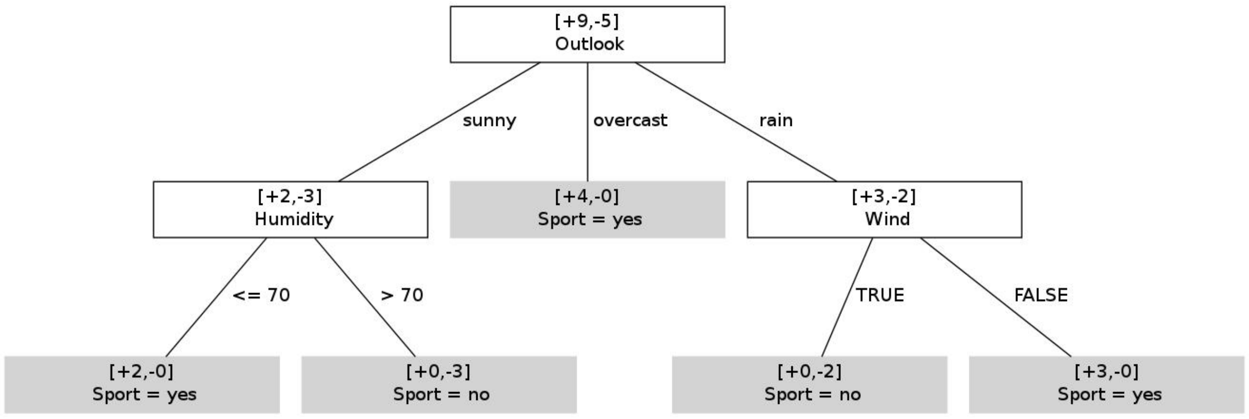


Figure 5: A decision tree that decides if to do sport or not according to weather observations. The two classes of $Sport$ are yes and no. The example is adapted from (Mitchell, 1997). Nodes correspond to attributes and are white, leaf nodes correspond to classes and are gray. The values in square brackets are the number of positive ($Sport = yes$) and negative ($Sport = no$) instances of the dataset covered at the current node.

Decision trees represent a disjunction of conjunctions, the conjunctions are the paths from the root node to the leaf nodes. The decision tree in figure 5 corresponds to the expression

$$(Outlook = sunny \wedge Humidity <= 70) \vee (Outlook = overcast) \vee (Outlook = rain \wedge Wind = FALSE)$$

Important examples for decision tree learning algorithms are ID3 (Quinlan, 1986) and its successor C4.5 (Quinlan, 1993). They search the space of possible decision trees by employing a greedy top-down search. Algorithm 2 shows the ID3 algorithm.

3.1 ID3

The decision tree in figure 5 is just *one* possible decision tree in the space of all decision trees that can be derived from the dataset. But different decision trees might perform differently. For example, a decision tree with longer paths and many duplicate subtrees would perform worse on average. In order to obtain a preferably small and efficient decision tree, it is necessary to choose the order of attributes along the pathes according to some measure of importance. This is the core of ID3: At each node it evaluates each attribute and selects the best attribute for testing (algorithm 2, l.10). How good an attribute is, is determined by the *information gain* measure. Before defining the information gain measure, the definition of a related measure, the *entropy*, is needed.

3.1.1 Entropy

The entropy is a measure of the average information content and is usually measured in bits. A high entropy results in more bits, a lower in less. Here, it characterizes the purity or impurity of an arbitrary set of examples.

```
ID3 (Instances, ClassAttribute, Attributes)
Create a Root node for the tree
if all instances are positive
    return the single-node tree Root, with label = +
if all instances are negative
    return the single-node tree Root, with label = -
if number of predicting attributes = $\emptyset$
    return the single node tree Root, with label = most common value of the target attribute in the instances
else
    $A \leftarrow$ attribute that best classifies instances
    Decision Tree attribute for Root $\leftarrow A$
    for each possible value $v_i$ of $A$
        Add a new tree branch below Root, corresponding to the test $A = v_i$
        $E \leftarrow$ EXAMPLES($A$, $v_i$)
        if $E = \emptyset$
            below this new branch add a leaf node with label = most common target value in the instances
        else
            below this new branch add the subtree ID3 (E, ClassAttribute, Attributes
{A})
return Root
```

Algorithm 2: Decision tree algorithm ID3. *Instances* are the training examples, *ClassAttribute* is the attribute whose value the tree should predict and *Attributes* are other attributes that may be tested by the decision tree. EXAMPLES(A, v_i) is the subset of instances that have the value v_i for A. The *best* attribute is the one with the highest information gain. The returned decision tree will classify all training examples correctly.

Definition 1 *For a set of examples S, a class attribute with c classes and p_i the proportion of S belonging to class i the entropy is*

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{1}$$

According to definition 1 a high entropy occurs, if the classes in the dataset are equally distributed. For example, the class attribute of the decision tree of figure 5 has two values. At node $Wind$ there are two negative (classified as Sport = no) and two positive (classified as Sport = yes) values covered. This results in

$$Entropy([+2,-2]) = -\frac{2}{4} \cdot \log_2(\frac{2}{4}) - \frac{2}{4} \cdot \log_2(\frac{2}{4}) = 1,$$

the highest possible entropy for a dataset with two classes. In contrast, the leaf nodes have an entropy of 0, the lowest possible value for a binary class. Another example is the entropy of the root node $Outlook$, which covers 9 positive examples and 5 negative examples and results in

$$Entropy([+9,-5]) = -\frac{9}{14} \cdot \log_2(\frac{9}{14}) - \frac{5}{14} \cdot \log_2(\frac{5}{14}) = 0.94.$$

When descending the tree from the root node to a leaf, entropy always decreases. So a low entropy is preferable; it means that the classes are distributed differently.

3.1.2 Information Gain

Information gain measures the expected reduction in entropy, when splitting at a certain node. To decide which attribute to use next for branching, the information gains of all remaining attributes are computed. The attribute that scores the highest reduction of entropy in average is selected for the split.

Definition 2 *For a set of examples S and an attribute A with v values, the information gain is defined as*

$$Gain(S,A) = Entropy(S) - \sum_{v \in Vaues(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v) \tag{2}$$

In definition 2, S is the set of examples covered after descending a branch from the parent node, which corresponds to assigning a particular value to the attribute of the parent node. The set S_v is the set of covered examples after assigning the value v to attribute A. For the root node in figure 5 the attribute $Outlook$ was selected because it has the lowest entropy. The highest information gain after branching to $Outlook = sunny$ scores the attribute $Humidity$. It results in

$$Gain(S_{sunny}, Humidity) = Entropy([+2, -3]) - (\frac{2}{5} \cdot Entropy([+2, -0]) + \frac{3}{5} \cdot Entropy([+0, -3]))$$

$$Gain(S, Humidity) = 0.97 - \frac{2}{5} \cdot 0.0 - \frac{3}{5} \cdot 0.0 = 0.97$$

3.2 C4.5

C4.5 is the enhanced variant of ID3. In contrast to ID3, it can also handle continuous attributes. The algorithm creates a threshold and then splits the list into those whose attribute value are above the threshold and those that are less than or equal to it. It is able to cope with missing attributes as well. They are not considered in gain and entropy calculations. Moreover, it can handle attributes with differing costs. After the creation of a probably overfitted decision tree, the tree is pruned. C4.5 goes back through the tree and tries to remove branches that do not help by replacing them with leaf nodes. A decision tree can also be viewed as a set of rules. For each path from the root node to a leaf node a rule is created and added to the rule set.

4 Lazy Learning

This section describes the concept of lazy learning and introduces two different lazy learning algorithms. Separate-and-conquer rule learning algorithms and decision tree learning algorithm have something in common: They are both *eager* learning methods. That is at first they learn a classifier that is used after training to classify all new instances. In contrast, lazy learning, or instance based learning, is a learning method that delays the building of the classifier until a query is made to the system. So the hypothesis is built for each instance separately and on request. The training data is kept in memory and is utilized by each query instance individually.

For computation lazy learning will require less time during training than eager methods, but more time for the classification of a query instance. Contrary, eager learning methods use all available training examples to build a classifier in advance that is later used for classification of all query instances.

The key advantage of lazy or instance based learning is that instead of once and for the whole training data, the target function will be approximated locally for every query instance. This allows a different target function for each of the query instances. Because of this lazy learning systems are able to simultaneously solve multiple problems and deal successfully with changes in the instances used for training. Lazy learning systems can simply alter an instance, store a new instance or throw an old instance away.

In contrast, the disadvantages of lazy learning include the large space requirement to store the entire training dataset and lazy learning methods are usually slower to evaluate, although this often goes with a faster training phase. In the following, two approaches of lazy learning will be viewed in more detail.

4.1 k-nearest Neighbor Algorithms

A very basic lazy learning method is the k-nearest neighbor algorithm (Cover and Hart, 1967; Aha and Kibler, 1991). The k-nearest neighbor algorithm views instances as points in the n-dimensional space R^n according to n features. For classifying an instance, the algorithm considers the classes of its k nearest neighbors. The most common class amongst its neighbors is then selected to be the correct class for the instance. The distance between a training example and the instance to be classified is usually measured by the Euclidean distance (definition 3).

Definition 3 *For two instances x and y with the i^{th} attribute's value $a_i(x)$, the distance is defined as*

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(a_i(x) - a_i(y))^2} \tag{3}$$

Because the focus of this thesis is on classification learning the k-nearest neighbor algorithm is only considered for discrete valued target functions (figure 6 and algorithm 3). But it similarly works for continous valued target functions in regression learning. While in classification learning the most frequent class of the k neighbors is selected, in regression learning the values of the neighbors are averaged.

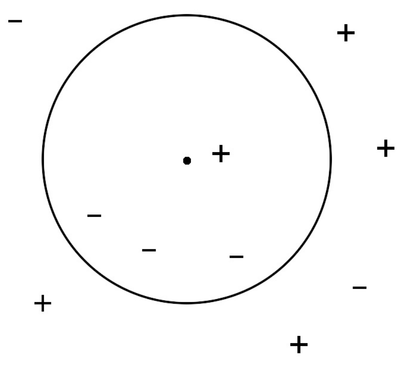


Figure 6: Example of k-nearest neighbor classification. The point in the center represents the test instance and should be classified either as positive ($+$) or negative ($-$). If $k = 1$ it is assigned to the positive class because the closest training example is a positive one. If $k = 4$ it is assigned to the negative class because there are 3 negative examples and one positive example.

```
kNN (Instance, T)
Let $x_1 ... x_k$ be the $k$ instances of $T$ that are nearest to Instance
return
    $\arg\max_{c \in C} \sum_{i=1}^{k} w_i \delta(c, f(x_i))$
```

Algorithm 3: The classification algorithm of the k-nearest neighbor algorithm kNN. *Instance* is the test instance, T is the set of training data, C is the set of class values, c a specific class value . Function $f(x_i)$ returns the class value of instance x_i, $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise. In the refined algorithm instead of setting the weight to $w_i = 1$, the weight of an instance's contribution depends on its distance according to equation 4.

A refinement of the algorithm is to assign a weight to the neighbors according to their distance to the query instance. The closer the neighboring instance is to the query instance, the more it influences the classification. In algorithm 3 the weight is thus set to

$$w_i = \frac{1}{d(Instance, x_i)} \tag{4}$$

4.2 Lazy Decision Trees

With the lazy decision tree algorithm Friedmann introduced another possibility of classifying instances in a lazy way (Friedman, 1996). As described in section 3, common decision tree algorithms construct one decision tree from a set of training data to be used for the whole set of instances to be classified. In contrast, the lazy decision tree algorithm builds a separate decision tree for each instance that should be classified. Rather than a whole tree, there is just a single path constructed; from the root to the labeled leaf node. Algorithm LazyDT shows the generic lazy decision tree algorithm. It expects a yet unclassified instance and a set of training examples. When the recursion ends, the algorithm returns a classification for the instance. It is important which test is selected for the split at each internal node. For this reason, the node the instance would branch to, that maximally decreases the entropy, is selected as test. More precisely, information gain like in eager decision trees is used but with normalized class probabilities. The normalization is necessary to avoid negative information gains or zero information gain for yet important splits, that otherwise could occur. This is due to the use of tests at nodes that are based on the test instance rather than on class distributions (cf., Friedman (1996)).

```
LazyDT (Instance, T)
if all instance in $T$ have the same class $c$
    return class $c$
else if all instances in $T$ have the same attribute values
    return majority class in $T$
else
    select a test $X$ and let x be the value of the test on the instance $I$
    assign the set of instances with $X = x$ to $T$
    LazyDT (Instance, T)
```

Algorithm 4: The generic lazy decision tree algorithm LazyDT

The motivation for using a lazy decision tree is to take advantage of the additional information given by a single instance and to clear some problems that occur in decision tree algorithms. Those problems include replication (duplicate subtrees) and fragmentation, i.e. partitioning of the data into smaller fragments (Pagallo and Haussler, 1990). The problem of fragmentation is shown in figure 7. Instances with attribute *Temperature* are fragmented. The subsequent tests on *Temperature* have lower signigicance because they are based on fewer instances.

Because in regular decision trees the best split on average is chosen at each node there might be a disadvantageous branch for a particular test instance. However, a lazy decision tree will select a better test with respect to the particular instance. The avoidance of splitting on unnecessary attributes may thereby produce a more accurate classifier and shorter paths. Next, the problem of replication will be exemplified. In fact, in the decision tree example above the tree is fragmented but may not contain any duplicate subtrees. Though the decision tree of the disjunctive concept $(A \wedge B) \vee (C \wedge D)$ will contain a duplicate subtree. This can be either $(A \wedge B)$ or $(C \wedge D)$ as shown in figure 8. In a lazy decision tree there is just one path and thus no duplicate subtrees.

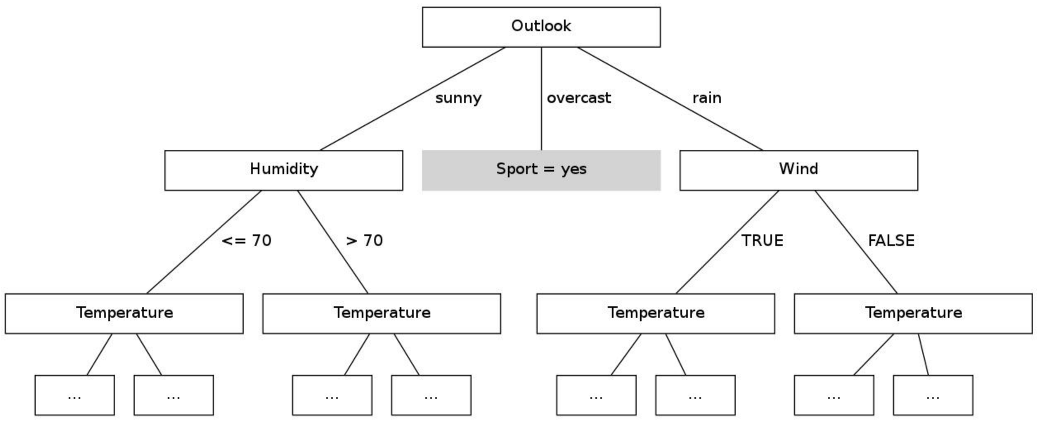


Figure 7: Fragmentation problem in eager decision tree. For an instance that has missing values for $Outlook$ and $Wind$ the tests at these nodes are irrelevant and lead to lower significance of the test on $Temperature$.

