Separate-and-conquer Regression

Frederik Janssen and Johannes Fürnkranz TU Darmstadt, Knowledge Engineering Group Hochschulstraße 10, D-64289 Darmstadt {janssen,juffi}@ke.tu-darmstadt.de

Abstract

In this paper a rule learning algorithm for the prediction of a numerical target variable is presented. It is based on the separate-and-conquer strategy and the classification phase is done by a decision list. A new splitpoint generation method is introduced for the efficient handling of numerical attributes. It is shown that the algorithm performs comparable to other regression algorithms where some of them are based on rules and some are not. Additionally a novel heuristic for evaluating the trade-off between consistency and generality of regression rules is introduced. This heuristic features a parameter to directly trade off the rule's consistency and its generality. We present an optimal setting for this parameter based on an optimization on several data sets.

1 Introduction

The accurate prediction of a numerical target variable is an important task in machine learning. There are several domains that can benefit from regression methods. For example, in the domain of financial data, it is a crucial issue to predict the volume of a credit. Here, classification algorithms can only provide a decision of whether or not a credit should be given but are not capable of predicting its size.

In the machine learning community the main task still is to predict a categorical outcome but through the last years the task of regression has gained more and more interest. Regression has its roots in the statistical community from where several algorithms were proposed over the years. The list includes the popular linear regression that is very efficient but still shows a good performance. The main advantage in using means of machine learning lies in the comprehensibility of the models. For instance, simple IF-THEN rules are directly interpretable by a data miner. Rules and trees are the two variants of interpretable models used in machine learning. As rules are typically more expressive because they are able to overlap, the goal of this work is the design of a rule learning system that on the one hand has a performance that is comparable to state-of-the-art algorithms and that on the other hand yields models that are still human-readable.

There are several strategies to induce a set of rules. Some of them rely on the gradient-descent algorithm for finding a rule ensemble that optimizes some loss function. Others convert given trees into sets of rules. However, one of the most popular strategy in classification is the so-called separate-and-conquer paradigm. Due to its simplicity and its good performance in classification¹, we decided to use this strategy to design the algorithm.

The paper is started with a brief recapitulation of related work. It is continued by a short introduction of separateand-conquer rule learning for classification. Then the adaptations that are necessary to extent separate-and-conquer rule learning for classification to regression are specified. Some error measures are introduced and the handling of numerical attributes is described. Then the experimental setup and the evaluation methods are specified and the method for optimizing the parameters of the algorithm is described. The following section describes the results and the last one concludes the paper.

2 Related work

The separate-and-conquer strategy is not used frequently for learning regression rules. Exceptions include *predictive clustering rules* (PCR) [Ženko *et al.*, 2005], the FRS system [Demšar, 1999], which is a reimplementation of the FORS system [Karalič and Bratko, 1997], and M5RULES [Holmes *et al.*, 1999; Quinlan, 1992; Wang and Witten, 1997] which generates the regression rules from model trees and uses linear models in the head of the rules. Predictive clustering rules are generated by modifying the search heuristic of CN2 [Clark and Niblett, 1989]. Instead of *accuracy* or *weighted relative accuracy*, it uses a heuristic that is based on the dispersion of the data. This algorithm also follows a different route by joining clustering approaches with predictive learning.

The R^2 system [Torgo, 1995] works to some extent analogously to other separate-and-conquer algorithms by selecting an uncovered region of the input data. But this selection differs from the mechanism used in regular separate-and-conquer learning. However, it also allows for rules to overlap and the rules predict linear models instead of a single target value.

Other mechanisms for learning regression rules are mainly based on ensemble techniques as used in the RULE-FIT learning algorithm [Friedman and Popescu, 2008] or in REGENDER [Dembczyński *et al.*, 2008]. The first algorithm performs a gradient descent optimization, allows the rules to overlap, and the final prediction is calculated by the sum of all predicted values of the covering rules instead of that of a single rule. The second one uses a forward stagewise additive modeling.