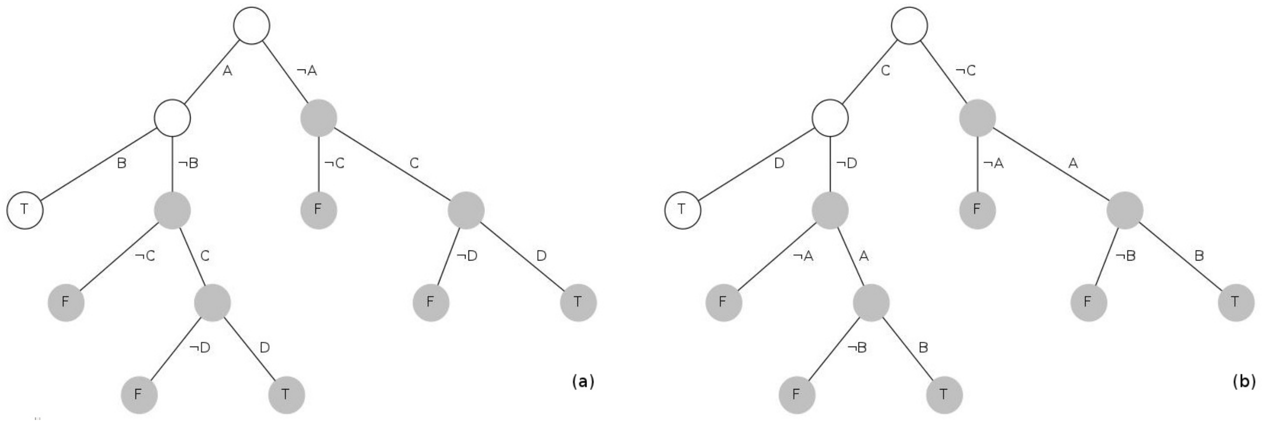


Figure 8: Duplicate subtrees for $(A \wedge B) \vee (C \wedge D)$ are highlighted grey. In (a) the subtree for $(C \wedge D)$ duplicates, in (b) the one for $(A \wedge B)$ does.

Another problem that is resolved by the usage of lazy decision trees is the way of handling missing attributes. While the usual decision tree algorithms provide complicated mechanisms to deal with missing attributes in instances, lazy decision trees just never branch on a missing value in the instance.

Compared to the decision tree algorithm C4.5 (Quinlan, 1993), Friedman (1996) measured a similar performance for the lazy decision tree algorithm - neither of the two algorithms could outperform the other on all datasets. Some improvement in performance was achieved by applying boosting[3] to the algorithm (Fern and Brodley, 2003). Another variant of the algorithm, the lazy option tree algorithm, uses a bagging[4] approach (Margineantu and Dietterich, 2001).

Like eager decision tree algorithms the lazy decision tree algorithm might also be used to derive rules from the tree. This approach may be another way of learning rules a lazy way but further analysis goes beyond the scope of this thesis and is thus just briefly described. One rule for each instance to be classified could be obtained if the path to the leaf node was transformed to a rule body consisting of the attribute-value pairs along the path while the class of the leaf node might be used as rule head. Let for example the instance

$$\langle Outlook = rain,\ Windy = true,\ Humidity = 60,\ Temperature = ?\rangle$$

be classified classied as $Sport = No$ by the lazy decision algorithm. Depending on then training examples a possible lazy decision tree (that is actually a path from the root to the leaf) could be:

$$Humidity = 60 \longrightarrow Windy = true$$

3 Boosting algorithms combine a set of weak classifiers to a single strong classifier.

4 In bagging algorithms several datasets are generated from the training data and then a classifier is learned on each of them. Afterwards the classifiers are combined by averaging to a possibly better classifier.

The missing attribute $Temperature$ is ignored by the algorithm. Since the tests at the nodes are derived from the test instance, the path might also be shorter. In the example the test on $Outlook$ was not necessary. The decision tree constructed by the example instance would consequently compose the rule

$$Sport = No \text{ :- } Humidity = 60, Windy = true$$

Unfortunately there was no implementation of the lazy decision tree algorithm publicly available and thus experiments in this thesis do not consider employment of this algorithm.

5 Lazy Rule Learning

Yet an algorithm for explicitly learning rules in a lazy way has not been published. While lazy decision trees could be used for this task (see section 4.2, Lazy Decision Trees) the lazy decision tree algorithm is not explicitly designed to produce one rule per instance to be classified. For this reason a new algorithm for rule learning is proposed in this section. The main hypothesis is, that by improving the heuristic value of rules that consist of conditions provided by a query instance, a rule that classifies the test instance correctly will emerge.

5.1 Algorithm LazyRule

The basic algorithm is rather straightforward and easy comprehensible. It starts with one outer loop that iterates over all classes available in the dataset (algorithm LazyRule). Thus the algorithm performs a one-against-all binarization. For each class value the procedure RefineRule is called with an empty rule as parameter. Each call will then return a rule whose head is the class attribute to the current class value. That means there is one rule for each class value. From this set the one with the highest heuristic value will be returned as rule to classify the instance. In some cases it may happen that for two or more classes exactly the same heuristic value is returned. From such rules the one that covers more instances in the whole training data is chosen, i.e. the majority class is preferred. In figure 9 a list of rules learned this way is shown. Each rule corresponds to a query instance and was learned on all instances of the dataset except the instance to be classified.

```
LazyRule (Instance, Examples)
InitialRule = $\emptyset$
BestRule = InitialRule
for Class $\in$ Classes
    Conditions $\leftarrow$ PossibleConditions(Instance)
    NewRule = RefineRule (Instance, Conditions, InitialRule, Class)
    if NewRule > BestRule
        BestRule = NewRule
return BestRule
```

Algorithm 5: LazyRule (*Instance*, *Examples*)

The procedure RefineRule (algorithm 6) constitutes the centerpiece of the algorithm[5]. After the procedure is called, the at first empty rule is improved by adding the condition that improves the rule most.

Adding conditions is repeated until either no further improvement is noticed or no conditions to add are left. A rule is assumed to have improved if its heuristic value with respect to the current class value became higher after adding a condition. The set of possible conditions encompasses all conditions that can be derived from the instance to be classified. Conditions are constructed from the attribute-value pairs of the instance (PossibleConditions, algorithm 7). If the instance has a missing value for an attribute, no condition will be added for this attribute.

```
RefineRule (Instance, Conditions, Rule, Class)
if Conditions $\neq \emptyset$
    BestRule = Rule
    BestCondtion = BestCondition (Rule, Conditions)
    Refinement = Rule $\cup$ BestCondtion
    Evaluation = EvaluateRule (Refinement)
    NewRule = <Evaluation, Refinement>
    if NewRule > BestRule
        BestRule = NewRule
        RefineRule (Instance, Conditions \ BestCondition, NewRule, Class)
return BestRule
```

Algorithm 6: RefineRule (*Instance*, *Conditions*, *Rule*, *Class*)

[5] The method RefineRule not to be confused with the one in the generic separate-and-conquer algorithm (algorithm 1). Here it returns a *single* rule instead of a set of rules

```
PossibleConditions (Instance)
Conditions ← $\emptyset$
for Attribute $\in$ Attributes
    Value = AttributeValue (Attribute, Instance)
    if Value $\neq \emptyset$
        Conditions = Conditions $\cup$ {(Attribute = Value)}
return Conditions
```

Algorithm 7: PossibleConditions (*Instance*)

```
play = yes :- windy = FALSE, humidity >= 70, humidity < 88.
play = yes :- humidity < 90.5, humidity >= 85.5.
play = yes :- outlook = overcast.
play = yes :- temperature >= 66.5, temperature < 70.5.
play = yes :- temperature >= 66.5, temperature < 70.5.
play = yes :- humidity >= 65, humidity < 82.5.
play = yes :- outlook = overcast.
play = yes :- temperature < 77.5, temperature >= 71.5.
play = yes :- temperature < 70.5, temperature >= 66.5.
play = yes :- humidity >= 70, humidity < 82.5, windy = FALSE.
play = yes :- humidity < 82.5, humidity >= 70.
play = yes :- outlook = overcast.
play = yes :- outlook = overcast.
play = yes :- temperature < 71.5, temperature >= 66.5.
```

Figure 9: A set of rules learned for the weather dataset with numeric and missing values. Each rule corresponds to a query instance. Here the $Laplace$ heuristic was employed.

The set of instances can be decreased after adding a condition to the rule. All training examples that are not covered by the rule may then be omitted in the following iterations of RefineRule. Thus only the set of covered examples is considered in the next iteration where the rule is refined. In contrast to separate-and-conquer rule learning algorithms, LazyRule does not try to cover all positive instances of the training dataset, but rather tries to cover the examples according to the chosen heuristic by a rule constructed of conditions derived from the test instance. A separate-and-conquer algorithm tries to cover examples by learning a set of rules, while the lazy rule learning algorithm only refines a single rule. The algorithm shown here can just handle nominal attributes. Later the algorithm is extended to handle numeric attributes as well (see section 5.2, Numerical Attributes).

In the current algorithm a hill climbing search is employed. Within each RefineRule call merely the one condition that performs best will be added to the rule. Perhaps a combination of several conditions would lead to a higher heuristic value than one single condition. Depending on the heuristic this might be an improvement for the algorithm. A modification of the algorithm that incorporates a beam search will be described later on (see section 5.7.1, Beam Search).

5.2 Numerical Attributes

Nominal attributes extracted from the instance to be classified are handled straightforward by the algorithm. The attribute-value pair of the instance is already a condition that can be used.

Somewhat more complicated is the construction of a condition when the current attribute value of the instance is a numeric one. For example, consider a dataset, where one attribute determines the size of a person. Let the value of a numeric attribute in an instance be $Size = 161.5$. Adding this attribute-value pair as a condition to the classifier would not make much sense since the condition probably will cover no instance in the training data (it would cover the test instance, but usually the test instance is not included in the training data). So how does the algorithm treat those numeric attributes? When processing a numeric attribute two conditions will be inferred. These two conditions determine the lower and the upper bound of the interval in which the class value of the instance is assumed. These conditions will be called *interval conditions* in the following.

The method RefineRule is called for each class of the dataset. To find the bounds from which the conditions will be derived, the algorithm has to find the closest examples of the training set around whose class values differ from the one that was assumed for the instance to be classified. First, the instances are put into an ordered list according to the attribute. For the lower bound, the next lower example in the training examples, that has a different class value has to be found. Thus, the closest instance, that has a different class value, is searched by going down the list, starting from the test instance's value. The assumed class switch is then calculated by averaging the two values and thus used for the interval condition. Adding two associated conditions instead of only one resulted in a higher accuracy.

For the example with $Size = 161.5$, the closest lower instance from the training data with a different class could have $Size = 143.4$ while the lowest example within the interval of same class values that the example is assumed to be in, could have $Size = 148.7$. Hence the lower bound would result in $\frac{(143.4+148.7)}{2} = 146.05$ which leads to the condition $Size \geq 146.05$ for the lower bound. Similarly, the condition for the upper bound is obtained by looking for the next higher example that differs in its class value and averaging both attribute values.

Since the algorithm iterates over all possible class values it will generate an interval conditions for each of the class values.
In the following example the instance is to be classified according to a dataset that comprises the class values $child$, $adolescent$ and $adult$. The algorithm will iterate over those three class values and with it generate interval conditions for each of them. Figure 10 depicts the interval for the case when the value $adolescent$ is assumed, i.e. REFINERULE was called for the class value $adolescent$. In figure 11 the interval is much smaller if presumed the class $adult$. In this interval only the instance that is to be classified will be contained but no instance of the training set. The resulting conditions of this interval will cover no example of the training set and will certainly not be chosen to improve the rule. In this example the class $child$ would also result in zero coverage. Examples for conditions derived from numerical attributes are shown figure 9.

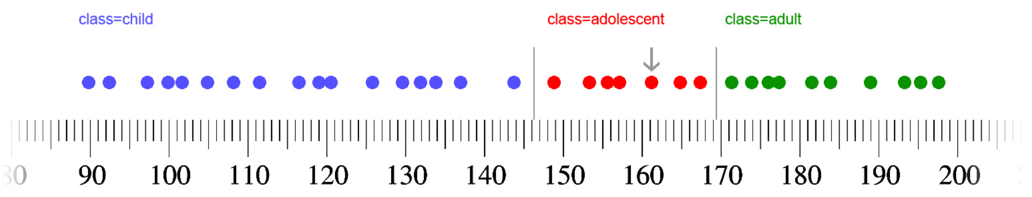


Figure 10: The instance to classify shows $Size = 161.5$ and is assumed to be classified as $adolescent$. Accordingly the class changes result in $Size = 146.05$ and $Size = 169.4$. This leads to the interval conditions $Size \geq 146.05$ and $Size \leq 169.4$.

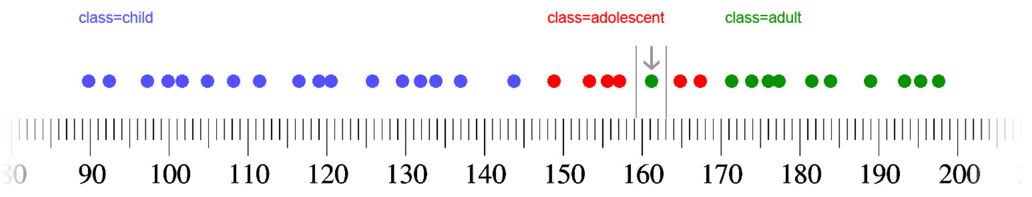


Figure 11: The instance to classify again shows $Size = 161.5$ but is assumed to be classified as $adult$. Accordingly the class changes this time result in $Size = 159.1$ and $Size = 163.0$ and leads to the interval conditions $Size \geq 159.1$ and $Size \leq 163.0$. This interval would cover no instance of the training dataset.

5.3 Search Heuristics

As previously mentioned, the algorithm's rule evaluation is based on search heuristics. Search heuristics used for lazy classification thus ought to find the correct class for the query instance in the majority of cases. The value of the particular search heuristic determines whether the condition should be added to a rule or if refinement should halt (method REFINERULE). The search heuristic can easily be changed for LAZYRULE. As the algorithm is built on the SECO-framework (Janssen and Fürnkranz, 2010a) it provides all heuristics that can be found in the framework too. A list of heuristics available for the algorithm is shown in table 5.

In contrast to heuristics for decision tree learning, which evaluate the average quality of a number of disjoint sets defined by each value of the tested attribute, heuristics in rule learning only evaluate the set of examples covered by the particular rule (Fürnkranz, 1999).

In preceding leave-one-out cross-validation (cf., 7.2.2) the Laplace heuristic performed notably better than other heuristics for several datasets. Since the reason for this grave differences was not really obvious at first the algorithm was designed to support a variety of heuristics in order to find possible other candidates and to check the assumption that there exists one or few superior heuristics, i.e. heuristics that perform significantly better than others for the algorithm. In the later evaluation process the algorithm will be examined with different heuristics applied.

5.4 Bottom-Up Approach

It seemed somewhat more intuitive and natural to learn a rule from an instance in a bottom-up way. The idea was to start from a rule whose body is constructed from all attributes and values of the instance to be classified instead of starting with an empty rule like in the LazyRule algorithm above. Hence the bottom-up algorithm starts with a maximally specific rule which most likely would not cover any instance from the training instances (since it is derived from the training instance). Algorithm InitializeRule shows the initialization of this rule.

The rule is then repeatedly generalized with the Generalize method until it covers at least one positive example (see algorithm LazyRule-BottomUp). Similar to RefineRule in the top-down algorithm Generalize returns a rule. But instead of adding a condition the rule without the condition whose removing most improved the heuristic value of the rule is returned here. Furthermore Generalize is not recursive but is called as long as there is no positive coverage yet or the rule's body became empty.

```
LazyRule-BottomUp (Example, Instances)
Positive ← ∅
BestRule = InitializeRule (Example)
for Class ∈ Classes
    while Positive = ∅
        NewRule = Generalize (Rule, Instances)
        Positive = Covered (Rule)
    if NewRule > BestRule
        BestRule = NewRule
return BestRule
```

Algorithm 8: LazyRule-BottomUp (*Example*, *Instances*)

```
InitializeRule (Example)
Rule = ∅
for Attribute ∈ Attributes
    Value = AttributeValue (Attribute, Example)
    if Value ≠ ∅
        Rule = Rule ∪ {(Attribute = Value)}
return Rule
```

Algorithm 9: InitializeRule (*Example*)

Another idea was to generalize in all directions, based on the initial rule. This results in a tree whose nodes split on all remaining conditions, as long as no positive coverage is achieved, which is a leaf in the tree. Then the class attribute with the highest true positive coverage as rule head was chosen. From this ruleset the rule with the highest heuristic value was then returned as best rule to classify the instance.

The bottom-up approaches resulted in a higher average rule length and performed worse compared to algorithm LazyRule in leave-one-out cross-validation. They were therefore not regarded in further evaluation. Moreover greedy generalization, which actually is used in these bottom-up concepts, is impossible if the instance to be classified differs in more than one attribute value from its nearest neighbor (Fürnkranz, 2002).

5.5 Search Space

The presented lazy rule learning algorithm tries to find the combination of conditions inferred from the instance to be classified that scores the highest heuristic value with respect to the currently assumed class. It does so by subsequently adding a condition to the body of the rule that improves the rule's heuristic value until there is no improvement anymore. By merely looking for the next improving condition and not concerning combinations of candidate conditions it thus performs a hill climbing search. Usually, it will not use all the conditions provided by the query instance.

In each iteration of RefineRule after adding an improving condition the set of remaining conditions will be decreased by 1. Let n be the number of conditions derived from the query instance and u the number of already used conditions. In the worst case, which means adding the whole set of conditions (which in fact will not happen if the dataset does not contain duplicates), the algorithm will check

$$Combinations(n) = \sum_{u=1}^{n} n - u = \frac{n\,(n+1)}{2} = \sum_{u=1}^{n} u \tag{5}$$

different condition-combinations (i.e. rule bodies) in order to find out that with adding the last one it obtained the best rule. However, the whole search space is much bigger. The amount of possible rule-bodies that may be derived from the test instance is constructed as follows: Start with an empty rule. Then next rule is constructed by adding an arbitrary condition. For each remaining condition, perform the following. Add all the combinations of it with each yet constructed rule-body (including the empty one) to the search space. Thus the amount doubles with each addition of an attribute. The number of all possible combinations of conditions is 2^n. In this way constructed rule-bodies will be without duplicates. Since different permutations of conditions combined to a rule's body yield the same heuristic value just one permutation needs to be considered. For c classes there are $c \cdot 2^n$ different rules.

For datasets with 20 attributes that makes $\sum_{u=1}^{20} 20 - u = 210$ rules to evaluate for each class in the worst case compared to $2^{20} = 1048576$ rules if each of the possible conditions is checked for one class.

Due to this big gap between regarded rule-bodies in the basic algorithm and possible rule-bodies, it seemed interesting to find out, if there was a significant improvement when considering more instances in the search. Hence, another variant of the algorithm, that employs a beam search, is introduced in section 5.7.1. The evaluation part of this thesis allows a closer look on effects of changing the set of condition-combinations considered by the algorithm.

5.6 Complexity

The outer loop iterates over all class attributes c. For datasets without missing values in the worst case the iterations of the RefineRule method add up to $\frac{(a+1)\,a}{2}$ for a attributes (see equation 5). In the case of missing attributes instead of all a attributes only the number of attributes that have a value assigned in the instance to be classified are considered. An upper bound for both is $O(c \cdot a^2)$ rule evaluations per query instance.

Until now, only the classification of a single query instance was considered. Classifying the whole dataset containing i instances needs maximally

$$O(c \cdot a^2 \cdot d) \tag{6}$$

rule evaluations.

In the first call of RefineRule, the whole set of training instances is used for evaluation. In the next iterations, the condition that is added to the rule[6], decreases the amount of training instances to consider in the subsequent iterations rapidly. However, for each class initially the whole set is used.

Lazy learning algorithms afford a fast classification for a test instance. Because of this it is desirable to improve the algorithm's runtime. One way to do so is to limit the number of considered attributes using a preselection as described in section 5.7.2. However, since the algorithm usually produces short rules (see table 6), the rule refining halts long before processing all possible conditions. Although the training dataset shrinks rapidly after the first RefineRule call, further optimization might be achieved by instead of using the whole training set for rule evaluation to just consider a subset for training as described in section 5.7.3 and section 5.7.4.

5.7 Possible Improvements

In the following, two magnitudes that may be improved for the algorithm will be discussed. Firstly, a possible improvement of accuracy will be suggested.

When the basic algorithm compares the currently best rule with the currently best rule added one condition it definitely constructs a better rule if the extra condition is an improvement. But is this single best rule always the best choice? One can imagine a condition is added after which no further improvement is possible, i.e. that all remaining conditions could not improve the rule anymore. It might happen while this condition is improving the rule indeed, there exists a combination of two conditions which each on its own would not gain a higher heuristic value than the added single condition. But in combination they would achieve a higher heuristic value. The choice of adding this pair of conditions would thus be a better one than adding the single rule. In light of these considerations another approach to improve accuracy was reviewed, namely a beam search.

[6] For a numeric attribute actually two conditions are added.

While a higher accuracy is certainly desirable, a serious drawback of the current algorithm is its high computation time for classifying a single instance when the datset has many classes or attributes and contains many training instances. This is especially because for each class in the first call of RefineRule the initially empty rule is evaluated on the entire set of given training instances. Additionally, the initial rule refined by one of the derived conditions to the rule's body has to be evaluated on the entire set of training instances. Since one wants to consider all classes for classification, only the amount of instances or the number of attributes remain to be considered for improving the execution time. For this reason, one approach that preselects attributes before calling RefineRule is reviewed for the algorithm. On the other hand, the set of instances actually used for classification by LazyRule can be reduced. While in separate-and-conquer rule learning a classifier is learned on the whole training data *once*, in lazy learning the classification happens for each test instance separately. If the whole set of available training instances is used for each test instance, the execution time of the algorithm depends also on the size of the training data and can be reduced by using less instances for training. In the end of this section, two possibilities of how to obtain a smaller training set are described.

5.7.1 Beam Search

Beam search is a heuristic search algorithm that explores a graph by expanding the most promising node in a limited set of nodes and is an optimization of best-first search[7] that reduces its memory requirements. How this beam search works for algorithm LazyRule is explained in the following. One can imagine improving a rule in the basic algorithm by RefineRule like descending along a single branch in a search tree of all possible rules from the root (the empty rule) to a leaf node (a node at which no further improvement by adding a condition is possible). A hill climbing search, as used in the basic algorithm, often just finds a local maximum. In contrast, considering a *beam* of the b most promising branches will consider b leaf nodes. In the beam search approach the most promising rules are kept in a memory which can hold b rules. These rules are kept in memory in descending order. The following steps are iterated until for each rule in memory there is no better rule found or all remaining conditions were added.

First, iterate over all b rules in the memory. In each iteration, generate candidate rules, *e.g.* all rules that are obtained by adding each of the yet unused conditions. Compare each candidate rule with the rules in the memory. If the current candidate rule evaluates better than one of the rules in the memory, insert the candidate rule in the right place and remove the last rule from the memory.

This approach requires some more operations for handling the memory but altogether it lies within the dimension of b times the maximum time required to decrease to a leaf node. This is because a branch in the memory can only be replaced by another branch that is obtained by a branch that already was in the memory before.

For several datasets this approach resulted in a notable improvement. For example with a beam size of 2 it resulted in more than 5% improvement for the dataset *cleveland-heart-disease.arff* and about 8% for a beam size of 4 in leave-one-out cross-validation (cf., 7.2.2). However, increasing the beam size further did not result in noteworthy better results. Combined with some preprocessing (cf., section 5.7.2) to compensate the higher computation costs, this approach might be an improvement for some datasets.

5.7.2 Preprocessing and Postprocessing

It is desirable to improve the algorithm's performance by some kind of pre- or postprocessing. The presented lazy rule learning algorithm is neither a decision tree learning algorithm nor can it be assigned to the separate-and-conquer rule learning algorithms (Fürnkranz, 1999) and therefore is not qualified for pruning. However, other kinds of pre- or postprocessing could be of note here.

Removing conditions from a rule that ought to classify the query instance in postprocessing will not have any effect on the correctness of the classification but will increase execution time. When removing conditions from a rule's body the positive coverage will increase. But the classification for the query instance will at all times stay the same since the rule to classify may only consist of the conditions taken from the query instance and the rule's head stays unchanged.

Somewhat more promising is to apply a kind of preprocessing to the algorithm. In (Pereira et al., 2011a) a method of data preprocessing for lazy classification is proposed. Lazy attribute selection performs a preselection of attributes which the particular lazy learning algorithm should consider for classification. The instance's attributes are ordered by their entropies (see definition 1, p. 10) from which the r best are selected for further processing. This method was applied to the k-Nearest-Neighbor algorithm (cf., section 4.1). Depending on the parameter r for determining the number of attributes to consider, lazy attribute selection was able to improve the accuracy of the classification significantly for several datasets. Additionally, a metric to estimate when a specific dataset can benefit from the lazy attribute selection

[7] Best-first search is a search algorithm that explores a graph by expanding the most promising node chosen according to a heuristic evaluation function.

was proposed in (Pereira et al., 2011a) and applied in (Pereira et al., 2011b). This way of preprocessing is also applicable to the proposed lazy rule learning algorithm and might achieve an improvement in performance and accuracy. However, within the scope of this thesis a further examination was not possible and this approach was not considered for the algorithm.

5.7.3 Stratified Subsets

The preceding section suggested a possibility to improve the algorithm's execution time by reducing the amount of attributes to consider. Another way to achieve a faster algorithm is to reduce the amount of training instances actually considered for training. If desired that the reduced set of training data is similar to the original training data, the class distribution of the original training data can be maintained, when a *stratified* subset of the training data can be used (cf., 7.2.1).

5.7.4 Nearest Neighbor Subsets

If one wants to reduce the amount of training instances given to LazyRule, another solution emerges. Instead of keeping the dataset's characteristics by using a stratified subset, it makes sense to already filter out useless instances, i.e. instances, that most likely will not contribute to a correct classification of the test instance. In other words, it seems promising to apply the lazy rule learning algorithm to a set of instances that are similar to the test instance. Of these nearest neighbors (cf., 4.1) either a fixed number of k nearest neighbors (see section 4.1) or fraction may be used. But when using a particular percentage of the training data the execution time of the algorithm still depends strongly on the size of the dataset. When using the k nearest neighbors though, the rule is learned on a set of fixed size and the dataset's instance count does not affect the computation time so much. Later it will turn out that the lazy rule learning algorithm combined with a nearest-neighbor algorithm performs best (cf., 7.5.3).

6 Implementation

According to the pseudo-code shown in section 4 the algorithm was implemented in Java. It is built on the the SeCo-framework (Janssen and Fürnkranz, 2010a). The algorithm basically supports two types of usage: First, within the framework to employ a leave-one-out cross-validation (see figure 12 and figure 13). Second, it may be used within the Weka suite (Hall et al., 2009) in order to utilize Weka's evaluation functionality.

6.1 SeCo-Framework

The SeCo-framework (derived from Separate and Conquer) is a framework for rule learning developed by the Knowledge Engineering Group at the TU Darmstadt. The framework's core is a general separate-and-conquer algorithm whose components are configurable. By using building blocks to specify each component it builds up a configuration for a rule learner. The requirement for an algorithm to be added to the framework is that it employs a separate-and-conquer strategy and uses a top-down or bottom-up approach. It already provides a variety of separate-and-conquer algorithms.

Although the lazy rule learning algorithm is implemented within the SeCo-framework and takes advantage of several components of this rule learning framework it should not be viewed as a part of it. Unlike the algorithms contained in the SeCo-framework the lazy rule learning algorithm cannot be fitted into a separate-and-conquer algorithm. The included evaluation functionality is tied to eager learning algorithms. For this reason, LazyRule can not be compared to the framework's algorithms directly. However, when a more general interface will be supplied in the future by the evaluation component of the framework, the proposed lazy learning algorithm can easily implement it and thus be compared to the various separate-and-conquer algorithms. Anyhow leave-one-out cross-validation (cf., 7.2.2) is yet possible to be performed directly inside the framework. An example for validation results after running the algorithm in the framework can be viewed in figure 12 and figure 13.

```
class = Iris-virginica :- petallength < 6.2, petallength >= 5.1.  [34|1] Val: 0.946
class = Iris-virginica :- petalwidth >= 1.8, petalwidth < 2.15.  [27|1] Val: 0.933
class = Iris-virginica :- petalwidth < 2.2, petalwidth >= 1.8.  [30|1] Val: 0.939
class = Iris-setosa :- petallength >= 1, petallength < 2.45.  [49|0] Val: 0.98
class = Iris-setosa :- petallength < 2.45, petallength >= 1.  [49|0] Val: 0.98
class = Iris-setosa :- petalwidth >= 0.1, petalwidth < 0.8.  [49|0] Val: 0.98
class = Iris-versicolor :- petalwidth < 1.4, petalwidth >= 0.8.  [34|1] Val: 0.946
class = Iris-versicolor :- petallength < 4.5, petallength >= 2.45.  [35|1] Val: 0.947
```

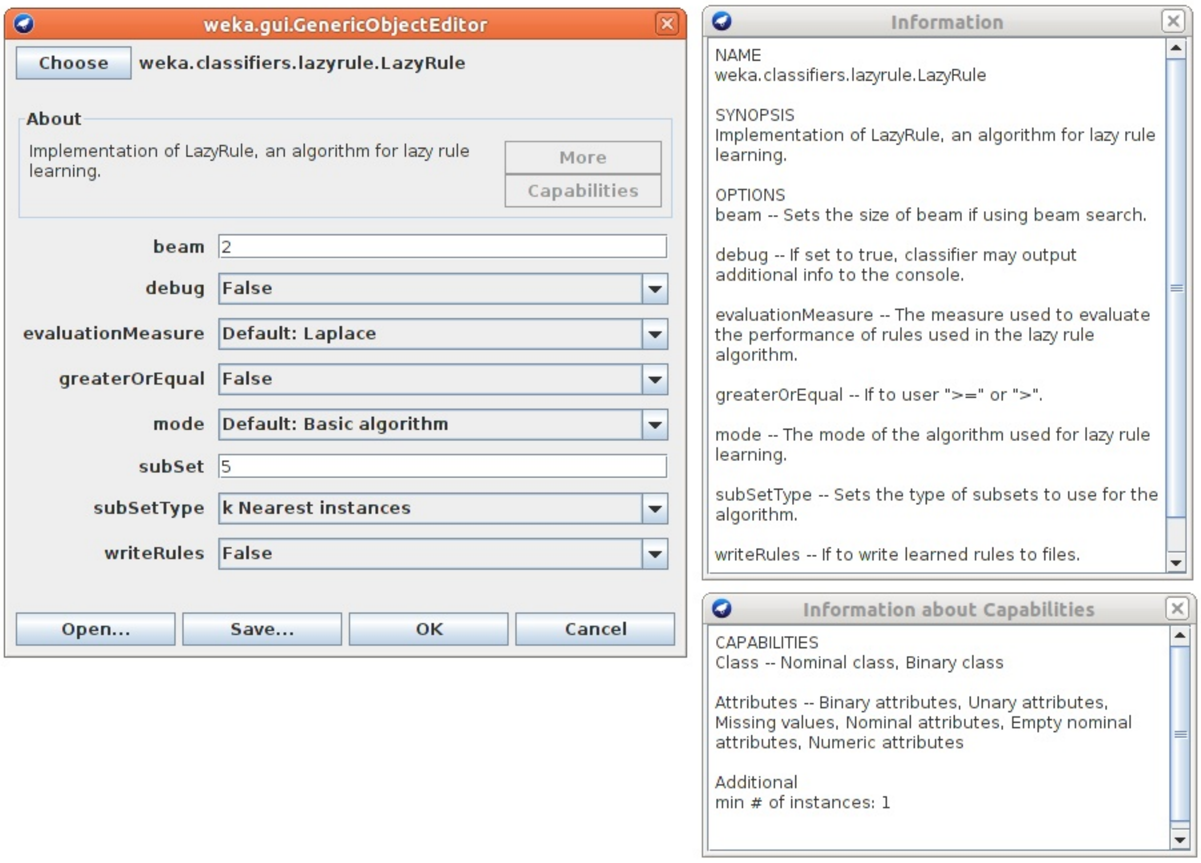
Figure 12: Excerpt of the output of the 150 learned rules by the algorithm employing leave-one-out cross-validation for the dataset *iris.arff* using the $Laplace$ estimate. The values in square brackets are the number of true and false positives followed by the rule's heuristic value.

```
Number of Rules:                   150
Number of Conditions:              424
Number of empty Rule Bodies:       0
Number of Multiple Rules Fired:    3
Referred Attributes:               212
Percent of Empty Rule Bodies:      0%
Percent of Multiple Rules Fired:   2%
Average rule length:               2.83

Time required for building model:  0.61 sec

Correctly classified:              143/150 95.33%
```

Figure 13: Output for the algorithm after employing leave-one-out cross-validation for the dataset *iris.arff* using the $Laplace$ estimate. The number of rules corresponds to the size of the dataset.