¹The famous RIPPER algorithm [Cohen, 1995], one of the most accurate rule learners for classification is also based on the separate-and-conquer paradigm.

Another popular technique to deal with a continuous target attribute is to discretize the numeric values as a preprocessing step and afterwards employ regular machine learning methods for classification. Research following this path can be found in [Torgo and Gama, 1996; Weiss and Indurkhya, 1995]. The main problem here is that the number of bags for the discretization process is not known in advance. For this reason the performance of this technique strongly depends on the choice of the number of classes.

3 Separate-and-conquer rule learning and Regression

Most inductive rule learning algorithms for classification employ a separate-and-conquer strategy for learning rules that allow to map the examples to their respective classes. The basic idea of the separate-and-conquer strategy [Fürnkranz, 1999] is to cover a part of the example space that is not explained by any rule yet (the conquer step). This region is covered by searching for a rule that fulfills some properties, i.e., has a low error on this partition of the input space. After this rule is found, it is added to a set of rules, and all examples that are covered by the rule are removed from the data set (the separate step). Then, the next rule is searched on the remaining examples. This procedure lasts as long as (positive) examples are left. The two constraints that all examples have to be covered (also called *completeness*) and that no negative example has to be covered in the binary case (consistency) can be relaxed so that examples remain uncovered in the data or negative examples are covered by the set of rules. This relaxation mostly is driven from preventing overfitting.

In the end, the algorithm returns a set of subsequently learned rules. For classification of unseen examples, each of the rules in the list is tested whether or not it covers the example. The first rule that covers the example (i.e., matches all the given attribute values) "fires" and predicts the value of the example by using the head of the rule. If no rule in the (decision) list covers the example, the prediction is given by a default rule that usually predicts the majority class in the data.

In the following we will have a closer look at the main step of the algorithm², namely how to navigate through the search space. Most of the algorithms build all possible candidate rules from the data by using all values for a given attribute and include these attribute-value pairs in a candidate rule. Thus, an attribute-value pair (a condition) is added to a given candidate rule which results in a refined candidate rule, i.e., a refinement of the former candidate rule. For nominal attributes these values are given from the data itself but for numerical attributes usually all possible splitpoints are used. The splitpoints are calculated as the mean between two adjacent (previously sorted) values.

Finally, when all candidates with one condition are generated a heuristic is used to determine the best one. Then, the best candidate rule is stored and refined to yield all refinements with two conditions. For nominal attributes the used ones are stored (and not used any more) and for numerical ones the relations < and \ge are evaluated. This means that a numerical attribute may occur twice in one rule by using it for a test on < and on \ge . This procedure usually runs as long as negative examples are covered. In this step the algorithm also ensures that a minimum number of examples is covered (a user-given value). For all experiments (cf. Section 5 and 6) we fixed the minimum coverage to 3 examples.

Note that missing attribute values can never be covered and the attribute with the missing value is ignored. When the class of an instance is missing it is removed from the dataset in a preprocessing step. As search strategy simple hill-climbing was used.

There are many different heuristics to navigate the search (for an overview see [Fürnkranz and Flach, 2005]) but all of them are trying to maximize the coverage of positive examples (p) and to minimize the negative coverage (n). To reach this objective different ways are employed but usually, in some way, there has to be a combination of consistency (i.e., the error or the negative coverage) of the rule and its generality (i.e., the number of examples that are covered). Most of the heuristics have a fixed trade-off but some of them feature a parameter to adjust it. In previous work the parameters of some of these heuristics were tuned, so that they achieved the most accurate trade-off between consistency and coverage [Janssen and Fürnkranz, 2010a]. In this work we follow the same path by defining such a parametrized heuristic and by tuning its parameter to yield the best fit between these two objectives.