6.2 Weka Interface

Weka (Waikato Environment for Knowledge Analysis) is a popular software suite for machine learning (Hall et al., 2009). It is written in Java and developed at the University of Waikato, New Zealand. It supports several standard data mining tasks and contains a variety of learning algorithms. Weka allows to easily evaluate learning algorithms. The algorithm implements an interfacte for Weka. Figure 14 shows how the algorithm appears in the Weka GUI.



weka.classifiers.lazyrule.LazyRule

Implementation of LazyRule, an algorithm for lazy rule learning.

beam	2
debug	False
evaluationMeasure	Default: Laplace
greaterOrEqual	False
mode	Default: Basic algorithm
subSet	5
subSetType	k Nearest instances
writeRules	False

NAME
weka.classifiers.lazyrule.LazyRule

SYNOPSIS
Implementation of LazyRule, an algorithm for lazy rule learning.

OPTIONS
beam -- Sets the size of beam if using beam search.

debug -- If set to true, classifier may output additional info to the console.

evaluationMeasure -- The measure used to evaluate the performance of rules used in the lazy rule algorithm.

greaterOrEqual -- If to user ">=" or ">".

mode -- The mode of the algorithm used for lazy rule learning.

subSetType -- Sets the type of subsets to use for the algorithm.

writeRules -- If to write learned rules to files.

CAPABILITIES
Class -- Nominal class, Binary class

Attributes -- Binary attributes, Unary attributes, Missing values, Nominal attributes, Empty nominal attributes, Numeric attributes

Additional
min # of instances: 1

Figure 14: A search heuristic, the mode of LazyRule, the subset of instances to consider for training and the the size of the beam, if the mode is beam search, can be chosen by the user in the Weka GUI. Learned rules can be written to files.

The user can select one of the heuristics listed in table 5 and select a mode of the algorithm, i.e. whether it should perform the standard hill climbing search (the default mode), perform a beam search or search in the whole set of rules that can be generated from the query instance. Though the latter is not recommended for larger datasets. If the user selects *Beam* as mode, the size of the beam can be chosen. Furthermore, the amount of instances that actually should be used for training when classifying an instance can be determined. Possible choices here are either the k nearest instances, a percentage of the nearest instances, k random instances (a stratified subset of size k, see section 7.2.1) or a percentage of random instances of the whole dataset.

6.3 Implementation of LazyRule

Most of the data structures needed for the implementation are provided by the SeCo-framework. Only the special treatment of numerical attributes by the algorithm required an additional data structure. The available type NumericCondition alone was not sufficient. As described in section 5.2 a numeric value assigned to an attribute in the test instance results in two numeric conditions. In order to deal with this special kind of associated conditions the type NumericCondition was extended to the type IntervalCondition which has an additional field for the related condition. Different variants of the algorithm are all located in the package variants.

For the beam search approach new types Branch and BestBranches were created. They handle the memory for the b best branches. The algorithm and additional data structures are located in a separate package lazyrule within the framework. The Weka interface is located in weka.classifiers.lazyrule. Two measures were added, the Entropy measure and the Gini index. A class for leave-one-out cross-validation and executable configurations were added. The selection of k nearest neighbors is delegated to the class weka.core.neighboursearch.LinearNNSearch. For that the Weka interface is available in the Weka GUI, the interface is located in weka.classifiers.lazyrule.

7 Evaluation

For the evaluation of the lazy rule learning algorithm the evaluation functionality of the Weka suite (Hall et al., 2009) was used. To compare different variants of the algorithm to each other and to compare it with other learning algorithms, each classifier's accuracy is estimated. The accuracy of a classifier is the probability, to correctly classify a randomly selected instance (Kohavi, 1995).

In this section, first the used datasets will be introduced and then the applied methods for evaluation will be explained. These methods constitute how to determine the accuracy of a classifier and how to compare it to others. Then configurations that possibly result in an improvement will be evaluated. Finally, after performing the evaluation, the obtained results will be reviewed.

7.1 Data Sets

Depending on the experiment, subsets of the datasets in table 1 were used. These datasets include a variety of different data. They differ in the number of instances, attributes and classes as well as in their occurrence of nominal and numeric attributes. It was tried to consider as many datasets as possible in each experiment. However, large datasets can require a remarkable amount of recources for the lazy rule learning algorithm and for this reason a trade-off had to be found in some cases. Which datasets were considered in the particular experiment will be marked by the letters *A, B, C, D*.

7.2 Cross-Validation

In the validation process it is important that the test data is not used in any way to create the classifier (Witten and Frank, 2005). A common statistical technique in machine learning is *cross-validation*. In cross-validation, one splits the data into a fixed number of partitions, the *folds*. All folds except one are used to train the classifier. After training, the left out fold is used for testing. This procedure is repeated for each fold to guarantee that each fold has been used once for testing. The results obtained from testing are then averaged by the number of folds. Thus the classifier's accuracy is the average percentage of correctly classified instances.

In this thesis two particular techniques for accuracy estimation, namely leave-one-out cross-validation and stratified cross-validation, were used.

7.2.1 Stratified Cross-Validation

When dividing the data into folds by employing cross-validation it might happen that the proportions of each classes vary in training and testing data. By coincidence it could have happened that all examples with a certain class were assigned to the testing data but none to the training data. A classifier learned on this training data will most likely not perform well on the testing data. For this reason the folds are *stratified* - the random sampling is done in a way that guarantees that the class distributions in training and test sets are approximately the same as in the whole dataset (Kohavi, 1995).

Employing stratified tenfold cross-validation is a usual way of predicting the accuracy of a learning algorithm and is therefore used in this thesis as well. Since each of the ten folds has to be in turn as testing set, the learning procedure is repeated 10 times. The partitioning into exactly 10 parts is based on extensive tests on numerous different datasets using different learning techniques (Witten and Frank, 2005). For obtaining a reliable accuracy estimate it is necessary to repeat the tenfold cross-validation because different experiments with the same classifier might produce different results due to the effect of random partitioning. Typically the tenfold cross-validation is repeated ten times and averaged afterwards. In total the learning algorithm will be called 100 times on datasets that are nine-tenths the size of the whole dataset.

7.2.2 Leave-One-Out Cross-Validation

One kind of cross-validation is leave-one-out cross-validation which is n-fold cross-validation, where n is the number of instances in the dataset. Hence the dataset is divided into n folds, each of size 1. Each instance in sequence is omitted while the classifier is trained on all the remaining instances. Then the omitted instance is classified by the learned classifier. The percentage of correctly classified instances is the final accuracy estimate.

An advantages of this method is, that it uses the greatest possible amount of data for training in each case. This presumably increases the chance that the classifier is accurate (Witten and Frank, 2005). Another important point is, that this method is deterministic because no random sampling takes place. In contrast to for example ten-fold cross-validation, there is no need to repeat it ten times. The result would always be the same. Since leave-one-out cross validation needs to run just once to obtain an accuracy estimate it was used when looking for improvements to rapidly judge some variant of the algorithm in order to decide about possible further review. Leave-one-out cross-validation is

Table 1: Datasets used for evaluation. Source: `http://archive.ics.uci.edu/ml/index.html`. The last columns show which datasets were used in which experiments. The letters *A, B, C, D* refer to portions of the datasets that are used in different experiments.

Name	Filename	Instances	Attributes	Classes	*A*	*B*	*C*	*D*
anneal	anneal.arff	798	38	6	x	x	x	x
anneal.ORIG	anneal.ORIG.arff	798	38	6	x	x	x	x
audiology	audiology.arff	226	70	24	x	x	x	x
autos	autos.arff	205	26	7	x	x	x	x
balance-scale	balance-scale.arff	625	5	3	x	x	x	x
breast-cancer	breast-cancer.arff	286	10	2	x	x	x	x
wisconsin-breast-cancer	breast-w.arff	699	10	2	x	x	x	x
horse-colic	colic.arff	368	23	2	x	x	x	x
horse-colic.ORIG	colic.ORIG.arff	368	28	2	x	x	x	x
credit-rating	credit-a.arff	690	16	2	x	x	x	x
german_credit	credit-g.arff	1000	21	2	x	x	x	x
pima_diabetes	diabetes.arff	768	9	2	x	x	x	x
Glass	glass.arff	214	10	7	x	x	x	x
cleveland-14-heart-disease	heart-c.arff	303	14	5	x	x	x	x
hungarian-14-heart-disease	heart-h.arff	294	14	5	x	x	x	x
heart-statlog	heart-statlog.arff	270	14	2	x	x	x	x
hepatitis	hepatitis.arff	155	20	2	x	x	x	x
hypothyroid	hypothyroid.arff	3772	30	4	x	x	x	x
ionosphere	ionosphere.arff	351	35	2	x	x	x	x
iris	iris.arff	150	5	3	x	x	x	x
kr-vs-kp	kr-vs-kp.arff	3196	37	3		x	x	x
labor	labor.arff	57	17	2	x	x	x	x
letter	letter.arff	20000	17	26				x
lymphography	lymph.arff	148	19	4	x	x	x	x
mushroom	mushroom.arff	8124	23	2				x
primary-tumor	primary-tumor.arff	339	18	22	x	x	x	x
segment	segment.arff	2310	20	7	x	x	x	x
sick	sick.arff	3772	30	2	x	x	x	x
sonar	sonar.arff	208	61	2	x	x	x	x
soybean	soybean.arff	683	36	19	x	x	x	x
splice	splice.arff	3190	62	3			x	x
vehicle	vehicle.arff	846	19	4	x	x	x	x
vote	vote.arff	435	17	2	x	x	x	x
vowel	vowel.arff	990	14	11	x	x	x	x
waveform	waveform-5000.arff	5000	41	3				x
zoo.arff	autos	101	18	7	x	x	x	x

feasible just for smaller datasets and usually leads to high computational costs, because the entire learning procedure must be executed n times (Witten and Frank, 2005). In case of the lazy rule learning algorithm, however, due to its lazy approach, leave-one-out cross-validation performs faster than ten times performing a tenfold cross-validation. In some cases leave-one-out cross-validation was used for evaluating classifiers. But unless otherwise stated, stratified ten-fold cross-validation, repeated ten times with different partitionings, was used.

7.3 Comparing Classifiers

For judging a classifier's performance compared to other classifiers, a corrected sampled paired t-test provided by the Weka suite is used. It is based on the observation that the variance of the cross-validation estimator is often underestimated which leads to the wrong conclusion that the new algorithm is significantly better although it is actually not (Nadeau and Bengio, 2003). Consequently the corrected t-test does not generate as many significant differences as the standard t-test. In particular, the tests proposed by Nadeau and Bengio (2003) do not reject the null hypothesis too often when the hypothesis is true, but they frequently reject the null hypothesis when the hypothesis is false.

7.3.1 Paired Student's t-Test

The null hypothesis is that the second of the two classifiers is not significantly better than the first and the confidence interval is $p = 0.05$. This means that the null hypothesis is rejected only when confidence that one classifier performs significantly better is more than 95%.

7.4 Different Heuristics

Before the algorithm was reviewed with possible improvements, the algorithm was evaluated with several heuristics. It showed that there are significant differences in accuracy depending on the particular heuristic.

In the SECO-framework heuristics are used to maximize the accuracy of a *set* of rules learned from a training set. But that does not necessarily imply that these heuristics would work in the same way for learning a *single* rule from the training set for classifying a *single* instance. In separate-and-conquer rule learning the heuristics try to maximize the positive coverage for a rule on a dataset. However, in the proposed lazy rule learning algorithm the main requirement for a learned rule is that it classifies the test instance correctly. If the whole training data is used for learning a single rule from the conditions derived from the test instance, the algorithm will tend to maximize the coverage of the whole training data by finding some rule that can be constructed from the available conditions. The algorithm thus might strive to construct rules that cover the majority class if procurable.

It was assumed, that there is a small set of heuristics that work best with the basic algorithm. To find out, which they are and which characteristics they have that make them superior to others, the algorithm was evaluated with almost all heuristics available in the SECO-framework. Because the basic variant of the algorithm needs much computation time, the biggest datasets were omitted in evaluation. Table 2 shows a ranking of the heuristics with respect to significantly better or significantly worse performance. A classifier wins against another one if it performs significantly better and loses if it performs significantly worse. The ranking is determined by subtracting the losses from the wins. In table 6 the complete results of the evaluation are shown. A description of the heuristics can be found in table 5.

Table 2: Ranking of LAZYRULE with different heuristics. The columns show how the algorithm performed depending on the applied heuristic against all other heuristics. Wins are the number of cases when the particular algorithm was significantly better than another. Classifiers are ranked according to their win-loss difference. Losses when it performed significantly worse. The ranking is based on the tested datasets in table 6. Datasets *B*.

Heuristic	Wins−Losses	Wins	Losses
Laplace	197	208	11
Linear Regression	114	153	39
F-Measure	32	120	88
Linear Cost	28	129	101
m-Estimate	25	113	88
Kloesgen-Measure	21	119	98
Correlation	-16	94	110
Foil Gain	-18	97	115
Relative Linear Cost	-31	109	140
Weighted Accuracy	-57	80	137
Precision	-126	71	197
Accuracy	-169	50	219

It shows that the basic algorithm performs significantly better on many datasets when employing the Laplace estimate. It often outperformed the other configurations by far. The configuration using the Laplace estimate performed significantly worse than other configurations on just four different datasets (11 losses in table 2). Table 3 gives more information about the learned rules using different heuristics obtained by evaluating rules learned on several datasets.

Table 3: Properties of rules that were found by LazyRule with different heuristics applied. The evaluation was performed on various datasets with 16347 instances altogether. Ordered by the percentage of correct classifications. *Correct* is the number of correct classified instances, *Conditions* is the number of conditions in all rules, *Empty* is the number of rules that have an empty rule body, *Length* is the average rule length. Evaluated on datasets *A*.

Heuristic	Correct	Conditions	Empty	Length	Empty (%)	Correct (%)
Laplace	12665	45770	2	2.80	0.01	77.48
Linear Regression	12039	43408	1	2.66	0.01	73.65
F-Measure	12036	32243	2387	1.97	14.60	73.63
m-Estimate	11735	45641	6	2.79	0.04	71.79
Linear Cost	11536	23487	6646	1.44	40.66	70.57
Foil Gain	11485	24882	1	1.52	0.01	70.26
Kloesgen Measure	11381	46314	1	2.83	0.01	69.62
Weight. Accuracy	10957	43606	1	2.67	0.01	67.03
Precision	10898	50966	1	3.12	0.01	66.67
Correlation	10260	48365	1	2.96	0.01	62.76
Rel. Linear Cost	9966	55631	9	3.40	0.06	60.97
Accuracy	5336	67436	570	4.13	3.49	32.64

For the Laplace heurisitc rules had an average length of 2.8 conditions per rule. It produced almost no rules that are empty, i.e. that have no conditions in its rule body. Like all other heuristics the Laplace heuristic rates those rules higher that cover more positive examples and less negative examples. Indeed it also guarantees that a rule which covers *more* (negatives and positives) examples scores *higher* than a rule with the same ratio of negative and positive examples, but in total less coverage. However, more than other heuristics, the Laplace heuristic tends to overfit the training data and thus also places stronger emphasis on consistency over coverage (Janssen and Fürnkranz, 2010b). This fact may explain the predominance of the Laplace estimate: In contrast to eager rule learning, coverage is less important for LazyRule - the contructed rule shall primarily classify a single instance, the test instance and not a set of training instances.It is concluded that the Laplace heuristic is the most suitable heuristic for the algorithm if the whole set of training data is used for each classification. Thus the Laplace heuristic was used in the further evaluation process.

7.5 Possible Refinements

It is assumed that there are several possibilities to improve accuracy and efficiency of the basic algorithm. In section 5.7 different approaches that possibly improve the algorithm are described. In this section different configurations of the algorithm are reviewed. Before comparing the algorithm to other learning algorithms, refined variants of the basic algorithm that attempted to improve its performance will be evaluated.

7.5.1 Different Beam Sizes

In order to improve the algorithm's accuracy a beam search approach was employed (see section 5.7.1). Since setting the beam size b results in a b-times longer runtime of the algorithm, the beam approach was just conducted for values ≤ 4. Table 7 shows the results. For $b = 2$ the accuracy is significantly better in five datasets. A beam size of $b = 3$ does not result in any significant improvement and a beam size of $b = 4$ results in just one more dataset with significantly better accuracy. The beam search with $b = 2$ seems to be a good compromise between runtime and improvement. It was therefore decided, to consider the beam search with a beam size of $b = 2$ for an improved variant of the algorithm.

Since there is no significant degradation in accuracy but rather a significant improvement in a few cases, it can be concluded, that by searching for rules with the highest Laplace estimate frequently finds a correct rule.

7.5.2 Beam Search and Stratified Subsets

Employing beam search results in increased execution time. As for some datasets using a beam search improved accuracy, the higher computation costs caused by it may be compensated. This was tried by using a stratified subset of the provided training data in each classification. In table 8 the amount of instances that composes the stratified subset is a fixed number. But the sizes of the datasets differ highly so that it is a better choice to construct the subset by a percentage of the available training data rather than by a fixed number. A beam search with $b = 2$ approximately doubles the execution time. On the other hand, using less data would decrease the time used for rule evaluation (cf., section 5.6). In order to keep the computation time in a similar magnitude, using a fraction of the training data shall not negatively affect the accuracy for any dataset. The results are shown in table 9. However, this configuration did not result in an improvement. For example, using 60% of the training data with a beam of size $b = 2$ instead of improving the accuracy for some datasets it significantly reduced it for two datasets. It was therefore abandoned.

7.5.3 Nearest Neighbor Subsets

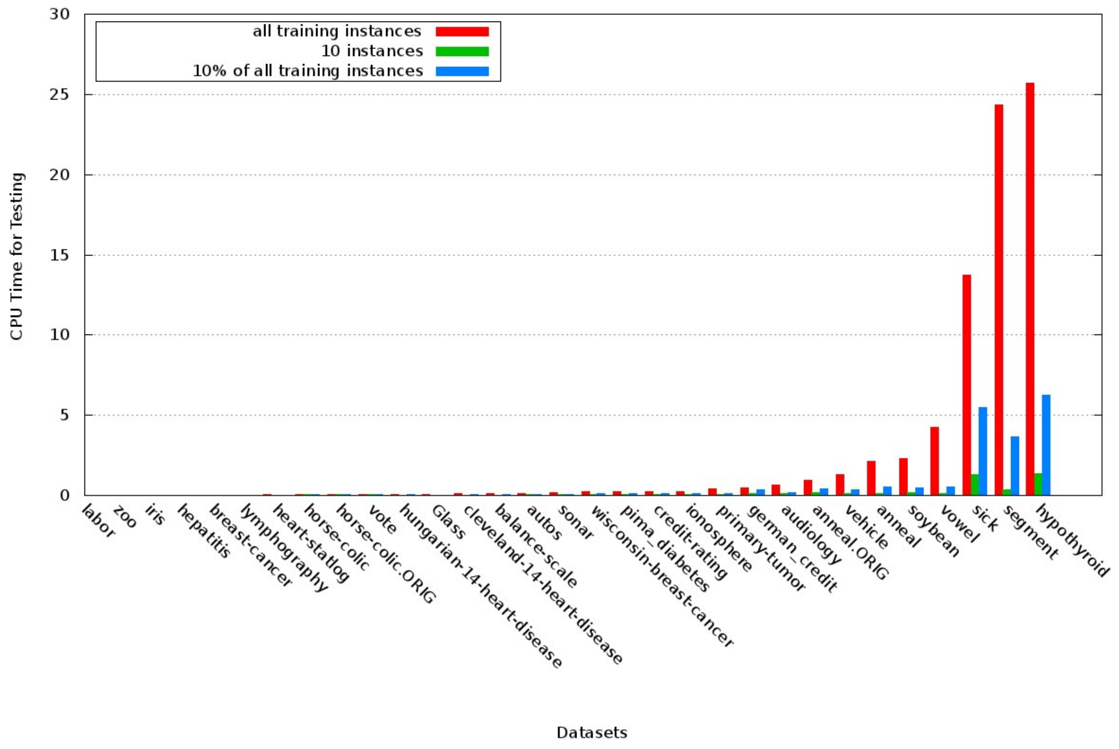


Figure 15: CPU time for testing LAZYRULE and LAZYRULENN

Since using stratified subsets of the whole training data did not result in a real improvement, another idea was realized. To achieve the goal of a reduced execution time by using less training data for learning rules on, one can already take advantage of lazy learning in this step. By ignoring instances that would not contribute to a good rule anyway the probability to learn a rule that classifies the test instance correctly is much higher. In this section the nearest k-neighbor algorithm (cf., section 4.1) and the basic lazy rule learning algorithm (cf., section 4) were combined to one improved lazy rule learning algorithm, called LAZYRULENN in the following. First, it was analyzed how different values for k affect accuracy and if accuracy would be better if a subset is used that include a percentage of the nearest neighbors rather than a fixed value for k (see table 10). According to the obtained results it was concluded that small values for k result in the highest accuracy. It was also tested if now, with the new precondition, different heuristics would perform better and if a beam search would further improve the accuracy. However, the LAZYRULENN using the Laplace estimate still outperformed the other configurations and the beam approach did not result in an improvement anymore. In figure 15 the execution times for LAZYRULE, the lazy rule learner in its basic version (using the whole set of training instances), LAZYRULENN using ten instances for training and LAZYRULENN using the nearest 10% of the instances are visualized. Due to enormous evaluation the largest datasets were omitted. But the tendency is obvious: While for LAZYRULEthe execution time climbs to an unacceptable level it stays manageable for LAZYRULENN.

Table 4: Longer rules for $\geq$. Evaluated on datasets *A*.

	LAZYRULE	LAZYRULENN, $>$, $k = 5$	LAZYRULENN, $\geq$, $k = 5$
Accuracy (%)	75.41	82.86	82.86
Average Rule Length	2.88	0.89	19.56
Empty Rules (%)	0.01	54.41	1.78

The configuration using k-nearest neighbors for learning rules will be used for the comparison to other learning algorithms in the next section. However, when learning rules on k-nearest neighbors, the algorithm produces many empty rules, depending on the dataset (see table 16). If one wants the algorithm to not produce empty rules, it could be modified in this way: Instead of starting with an empty rule, start with the rule that contains the best condition. Another possibility to obtain less empty rules and generally longer rules is to accept rules as improvement that score the same heuristic value as their predecessor rule but contain one condition more. To achieve this, the test in line 9 of REFINERULE is replaced by $NewRule \geq BestRule$. The results in table 4 show that the accuracy does not change significantly but the rules are considerably longer and more rarely empty.

7.6 LAZYRULE compared to other Learning Algorithms

In this section the lazy rule learning algorithm LAZYRULENN is compared to other learning algorithms. These include a decision tree algorithm, a separate-and-conquer rule learning algorithm and different configurations for a weighted and unweighted k-nearest neighbor algorithm.

The decision tree algorithm is the Weka implementation of the C4.5 algorithm (cf., section 3), called J48. It is used in its standard configuration[8]. JRip is Weka's implementation of RIPPER (Cohen, 1995) standing for Repeated Incremental Pruning to Produce Error Reduction) and represents a the separate-and-conquer rule learning algorithm (cf., section 2.3). It is also used in its standard configuration[9]. For the k-nearest neighbor algorithm (cf., section 4.1) the Weka implementation was used with the values 1, 2, 3, 5, 10, 15, 25 for k, for each k unweighted and weighted according to equation 4 (cf., section 4.1). The detailed results can be found in the appendix (cf., tables 12, 13, 14).

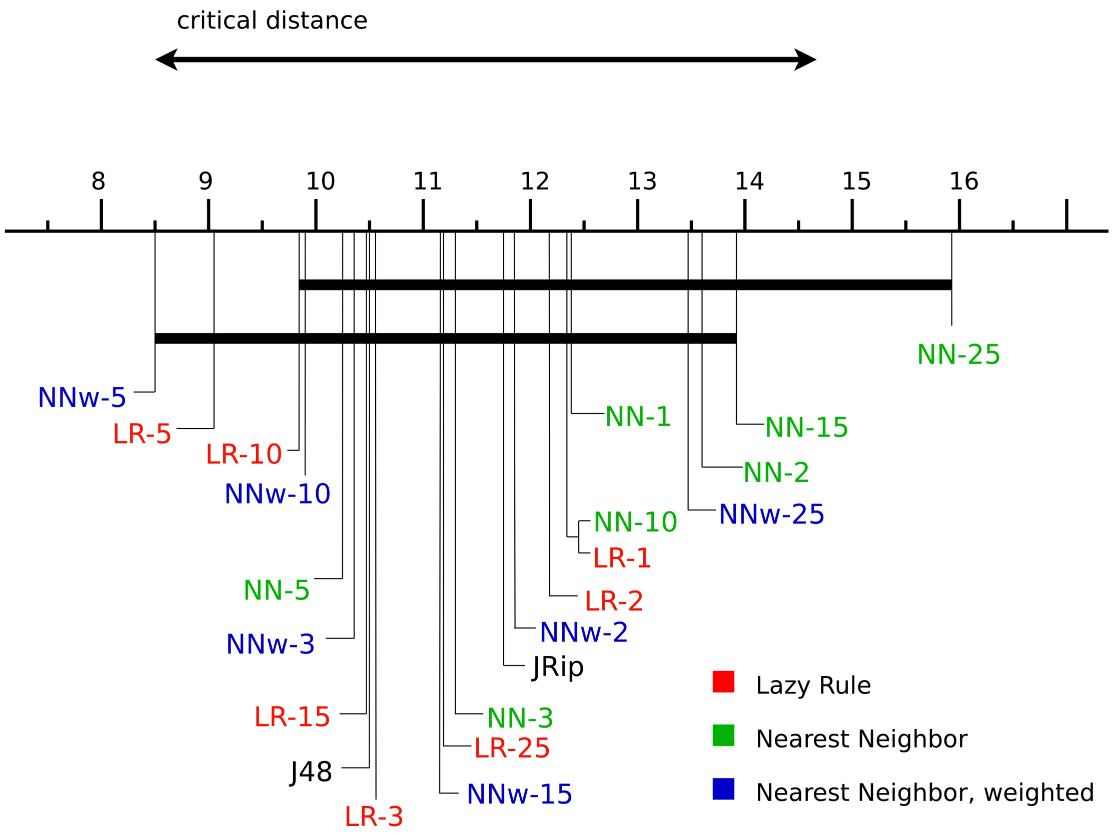


Figure 16: Results of significance test. The number for the lazy algorithms correspond to the number of used instances. Algorithms are grouped if their accuracy does not differ significantly for $\alpha = 0.01$. Only algorithm NNw-5 and LR-5 are significantly better than NN-25. Evaluated on datasets *D*.

[8] J48's standard configuration is: no binary splits, collapse tree, confidence factor used for pruning 0.25, minimum 2 instances per leaf, no reduced-error pruning, subtree raising for pruning, pruned and MDL correction (cf., Quinlan (1993)).

[9] JRip's standard configuration is: check for error rate $\leq \frac{1}{2}$ is included in stopping criterion, 3 folds, minimum weight = 2, 2 optimization runs, seed = 1, pruning.

For comparing all the algorithms and their configurations described above, another evaluation method was employed: Significant differences in accuracy between the algorithms are determined by a Friedmann Test with a post-hoc Nemenyi Test (Demšar, 2006). First, the fractional ranks[10] are calculated for each dataset. The average ranks are then used to calculate the statistic by Iman and Davenport (cf., Demšar (2006)) that is used by the Friedmann Test to determine if the classifiers differ significantly in accuracy. The null hypothesis of the Friedmann Test is that the classifiers are not significantly different. When the null hypothesis is rejected, the classifiers are significantly different and the post-hoc Nemenyi Test is performed. For the post-hoc Nemenyi Test the critical difference is computed. Two classifiers are considered to have a significantly different accuracy if their average ranks' difference is at least the critical distance.

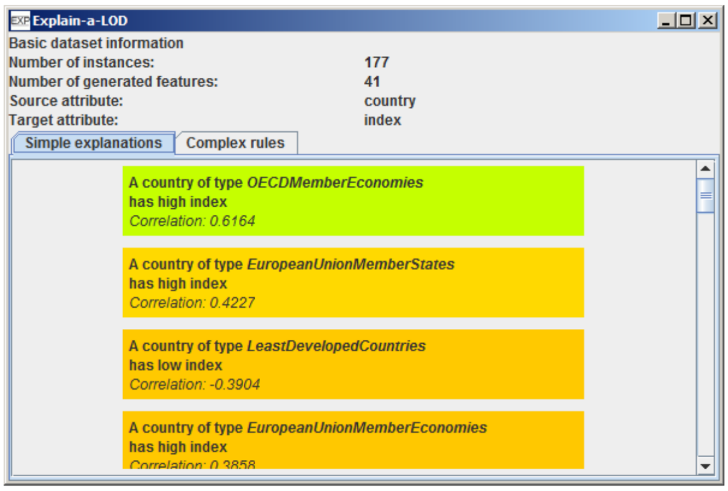
7.6.1 Conclusion

In figure 16 the result of the significance test over all algorithms is shown. For the significance level 0.01 there are only significant differences between LazyRuleNN with $k = 5$ and the unweighted k-nearest neighbor algorithm with $k = 25$ and between the weighted k-nearest neighbor algorithm with $k = 5$ and the unweighted k-nearest neighbor algorithm with $k = 25$. So algorithm LazyRuleNN is neither significantly better nor significantly worse than the considered algorithms. A particular type of dataset - characterized by number of attributes, classes, instances - where the algorithm is worse or better than others could not be determined.

[10] Items that compare equal receive the same ranking number, which is the mean of what they would have under ordinal rankings.

8 An Application of LazyRule

Explain-a-LOD (Paulheim, to appear) is a tool for using Linked Open Data[11] (LOD) for interpreting statistics. It is often difficult to interpret a given statistic, i.e. to explain it. For example, given a statistics file containing two attributes, a country's name and its corruption index, it is interesting to know why the corruption is higher in some countries than in others. When a statistics file is loaded into Explain-a-LOD, the tool automatically retrieves data from the Linked Open Data cloud and generates possible explanations: The data set is enriched by utilizing the additional data[12] so that it contains more attributes. Then, a correlation analysis is performed and rule learning is used for discovering more complex patterns. The results are presented to the user (figure 17).



Basic dataset information

Number of instances:	177
Number of generated features:	41
Source attribute:	country
Target attribute:	index

A country of type *OECDMemberEconomies*
has high index
Correlation: 0.6164

A country of type *EuropeanUnionMemberStates*
has high index
Correlation: 0.4227

A country of type *LeastDevelopedCountries*
has low index
Correlation: -0.3904

A country of type *EuropeanUnionMemberEconomies*
has high index
Correlation: 0.3858

Figure 17: Explain-a-LOD (Paulheim, to appear), showing possible explanations for a statistics file

For the tool's functionality that uses rule learning, the application of LazyRuleNN might be useful. It is planned to utilize the rules generated by the lazy rule learning algorithm. For the purpose of the tool it is advantageous that the algorithm rapidly learns many context-less rules - in contrast to eager rule learners that indeed learn the first rule of the rule set independently, however the following rules all relate to their preceding rules. The set of rules obtained from the algorithm needs further processing: From that set of rules that were each learned for a single instance, similar rules, i.e., rules that have the same attributes, should be considered together to compare their accuracies on the whole dataset. The rule with highest accuracy might be a useful rule to contribute to the explanation of the statistics file.

[11] Linked data describes a method of publishing structured data so that it can be interlinked and become more useful. Linked Open Data (LOD) is a large collection of semantically annotated datasets.

[12] This is accomplished by the feature generation toolkit FeGeLOD (Paulheim and Fürnkranz, 2011).

9 Summary and Conclusion

Up to now, many rule learning algorithm have been proposed. However, all of them have one crucial property in common: They are all eager learning algorithms. For this reason, in this thesis the concept of lazy learning was applied to rule learning. The result is a lazy rule learning algorithm, referred to as LazyRule, that learns a single rule for a query instance. Basically, it derives conditions from the query instance and constructs a rule from them. Since the algorithm in its basic version performed worse than many other learning algorithm, an enhancement is presented, too. By employing a k-nearest neighbor search before searching for rules, the basic algorithm's accuracy could be improved significantly. Additionally, by downsizing the training data this way, the computation time strongly decreased.

The improved algorithm LazyRuleNN, employing the Laplace estimate, was then compared to other, widely-used learning algorithms, including a rule learning algorithm as well as k-nearest neighbor algorithms. Evaluating these algorithm on various datasets showed that the lazy rule learning algorithm neither performed significantly worse nor significantly better.

This algorithm distinguishes from other learning algorithm by the fact that it learns many, context-less rules. Depending on the configuration, these rules may vary in their lengths. Although each rule is learned to classify only one instance, after further processing the resulting rules might be useful for other applications.

Within this thesis, several possible improvements were reviewed. Yet the algorithm is a very basic one and offers different opportunities to improve it further. One promising possible improvement that was just briefly described within this thesis, is preselecting attributes, especially decreasing the learning time for datasets with many attributes.

Table 5: Search heuristics for rule evaluation used by the algorithm. The measure p is for the number of true positives i.e. number of positives covered by a rule, the measure n for the number of negatives covered by the rule. P is the total number of positive examples and N the total number of negative examples. The measures p' and n' correspond to the preceding rule (before refinement). Formulas have been taken from (Fürnkranz, 1999) and (Janssen and Fürnkranz, 2010a).

Heuristic	Formula	Measures	Weka parameter
Laplace Estimate	$\frac{p+1}{p+n+2}$	p, n	*default*
Precision	$\frac{p}{p+n}$	p, n	prec
Accuracy	$\frac{p+(N-n)}{P+N} \simeq p-n$	P, N, p, n	acc
Weighted Accuracy	$\frac{p}{P} - \frac{n}{N}$	P, N, p, n	wra
Correlation	$\frac{p\,N-n\,P}{\sqrt{P\,N(p+n)\,(P-p+N-n)}}$	P, N, p, n	corr
F-Measure	$\frac{(1+\beta^2)\cdot(\text{Precision}\cdot\frac{p}{P})}{(\beta^2\cdot\text{Precision}+\frac{p}{P})}$	P, N, p, n	fm
Kloesgen-Measure	$(\frac{p+n}{P+N})^{\omega}\cdot(\frac{p}{p+n})$	P, N, p, n	km
m-Estimate	$\frac{p+m\cdot\frac{p}{P+N}}{p+n+m}$	P, N, p, n	mest
Generalized m-Estimate	$\frac{p+m\cdot c}{p+n+m}$	P, N, p, n	mest
Linear Cost	$c\cdot p-(1-c)\cdot n$	P, N, p, n	lc
Relative Linear Cost	$c_r\cdot\frac{p}{P}-(1-c_r)\cdot\frac{n}{N}$	P, N, p, n	rlc
Foil Gain	$p\cdot(\log_2(\frac{p}{p+n}))-\log_2(\frac{p'}{p'+n'})$	p, n, p′, n′	foil
Entropy	$-\frac{p}{p+n}\cdot\log_2(\frac{p}{p+n})-\frac{n}{p+n}\cdot\log_2(\frac{n}{p+n})$	p, n,	entr
Gini Index	$1-(\frac{p}{p+n})^2-(\frac{n}{p+n})^2$	p, n	gini

Table 6: Various heuristics applied to algorithm LazyRule. The columns show the results for applying the measures Laplace Estimate, Precision, Accuracy, Weighted Accuracy, Correlation, F-Measure, Kloesgen Measure, m-Estimate, Linear Cost, Relative Linear Cost, Linear Regression, Foil Gain. Evaluated on datasets *B*. Significant differences refer to Laplace estimate (first column).

Dataset	Lapl.	Prec.	Acc.	W.acc.	Corr.	F-ms.	Kloesg.	m-est.	Lin.C.	R.lin.c.	L.reg.	Foil g.
kr-vs-kp	95.68	71.23 •	86.80 •	85.03 •	90.43 •	90.43 •	90.43 •	91.05 •	90.43 •	89.67 •	90.45 •	88.55 •
iris	94.27	81.73 •	84.07 •	70.33 •	78.53 •	88.73 •	90.27 •	91.13	88.47 •	83.73 •	90.93	78.40 •
vote	94.02	87.91 •	95.95	95.61	95.82	95.86	84.12 •	88.58 •	85.43 •	95.84	93.82	95.38
sick	93.88	93.98	93.41	62.84 •	62.18 •	93.88	61.13 •	93.88	93.88	76.10 •	93.88	63.09 •
hypothyroid	92.86	92.57	0.05 •	81.24 •	73.35 •	91.49 •	91.88	92.29 •	92.29 •	61.35 •	92.29 •	94.65 ∘
ionosphere	91.74	35.90 •	88.89	88.86	89.66	90.92	73.56 •	83.76 •	69.43 •	80.65 •	92.08	80.28 •
wisconsin-breast-cancer	91.47	86.98 •	81.06 •	78.90 •	80.58 •	77.47 •	94.09 ∘	81.10 •	65.91 •	83.69 •	81.92 •	88.35 •
anneal	90.83	70.79 •	1.87 •	75.69 •	33.03 •	91.69	36.03 •	76.17 •	76.17 •	14.50 •	81.96 •	68.34 •
anneal.ORIG	89.33	85.26 •	2.15 •	75.90 •	43.22 •	87.36 •	67.04 •	76.17 •	76.17 •	36.89 •	82.79 •	64.28 •
balance-scale	85.44	84.58	7.84 •	64.94 •	66.14 •	65.63 •	76.01 •	79.74 •	70.83 •	76.52 •	76.39 •	65.23 •
credit-rating	85.04	47.55 •	85.51	85.51	85.51	85.51	85.51	85.64	85.51	85.51	85.51	85.51
labor	83.70	52.07 •	38.97 •	59.83 •	60.93 •	80.57	61.13 •	64.67 •	68.00 •	75.93	82.43	60.27 •
horse-colic	81.87	66.88 •	80.97	80.97	81.03	81.52	80.19	82.01	85.40	81.46	85.84 ∘	80.97
soybean	80.49	24.69 •	36.00 •	24.63 •	62.99 •	71.86 •	40.44 •	42.23 •	59.21 •	40.19 •	52.36 •	32.93 •
hungarian-14-heart-disease	80.38	66.82 •	0.00 •	79.10	79.19	79.33	77.94	77.86	64.78 •	78.96	78.80	78.52
hepatitis	79.89	35.09 •	77.67	74.33	72.33	79.38	44.37 •	79.38	79.38	70.62 •	79.38	52.18 •
zoo	77.50	80.18	85.72 ∘	54.20 •	83.95 ∘	84.35 ∘	75.85	44.01 •	54.67 •	84.93 ∘	69.31 •	73.27
lymphography	76.16	52.72 •	3.00 •	72.83	44.93 •	58.42 •	73.12	63.82 •	65.02 •	11.37 •	72.06	69.39
cleveland-14-heart-disease	75.52	56.40 •	0.00 •	77.47	78.69	78.44	81.22 ∘	78.35	66.19 •	82.24 ∘	80.59	74.19
segment	73.87	40.01 •	71.28 •	53.18 •	70.14 •	58.70 •	71.09 •	70.05 •	71.78 •	67.46 •	69.88 •	60.48 •
breast-cancer	72.15	68.80	64.02 •	70.01	68.12	70.30	62.91 •	70.30	70.30	59.41 •	70.58	67.66
horse-colic.ORIG	71.95	67.42 •	47.87 •	65.21 •	67.58	66.31 •	63.94 •	66.31 •	66.31 •	71.19	67.93 •	64.95 •
german-credit	70.88	70.00	49.41 •	67.64	67.22 •	70.00	65.71 •	70.00	70.00	70.10	70.00	64.78 •
heart-statlog	70.74	57.63 •	74.30	75.44	75.85	72.67	78.89 ∘	70.30	63.67 •	78.59 ∘	74.33	74.41
pima-diabetes	69.99	65.50 •	46.73 •	65.33 •	68.89	65.11 •	68.85	65.11 •	65.11 •	70.99	65.67 •	65.45 •
autos	66.76	20.46 •	0.00 •	49.89 •	22.74 •	41.94 •	60.96	48.75 •	60.95	2.85 •	56.35 •	54.09 •
sonar	65.50	46.62 •	49.00 •	65.34	64.48	53.24 •	63.46	56.42 •	65.88	65.39	64.24	64.30
Glass	63.48	37.65 •	0.00 •	48.35 •	35.65 •	47.75 •	56.17 •	58.60	63.66	28.54 •	60.35	47.72 •
vehicle	52.46	26.66 •	42.56 •	42.99 •	48.80	48.70	53.16	54.26	54.16	51.37	51.52	44.32 •
audiology	48.56	38.67 •	8.86 •	36.19 •	39.72 •	38.02 •	37.55 •	31.71 •	54.18	15.47 •	36.42 •	42.82
primary-tumor	41.47	37.23 •	2.30 •	30.94 •	19.27 •	27.23 •	37.00 •	38.65	41.03	3.22 •	38.18	37.17 •
vowel	29.45	10.24 •	29.24	23.22 •	29.43	27.16 •	29.52	29.83	29.34	29.63	29.78	29.83
Average	76.17	58.13	44.86	65.06	63.76	70.62	66.67	68.81	69.17	60.14	72.44	65.99

∘ statistically significant improvement, • statistically significant degradation , corrected paired Student's t-Test, $\alpha = 0.05$

Table 7: Different values for b when employing beam search to the basic algorithm. The comlumns show the results for keeping the 1, 2, 3 and 4 most promising branches in memory. The results are compared to the basic algorithm (first column). Evaluated on datasets *A*.

		Beamsize		
Dataset $b =$	1	2	3	4
iris	94.27	94.33	94.33	94.33
vote	94.05	94.30	94.32	94.37
sick	93.88	93.88	93.87	93.87
hypothyroid	92.86	92.84	92.84	92.84
ionosphere	91.74	91.74	91.80	91.80
wisconsin-breast-cancer	91.47	91.86	91.76	91.76
anneal	90.83	92.56 $*$	93.05 $*$	93.05 $*$
anneal.ORIG	89.33	89.59	89.76	89.81
balance-scale	85.44	84.71	84.82	84.82
credit-rating	85.03	85.39	85.64	85.83
labor	83.53	82.80	82.63	82.83
horse-colic	81.87	83.75	83.74	83.80
hungarian-14-heart-disease	80.44	79.44	79.65	80.06
soybean	80.44	82.18	81.99	81.87
hepatitis	79.89	79.82	79.82	79.88
zoo	77.50	78.18	78.09	78.09
lymphography	76.29	78.19	79.13	79.81
cleveland-14-heart-disease	75.42	79.58	80.27	80.47 $*$
segment	73.85	74.34 $*$	74.63 $*$	74.81 $*$
breast-cancer	72.15	71.87	71.72	71.62
horse-colic.ORIG	71.90	72.57	72.55	72.58
german-credit	70.84	70.73	70.88	70.92
heart-statlog	70.67	75.52 $*$	76.41 $*$	76.74 $*$
pima-diabetes	69.92	70.42	70.39	70.39
autos	66.85	67.84	66.95	67.15
sonar	65.50	65.50	65.50	65.50
Glass	63.39	64.23	64.55	64.56
vehicle	52.41	55.92 $*$	57.96 $*$	59.13 $*$
audiology	48.46	59.70 $*$	60.19 $*$	58.72 $*$
primary-tumor	41.30	42.45	42.54	42.54
vowel	29.43	29.40	29.39	29.42
Average	75.51	76.63	76.81	76.88

$*$ statistically significant improvement, $\alpha = 0.05$

Table 8: The algorithm with a stratified random subset of instances used for training. The stratified subsets contain 100, 200, 500 and 1000 instances. The results are compared to the basic algorithm (first column). Evaluated on datasets *C*.

	Number of instances								
	all	100	100	200	200	500	500	1000	1000
Dataset	$b = 1$	$b = 1$	$b = 2$	$b = 1$	$b = 2$	$b = 1$	$b = 2$	$b = 1$	$b = 2$
kr-vs-kp	95.69	80.82 ●	87.31 ●	85.30 ●	90.92 ●	90.71 ●	94.17 ●	93.69 ●	95.70
iris	94.27	94.27	94.33	94.27	94.33	94.27	94.33	94.27	94.33
vote	94.05	93.08	93.68	94.05	94.23	94.05	94.23	94.02	94.35
sick	93.88	93.88	93.88	93.88	93.88	93.88	93.89	93.88	93.88
hypothyroid	92.86	92.29 ●	92.29 ●	92.29 ●	92.29 ●	92.29 ●	92.30 ●	92.38 ●	92.39 ●
ionosphere	91.74	86.78 ●	86.64 ●	91.74	91.74	91.74	91.74	91.74	91.74
wisconsin-breast-cancer	91.47	88.68	89.16	90.52	90.99	91.47	91.85	91.47	91.85
anneal	90.83	78.89 ●	79.95 ●	83.29 ●	85.44 ●	90.83	92.58 ○	90.83	92.62 ○
anneal.ORIG	89.33	81.18 ●	81.22 ●	84.52 ●	84.45 ●	89.33	89.59	89.33	89.60
balance-scale	85.44	78.05 ●	78.37 ●	83.57	83.33	85.44	84.69 ●	85.44	84.71
credit-rating	85.03	80.09 ●	81.16 ●	82.48	83.39	85.03	85.32	85.04	85.35
labor	83.70	83.53	82.80	83.70	82.80	83.53	82.80	83.70	82.80
horse-colic	81.90	78.15	80.08	81.90	83.72	81.82	83.61	81.90	83.72
soybean	80.48	50.12 ●	53.40 ●	63.28 ●	67.36 ●	80.48	82.16	80.42	82.08
hungarian-14-heart-disease	80.38	78.86	77.10	80.31	79.38	80.34	79.21	80.27	79.38
hepatitis	79.89	79.89	79.82	79.89	79.82	79.89	79.82	79.89	79.82
zoo	77.50	77.50	78.09	77.50	78.18	77.50	78.09	77.50	78.09
lymphography	76.22	76.02	78.40	76.16	78.86	76.36	78.40	76.29	78.28
cleveland-14-heart-disease	75.55	74.16	77.99	75.49	79.62	75.59	79.75	75.52	79.68
segment	73.87	65.49 ●	65.45 ●	67.14 ●	67.14 ●	68.96 ●	69.03 ●	71.30 ●	71.44 ●
splice	72.91	65.58 ●	66.67 ●	68.30 ●	73.65	71.64	83.28 ○	72.52	87.75 ○
breast-cancer	72.18	71.20	71.34	72.15	71.87	72.11	71.87	72.11	71.87
horse-colic.ORIG	72.06	73.10	72.81	72.03	72.49	72.09	72.60	71.90	72.60
german-credit	70.90	70.08	69.97	70.17	70.11	70.81	70.66	70.88	70.65
heart-statlog	70.52	73.52	77.19	70.59	75.44 ○	70.56	75.44 ○	70.56	75.59 ○
pima-diabetes	70.00	70.26	70.20	69.09	69.06	69.92	70.41	69.96	70.39
autos	66.71	66.80	67.84	66.75	67.79	66.72	67.59	66.86	67.40
sonar	65.50	65.50	65.50	65.50	65.50	65.50	65.50	65.50	65.50
Glass	63.58	63.53	64.18	63.58	64.32	63.62	64.37	63.44	64.37
vehicle	52.55	45.20 ●	46.33 ●	49.48	51.48	52.55	55.92 ○	52.34	56.00 ○
audiology	48.42	44.99	51.74	48.47	59.44 ○	48.51	59.89 ○	48.56	59.88 ○
primary-tumor	41.30	39.15	39.06	41.44	42.39	41.24	42.68	41.47	42.39
vowel	29.42	27.30	27.25	28.54	28.56	29.41	29.42	29.43	29.39
Average	76.06	72.36	73.37	74.16	75.57	75.70	77.19	75.89	77.44

○ statistically significant improvement, ● statistically significant degradation , corrected paired Student's t-Test, $\alpha = 0.05$

Table 9: The algorithm with a stratified random subset of instances used for training. The stratified subsets contain either 40%, 45%, 50%, 55% or 60% of the original training data. Each subset is evaluated with a beam size of 1 as well with a beam size of 2. The results are compared to the basic algorithm (first column). Evaluated on datasets *B*.

	Data used for training (%)										
	all	40	40	45	45	50	50	55	55	60	60
Dataset	$b = 1$	$b = 1$	$b = 2$	$b = 1$	$b = 2$	$b = 1$	$b = 2$	$b = 1$	$b = 2$	$b = 1$	$b = 2$
kr-vs-kp	95.69	91.68 ●	94.07 ●	92.98 ●	95.25	93.70 ●	95.68	94.04 ●	95.86	94.34 ●	96.01
iris	94.27	86.53 ●	86.53 ●	92.67	92.73	93.60	93.53	93.00	93.00	93.07	93.07
vote	94.05	88.89 ●	89.67 ●	91.75	92.46	93.68	93.93	93.82	94.14	94.18	94.32
sick	93.88	93.88	93.88	93.88	93.88	93.88	93.88	93.88	93.88	93.88	93.89
hypothyroid	92.86	92.29 ●	92.29 ●	92.29 ●	92.29 ●	92.63	92.60	92.58	92.58	92.65	92.62
ionosphere	91.74	76.97 ●	76.40 ●	84.91 ●	84.55 ●	89.20	89.15	90.06	90.03	90.71	90.71
wisconsin-breast-cancer	91.47	86.69	86.85	90.26	90.72	91.60	92.00	90.74	91.22	90.77	91.06
anneal	90.84	76.89 ●	77.14 ●	80.44 ●	82.61 ●	86.97 ●	89.58	87.36 ●	90.22	87.98 ●	90.49
anneal.ORIG	89.33	77.18 ●	77.28 ●	80.29 ●	80.46 ●	86.04 ●	86.12 ●	87.42 ●	87.72	87.70 ●	87.85
balance-scale	85.44	77.86 ●	77.26 ●	81.57 ●	80.90 ●	83.63	83.38	82.46 ●	81.95 ●	82.99 ●	82.44 ●
credit-rating	84.99	75.49 ●	76.42 ●	81.10 ●	82.01	83.54	84.61	83.61	83.80	83.64	83.78
labor	83.70	67.90 ●	68.83 ●	74.13	73.60	79.77	79.07	77.00	76.00	77.37	77.93
horse-colic	81.82	73.06 ●	74.09 ●	78.36	80.43	81.02	82.28	80.27	82.44	80.38	82.98
soybean	80.43	59.41 ●	61.29 ●	65.01 ●	67.19 ●	70.26 ●	73.35 ●	71.44 ●	74.79 ●	72.74 ●	75.58 ●
hungarian-14-heart-disease	80.34	70.13 ●	69.38 ●	75.88	73.88 ●	79.00	77.27	79.47	77.22	79.61	77.39
hepatitis	79.89	79.75	79.75	79.75	79.75	79.82	79.56	79.59	79.65	79.38	79.51
zoo	77.50	63.45 ●	63.06 ●	68.41 ●	67.82 ●	72.06 ●	71.48 ●	71.70	72.21	71.41	71.42
lymphography	76.49	67.55	69.78	72.33	74.57	74.24	76.41	74.32	75.41	74.40	76.10
cleveland-14-heart-disease	75.56	65.77 ●	69.80	71.67	76.11	73.92	77.75	74.52	77.29	74.35	77.78
segment	73.86	65.57 ●	65.69 ●	69.95 ●	70.06 ●	71.32 ●	71.45 ●	71.53 ●	71.67 ●	71.98 ●	72.21
breast-cancer	72.18	69.87	69.52	70.71	70.61	71.56	71.52	71.15	71.51	70.99	71.34
horse-colic.ORIG	71.92	68.50	68.74	71.15	70.55	73.04	73.61	72.72	72.96	72.56	73.09
german-credit	70.87	70.13	70.26	70.27	70.26	70.43	70.69	70.29	70.66	70.53	70.69
heart-statlog	70.44	63.19 ●	66.04	70.41	74.70	73.30	77.37 ○	72.30	76.00	71.59	74.78
pima-diabetes	69.94	66.92	66.85	67.83	67.74	69.65	69.53	69.60	69.56	69.43	69.45
autos	66.95	53.32 ●	52.21 ●	57.04 ●	56.80 ●	60.45	59.39 ●	58.85 ●	58.60 ●	60.60	60.41
sonar	65.50	52.24 ●	52.24 ●	59.74	59.74	66.02	66.02	64.41	64.41	65.22	65.22
Glass	63.44	52.70 ●	52.61 ●	56.48	56.66	57.26	57.35	58.73	58.82	58.95	58.82
vehicle	52.43	45.21 ●	46.96 ●	49.02	50.67	49.62	52.51	50.90	53.31	51.25	53.70
audiology	48.47	43.53	48.77	44.45	51.20	45.30	51.38	44.32	51.32	44.65	52.81
primary-tumor	41.30	36.20 ●	36.22 ●	38.09	38.52	39.26	39.64	39.38	40.15	39.44	40.56
vowel	29.44	29.11	29.11	29.43	29.41	29.97	30.01	29.62	29.64	29.82	29.71
Average	76.16	68.37	69.03	71.94	72.75	74.24	75.07	74.10	74.94	74.33	75.24

○ statistically significant improvement, ● statistically significant degradation , corrected paired Student's t-Test, $\alpha = 0.05$

Table 10: LazyRule using ubsets of nearest neighbors for $k = 1$, $k = 5$, $k = 10$ and subsets containing 1%, 5% and 10% of the nearest neighbors for training. The results are compared to the basic algorithm (first column). Evaluated on datasets *A*.

Dataset	*all*	$k = 1$	$k = 5$	$k = 10$	$p = 1\%$	$p = 5\%$	$p = 10\%$
iris	94.27	95.53	95.40	94.80	95.53	95.87	94.73
vote	94.05	92.65	93.38	93.47	93.17	93.01	92.57
sick	93.88	95.97 ∘	96.28 ∘	96.31 ∘	96.73 ∘	96.11 ∘	94.62 ∘
hypothyroid	92.86	91.44 •	93.43 ∘	93.53 ∘	93.52 ∘	93.54 ∘	93.89 ∘
ionosphere	91.91	87.10 •	85.36 •	84.73 •	86.02 •	85.30 •	82.62 •
wisconsin-breast-cancer	91.47	95.28 ∘	96.81 ∘	96.55 ∘	96.40 ∘	96.01 ∘	95.62 ∘
anneal	90.75	99.13 ∘	98.24 ∘	98.52 ∘	98.15 ∘	97.31 ∘	96.55 ∘
anneal.ORIG	89.21	95.49 ∘	94.42 ∘	94.56 ∘	94.33 ∘	93.99 ∘	92.86 ∘
labor	85.67	85.83	88.20	89.90	85.83	83.97	88.37
balance-scale	85.44	85.33	86.16	86.79	86.16	87.55	87.92 ∘
credit-rating	84.55	81.43	86.17	86.61	86.39	86.30	86.83
horse-colic	81.79	79.05	82.15	83.01	82.10	82.36	82.47
hungarian-14-heart-disease	80.41	78.33	82.52	82.16	80.48	82.46	83.55
soybean	80.08	91.67 ∘	91.07 ∘	90.17 ∘	90.61 ∘	86.91 ∘	84.64 ∘
hepatitis	79.70	81.40	84.91	84.40	81.40	84.60	83.17
zoo	77.10	96.05 ∘	96.35 ∘	89.10 ∘	96.05 ∘	95.74 ∘	89.79 ∘
lymphography	76.16	81.69	84.11 ∘	80.28	81.69	83.57 ∘	81.99
cleveland-14-heart-disease	75.59	76.19	83.08 ∘	82.95 ∘	77.36	82.06 ∘	82.54 ∘
segment	73.89	97.14 ∘	95.68 ∘	95.44 ∘	95.64 ∘	94.06 ∘	93.27 ∘
breast-cancer	72.15	72.35	73.37	74.18	73.68	73.80	73.20
horse-colic.ORIG	71.92	65.24 •	63.02 •	65.11 •	64.16 •	67.99	71.16
german-credit	70.84	71.87	73.11	73.73 ∘	73.96 ∘	72.75 ∘	72.50
heart-statlog	70.63	76.59	79.33 ∘	80.67 ∘	77.63 ∘	81.00 ∘	82.56 ∘
pima-diabetes	69.91	70.52	73.84 ∘	73.41 ∘	73.25 ∘	74.93 ∘	74.19 ∘
autos	65.45	74.65 ∘	68.81	68.51	74.65 ∘	67.43	69.20
sonar	64.14	85.88 ∘	82.42 ∘	75.10 ∘	85.88 ∘	74.39 ∘	70.55
Glass	61.10	69.15 ∘	66.77	65.51	69.15 ∘	64.96	68.89 ∘
vehicle	52.74	69.62 ∘	70.56 ∘	70.66 ∘	70.74 ∘	67.25 ∘	65.96 ∘
audiology	48.30	78.39 ∘	66.02 ∘	63.26 ∘	72.30 ∘	63.17 ∘	59.58 ∘
primary-tumor	41.18	38.97	43.45	44.49	41.56	44.69	45.05 ∘
vowel	29.42	99.06 ∘	93.67 ∘	71.31 ∘	82.59 ∘	46.29 ∘	60.99 ∘
Average	75.37	82.55	82.84	81.59	82.49	80.63	80.70

∘ statistically significant improvement, • statistically significant degradation , $\alpha = 0.05$

Table 11: LazyRuleNN and different values for k compared to the highest ranked ($k = 5$) in the first column. Evaluated on datasets *D*.

Dataset	$k = 5$	$k = 1$	$k = 2$	$k = 3$	$k = 10$	$k = 15$	$k = 25$
mushroom	100.00	100.00	100.00	100.00	99.98	99.96	99.94
anneal	98.24	99.13	97.58	97.44 ●	98.52	98.11	97.63
kr-vs-kp	97.10	96.13 ●	96.24 ●	96.92	96.93	97.05	97.10
wisconsin-breast-cancer	96.85	95.28 ●	94.81 ●	96.60	96.58	96.24	95.95
sick	96.28	95.97	95.76 ●	96.12	96.31	96.49	96.52
zoo	96.15	96.05	96.05	96.05	89.50 ●	90.90 ●	88.34 ●
segment	95.68	97.14 ○	96.13	96.25	95.44	95.34	95.73
letter	95.62	96.02 ○	94.87 ●	95.69	95.09 ●	94.61 ●	93.51 ●
iris	95.40	95.53	95.53	95.33	95.00	94.40	94.33
anneal.ORIG	94.42	95.49	94.98	94.48	94.56	94.53	94.59
vowel	93.67	99.06 ○	97.76 ○	96.99 ○	71.31 ●	55.57 ●	50.09 ●
hypothyroid	93.43	91.44 ●	93.43	93.25	93.53	93.49	93.40
vote	93.36	92.69	93.17	93.08	93.45	93.06	92.76
soybean	91.05	91.67	91.77	91.48	90.16	89.15 ●	87.62 ●
labor	88.20	85.83	83.97	93.00	90.40	89.13	85.57
credit-rating	86.19	81.43 ●	81.70 ●	84.83	86.59	86.93	86.36
balance-scale	86.16	85.33	85.31	85.50	86.77	86.90	87.14
ionosphere	85.36	87.10	89.77 ○	86.02	84.73	85.30	83.22
hepatitis	84.91	81.40	76.42 ●	81.10	84.47	83.97	84.35
lymphography	84.17	81.69	81.44	82.15	80.48	81.68	78.37 ●
cleveland-14-heart-disease	83.08	76.19 ●	77.36 ●	81.59	82.88	81.66	82.61
hungarian-14-heart-disease	82.52	78.33 ●	80.48	82.33	82.06	82.70	83.58
sonar	82.42	85.88	86.70	83.57	75.10 ●	70.62 ●	72.09 ●
horse-colic	82.23	79.05	82.10	82.10	83.04	82.03	82.25
splice	80.40	74.49 ●	72.39 ●	77.92 ●	82.46 ○	82.50 ○	83.39 ○
heart-statlog	79.33	76.59	77.63	79.07	80.78	81.44	82.78
waveform	79.21	73.45 ●	72.40 ●	77.67 ●	80.69 ○	82.31 ○	83.30 ○
pima-diabetes	73.85	70.52 ●	71.39 ●	73.84	73.41	74.28	74.06
breast-cancer	73.34	72.39	73.71	72.49	74.36	73.91	73.31
german-credit	73.14	71.87	72.26	72.26	73.72	73.55	73.46
vehicle	70.55	69.62	67.63	70.30	70.71	69.36	67.89
autos	68.81	74.65	70.43	70.82	68.51	71.48	68.38
Glass	66.77	69.15	67.74	70.43	65.51	65.79	70.58
audiology	66.06	78.39 ○	72.26 ○	71.63 ○	63.30	61.57	58.69 ●
horse-colic.ORIG	63.07	65.24	68.47 ○	64.16	65.01	68.32	70.76 ○
primary-tumor	43.39	38.97 ●	39.23 ●	41.41	44.67	44.46	45.43
Average	83.90	83.31	83.02	84.00	82.94	82.47	82.09

○ statistically significant improvement, ● statistically significant degradation , $\alpha = 0.05$

Table 12: LazyRuleNN with $k = 5$ compared to different values for k for k-nearest neighbor algorithm. Evaluated on datasets *D*.

Dataset	LR-5	kNN-1	kNN-2	kNN-3	kNN-5	kNN-10	kNN-15	kNN-25
mushroom	100.00	100.00	100.00	100.00	100.00	99.92 ●	99.88 ●	99.83 ●
anneal	98.24	99.13	97.52	97.29 ●	97.27 ●	96.09 ●	95.27 ●	92.64 ●
kr-vs-kp	97.10	96.12 ●	96.15 ●	96.56 ●	96.16 ●	95.04 ●	94.55 ●	93.22 ●
wisconsin-breast-cancer	96.85	95.28 ●	94.81 ●	96.60	96.93	96.62	96.58	96.27
sick	96.28	96.10	95.86 ●	96.21	96.28	96.03	95.80	95.47 ●
zoo	96.15	96.05	92.61	92.61	95.05	88.71 ●	88.53 ●	81.69 ●
segment	95.68	97.15 ○	95.82	96.12	95.25 ●	94.55 ●	94.36 ●	93.58 ●
letter	95.62	96.01 ○	94.82 ●	95.62	95.50 ●	94.76 ●	94.11 ●	92.81 ●
iris	95.40	95.40	95.53	95.20	95.73	95.73	96.13	95.33
anneal.ORIG	94.42	95.49	94.89	93.77	93.23 ●	90.97 ●	89.43 ●	87.34 ●
vowel	93.67	99.05 ○	97.77 ○	96.99 ○	93.39	58.96 ●	23.68 ●	3.58 ●
hypothyroid	93.43	91.52 ●	93.33	93.21	93.31	93.22	92.95 ●	92.82 ●
vote	93.36	92.58	93.10	93.08	93.17	92.94	92.23	91.43 ●
soybean	91.05	91.20	91.71	91.20	90.12	87.20 ●	84.50 ●	76.48 ●
labor	88.20	84.30	83.97	92.83	87.53	88.70	85.83	75.93 ●
credit-rating	86.19	81.57 ●	81.42 ●	84.96	86.13	86.12	85.75	86.06
balance-scale	86.16	86.72	86.72	86.74	87.97 ○	90.26 ○	90.14 ○	89.30 ○
ionosphere	85.36	87.10	89.77 ○	86.02	85.10	84.87	84.25	79.43 ●
hepatitis	84.91	81.40	76.73 ●	80.85	84.93	83.32	81.96	81.39
lymphography	84.17	81.69	81.50	81.74	84.18	81.19	82.48	79.78
cleveland-14-heart-disease	83.08	76.06 ●	77.20 ●	81.82	82.13	82.31	82.22	83.01
hungarian-14-heart-disease	82.52	78.33 ●	80.37	82.33	82.32	82.63	81.94	81.46
sonar	82.42	86.17	86.11	83.76	82.28	75.25	68.77 ●	70.26 ●
horse-colic	82.23	79.11	81.22	80.95	81.88	82.99	81.79	82.60
splice	80.40	74.43 ●	72.14 ●	77.59 ●	79.86 ●	83.48 ○	85.15 ○	87.12 ○
heart-statlog	79.33	76.15	77.70	79.11	79.89	81.30	81.37	82.48
waveform	79.21	73.41 ●	72.35 ●	77.67 ●	79.29	80.46 ○	82.46 ○	83.78 ○
pima-diabetes	73.85	70.62 ●	71.40 ●	73.86	73.86	72.94	74.38	74.20
breast-cancer	73.34	72.85	73.19	73.13	74.00	73.44	73.31	73.06
german-credit	73.14	71.88	72.18	72.21	73.17	73.93	73.54	73.71
vehicle	70.55	69.59	67.63	70.21	70.17	69.90	69.07	68.22
autos	68.81	74.55	66.26	67.23	62.36 ●	59.99 ●	57.82 ●	52.43 ●
Glass	66.77	69.95	67.46	70.02	66.04	63.26	62.36	61.86
audiology	66.06	78.43 ○	68.81	67.97	62.31	55.42 ●	54.44 ●	47.75 ●
horse-colic.ORIG	63.07	65.18	67.68	63.94	60.92 ●	62.83	64.41	63.34
primary-tumor	43.39	39.91	43.21	44.98	47.32 ○	46.96	46.75	44.87
Average	83.90	83.35	82.75	83.73	83.47	81.73	80.23	78.18

○ statistically significant improvement, ● statistically significant degradation , $\alpha = 0.05$

Table 13: LazyRuleNN with $k = 5$ compared to different values for k for weighted k-nearest neighbor algorithm. Evaluated on datasets *D*.

Dataset	LR-5	w.NN-2	w.NN-3	w.NN-5	w.NN-10	w.NN-15	w.NN-25
mushroom	100.00	100.00	100.00	100.00	99.96	99.93 •	99.90 •
anneal	98.24	99.12	98.02	98.14	98.20	97.60	96.92 •
kr-vs-kp	97.10	96.23 •	96.67 •	96.44 •	95.67 •	95.21 •	94.17 •
wisconsin-breast-cancer	96.85	95.45 •	96.64	97.03	96.93	96.67	96.28
sick	96.28	96.10	96.21	96.28	96.30	95.78	95.43 •
zoo	96.15	93.10	92.61	95.05	90.90 •	89.81 •	86.94 •
segment	95.68	97.03 ◦	96.51 ◦	96.15 ◦	95.90	95.85	95.57
letter	95.62	95.96 ◦	96.08 ◦	96.01 ◦	95.54	95.01 •	94.07 •
iris	95.40	95.40	95.20	95.60	95.27	95.73	95.60
anneal.ORIG	94.42	95.22	93.86	93.15 •	91.39 •	89.37 •	87.31 •
vowel	93.67	99.05 ◦	97.51 ◦	96.29 ◦	93.40	92.48	92.79
hypothyroid	93.43	91.55 •	93.35	93.40	93.38	93.04 •	92.85 •
vote	93.36	93.03	93.01	93.08	92.92	92.21	91.47 •
soybean	91.05	91.61	91.13	90.16	88.26 •	86.91 •	80.60 •
labor	88.20	84.30	92.83	87.53	89.43	86.90	78.00
credit-rating	86.19	81.57 •	84.57	86.13	86.54	86.35	86.77
balance-scale	86.16	86.72	86.74	87.98 ◦	90.27 ◦	90.15 ◦	89.49 ◦
ionosphere	85.36	87.41	86.02	85.33	84.27	84.36	80.66 •
hepatitis	84.91	81.40	80.72 •	84.47	83.53	82.73	81.98
lymphography	84.17	83.57	83.69	85.05	82.47	82.61	80.73
cleveland-14-heart-disease	83.08	76.06 •	80.34	81.86	82.02	83.38	83.44
hungarian-14-heart-disease	82.52	78.33 •	82.33	82.32	82.15	81.94	81.46
sonar	82.42	86.17	83.90	83.18	77.80	72.56 •	72.37 •
horse-colic	82.23	79.19	81.01	81.93	82.39	81.87	82.60
splice	80.40	75.77 •	79.62	82.14 ◦	85.09 ◦	86.32 ◦	87.85 ◦
heart-statlog	79.33	76.15	78.56	79.81	81.07	81.15	82.15
waveform	79.21	73.41 •	77.68 •	79.33	81.10 ◦	82.49 ◦	83.79 ◦
pima-diabetes	73.85	70.62 •	73.81	73.81	73.83	74.92	74.80
breast-cancer	73.34	73.30	72.88	73.86	73.62	73.35	73.03
german-credit	73.14	71.88	72.13	73.30	74.57	73.93	73.98
vehicle	70.55	69.59	70.75	71.41	70.16	69.22	68.60
autos	68.81	74.55	75.08 ◦	72.79	70.50	70.24	67.79
Glass	66.77	69.95	71.32	71.26 ◦	67.63	67.03	65.77
audiology	66.06	73.19 ◦	71.42 ◦	65.81	60.46 •	59.57 •	52.75 •
horse-colic.ORIG	63.07	64.96	63.86	61.98	63.56	65.16	64.02
primary-tumor	43.39	41.47	43.62	45.66	46.46	46.52	45.10
Average	83.90	83.29	84.16	84.27	83.69	83.29	82.14

◦ statistically significant improvement, • statistically significant degradation , $\alpha = 0.05$

Table 14: LazyRuleNN with $k = 5$ compared to separate-and-conquer algorithm (JRip) and decision tree algorithm (J48). Evaluated on datasets *D*.

Dataset	LR-5	JRip	J48
mushroom	100.00	100.00	100.00
anneal	98.24	98.26	98.57
kr-vs-kp	97.10	99.21 ○	99.44 ○
wisconsin-breast-cancer	96.85	95.61	95.01 ●
sick	96.28	98.29 ○	98.72 ○
zoo	96.15	86.62 ●	92.61
segment	95.68	95.47	96.79 ○
letter	95.62	86.31 ●	88.03 ●
iris	95.40	93.93	94.73
anneal.ORIG	94.42	95.03	92.35
vowel	93.67	70.39 ●	80.20 ●
hypothyroid	93.43	99.42 ○	99.54 ○
vote	93.36	95.75 ○	96.57 ○
soybean	91.05	91.80	91.78
labor	88.20	83.70	78.60
credit-rating	86.19	85.16	85.57
balance-scale	86.16	80.30 ●	77.82 ●
ionosphere	85.36	89.16	89.74 ○
hepatitis	84.91	78.13 ●	79.22 ●
lymphography	84.17	76.31	75.84 ●
cleveland-14-heart-disease	83.08	79.95	76.94 ●
hungarian-14-heart-disease	82.52	79.57	80.22
sonar	82.42	73.40 ●	73.61 ●
horse-colic	82.23	85.10	85.16
splice	80.40	94.20 ○	94.03 ○
heart-statlog	79.33	78.70	78.15
waveform	79.21	79.30	75.25 ●
pima-diabetes	73.85	75.18	74.49
breast-cancer	73.34	71.45	74.28
german-credit	73.14	72.21	71.25
vehicle	70.55	68.32	72.28
autos	68.81	73.62	81.77 ○
Glass	66.77	66.78	67.63
audiology	66.06	73.10 ○	77.26 ○
horse-colic.ORIG	63.07	82.82 ○	66.31
primary-tumor	43.39	38.74	41.39
Average	83.90	83.09	83.37

○, ● statistically significant improvement/degradation , $\alpha = 0.05$

Table 15: Average rule lengths for LazyRule and LazyRuleNN. Evaluated on datasets *A*.

Dataset	LazyRule	LazyRuleNN, $>$, $k = 5$	LazyRuleNN, $\geq$, $k = 5$
zoo	1.68	0.24	15.26
labor	1.80	0.95	8.29
sonar	2.00	0.99	49.28
ionosphere	2.00	0.51	36.30
vowel	2.03	1.19	13.59
hypothyroid	2.10	0.40	25.14
segment	2.16	0.24	22.95
Glass	2.18	1.35	11.46
anneal.ORIG	2.25	0.28	20.68
autos	2.31	1.55	34.67
iris	2.33	0.29	3.72
anneal	2.34	0.21	44.85
hepatitis	2.40	0.71	13.44
pima-diabetes	2.48	1.32	8.05
sick	2.51	0.21	24.64
vote	2.54	0.30	9.81
horse-colic.ORIG	2.54	1.44	12.87
soybean	2.72	0.48	25.05
horse-colic	2.78	0.67	8.20
credit-rating	3.07	0.84	13.46
breast-cancer	3.11	1.41	4.83
lymphography	3.18	0.84	14.78
german-credit	3.30	1.17	20.02
cleveland-14-heart-disease	3.32	0.97	11.34
hungarian-14-heart-disease	3.45	0.84	10.34
vehicle	3.62	1.38	16.86
wisconsin-breast-cancer	3.78	0.23	16.45
audiology	4.50	1.29	61.98
heart-statlog	4.81	0.98	28.49
primary-tumor	4.93	2.24	13.60
balance-scale	4.98	2.05	5.90
Average	2.88	0.89	19.56

Table 16: Percent of empty rules learned for LazyRule and LazyRuleNN. Evaluated on datasets *A*.

Dataset	LazyRule	LazyRuleNN, $>$, $k = 5$	LazyRuleNN, $\geq$, $k = 5$
vehicle	0.00	31.19	0.98
audiology	0.00	24.38	0.00
ionosphere	0.00	74.36	0.00
horse-colic	0.00	56.17	0.00
segment	0.00	87.84	0.00
cleveland-14-heart-disease	0.00	48.48	0.00
horse-colic.ORIG	0.00	15.11	0.00
german-credit	0.00	31.79	0.00
anneal	0.00	88.13	0.00
sick	0.00	89.36	0.00
soybean	0.00	65.37	0.00
vowel	0.00	40.56	0.00
anneal.ORIG	0.00	85.28	0.00
sonar	0.00	50.64	0.00
hepatitis	0.00	60.01	0.00
zoo	0.00	83.22	0.00
wisconsin-breast-cancer	0.00	88.96	0.36
Glass	0.00	32.67	0.00
labor	0.00	48.03	1.57
iris	0.00	85.53	28.80
heart-statlog	0.00	51.04	0.00
balance-scale	0.00	56.91	13.45
primary-tumor	0.00	14.09	0.00
hungarian-14-heart-disease	0.00	57.35	0.00
credit-rating	0.00	54.36	0.00
breast-cancer	0.00	16.03	0.00
lymphography	0.00	37.36	0.00
autos	0.00	20.21	0.00
hypothyroid	0.00	79.83	0.00
pima-diabetes	0.09	34.23	9.53
vote	0.23	78.20	0.51
Average	0.01	54.41	1.78

References

David W. Aha and Dennis Kibler. Instance-based learning algorithms. In *Machine Learning*, pages 37–66, 1991.

Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, NY, 1984.

William W. Cohen. Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, january 1967. ISSN 0018-9448. doi: 10.1109/TIT.1967.1053964.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, December 2006. ISSN 1532-4435.

Xiaoli Zhang Fern and Carla E. Brodley. Boosting lazy decision trees. In *Proceedings of the Twentieth International Conference on Machine Learning, August 2003 Washington D.C*, pages 178–185, 2003.

Jerome H. Friedman. Lazy decision trees. In *Proceedings of the thirteenth national conference on Artificial intelligence - Volume 1*, AAAI'96, pages 717–724. AAAI Press, 1996. ISBN 0-262-51091-X.

Johannes Fürnkranz. Pruning algorithms for rule learning. In *Machine Learning*, pages 139–171, 1997.

Johannes Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13:3–54, 1999.

Johannes Fürnkranz. A pathology of bottom-up hill-climbing in inductive rule learning. In *Proceedings of the Thirteenth European Conference on Algorithmic Learning Theory*, pages 263–277. Springer-Verlag, 2002.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18, November 2009. ISSN 1931-0145. doi: http://doi.acm.org/10.1145/1656274.1656278.

Frederik Janssen and Johannes Fürnkranz. The seco-framework for rule learning. Technical Report TUD-KE-2010-02, TU Darmstadt, Knowledge Engineering Group, 2010a. URL `http://www.ke.tu-darmstadt.de/publications/reports/tud-ke-2010-02.pdf`.

Frederik Janssen and Johannes Fürnkranz. On the quest for optimal rule learning heuristics. *Machine Learning*, 78(3): 343–379, March 2010b. doi: 10.1007/s10994-009-5162-2. URL `http://www.ke.tu-darmstadt.de/publications/papers/ML2010.pdf`.

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143. Morgan Kaufmann, 1995.

Dragos D. Margineantu and Thomas G. Dietterich. Improved class probability estimates from decision tree models, 2001.

Tom Michael Mitchell. *Machine Learning*. McGraw-Hill Series in Computer Science. WCB/McGraw-Hill, Boston, MA, 1997. ISBN 0-07-042807-7.

Claude Nadeau and Yoshua Bengio. Inference for the generalization error. *Mach. Learn.*, 52:239–281, September 2003. ISSN 0885-6125. doi: 10.1023/A:1024068626366.

Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Mach. Learn.*, 5:71–99, May 1990. ISSN 0885-6125. doi: 10.1023/A:1022611825350.

Heiko Paulheim. Explain-a-lod: Using linked open data for interpreting statistics. In *2012 International Conference on Intelligent User Interfaces (IUI 2012)*, to appear.

Heiko Paulheim and Johannes Fürnkranz. Unsupervised generation of data mining features from linked open data. Technical Report TUD-KE-2011-2, Knowledge Engineering Group, Technische Universität Darmstadt, 2011.

R.B. Pereira, A. Plastino, B. Zadrozny, L.H.C. Merschmann, and A.A. Freitas. Lazy attribute selection: Choosing attributes at classification time. *Intelligent Data Analysis*, 15(5):715–732, September 2011a. ISSN 1088467X. URL `http://www.cs.kent.ac.uk/pubs/2011/3158`.

R.B. Pereira, A. Plastino, B. Zadrozny, L.H.C. Merschmann, and A.A. Freitas. Improving lazy attribute selection. *Journal of Information and Data Management*, 2(3):447–462, October 2011b. ISSN 2178-7107. URL `http://www.cs.kent.ac.uk/pubs/2011/3170`.

J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

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

Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 0120884070.

List of Figures

List of Tables

List of Algorithms