3.1 Separate-and-conquer for regression

As noted above some of the properties that come with categorical binary data do not apply for numerical target variables. Thus the algorithm had to be adapted in several ways. First of all, each evaluation of a single splitpoint requires a scan through the data. For this reason a novel splitpoint method has to be developed that allows using only a subset of all splitpoints to prevent the algorithm from getting too inefficient (cf. Section 3.3). Mechanisms for the efficient computation of splitpoints known from classification proved to be inefficient in our first experiments.

The heuristics that were introduced for the task of classification were not suitable for regression either. In Regression there is no notion for positive or negative examples. Hence, an alternative error measure has to be defined. The default rule also has to be adapted because there is no majority class any more. A simple way to do this is to take the mean over all remaining examples as prediction. Another way would be to take the mean of all examples. We experimented with both settings (cf. Section 6). Finally, the methods for evaluating the final model have to be adapted because using measures like *accuracy* (the percentage of correctly predicted examples) is not practicable any more.

3.2 Error measures for regression

There are several ways to compute the error of a rule or of a complete model for regression tasks. This section gives an overview of the measures we used.

In the following m denotes the total number of examples in the (current) dataset, y is the value of the current example, \bar{y} is the value predicted by the rule, and y' is the mean over all examples.

The *mean absolute error* is the mean of the sum of the absolute errors of all examples that are covered by the rule

$$L_{MAE} = \frac{1}{m} \sum_{i=1}^{m} |y_i - \bar{y}_i|$$
(1)

The *root mean squared error* is defined by taking the root of the *mean squared error*

²For pseudo-code of the algorithm see [Janssen and Fürnkranz, 2010b].

$$L_{RMSE} = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (y_i - \bar{y}_i)^2}$$
(2)

The problem of L_{RMSE} is that it is domain-dependent. As the amplitude of the values in the domain is changing the amplitude in the error measures is changing as well. Thus, the errors are not comparable among different datasets. For using this measures to compute the heuristic value this may not be a problem because only candidate rules are compared to each other. But if a combination of the error and the coverage is taken this becomes crucial due to normalization issues.

For the normalization of the L_{RMSE} usually the deviation from the mean is used which is given by

$$L_{default} = \sum_{i=1}^{m} (y_i - y')^2.$$
 (3)

Thus, the *relative root mean squared error*³ becomes

$$L_{RRMSE} = \frac{L_{RMSE}}{\sqrt{\frac{1}{m} \cdot L_{default}}}.$$
 (4)

These measures can be used for evaluating a single candidate rule but also for evaluating a whole theory (an ordered set of rules). Note that the L_{RRMSE} has its best value at 0 when each example is classified with the correct value. Theoretically, its worst value is 1 because only the value calculated by $L_{default}$ was used for predicting the value of an unseen example. Due to the split in 10 folds that happens during the cross-validation the values of the L_{RRMSE} can become bigger than 1. A stratification of regression values is not possible and this may results in splits where the test fold contains examples that share values that have never appeared in the training fold. In these cases the prediction with the value of $L_{default}$ would have been superior to the values predicted by the learned model. Note that some databases are practically even encode randomness because the L_{RRMSE} values on those where always bigger than 1 independently from the used algorithm.

To derive the *relative coverage* the number of covered examples divided by the total number of examples is taken.

$$relCov = \frac{1}{m} \cdot coverage(Rule).$$
 (5)

We decided to combine the error and the generality of a rule by using the *rrmse* and the *relative coverage*

$$h_{cm} = \alpha \cdot (1 - L_{RRMSE}) + (1 - \alpha) \cdot relCov.$$
 (6)

Here, the parameter α enables a trade-off between the error and the generality of the rule. For $\alpha = 1$ the relative coverage is ignored and thus the rules are evaluated solely by inspecting their error. This setting would yield a model that consists only of rules that cover a single example in the data and would thus clearly lead to overfitting⁴. The other extreme is to set $\alpha = 0$ which results in ignoring the error of the rule. A model built with this setting would only consist of the default rule, because its coverage is the highest that could be achieved by any rule. The optimal trade-off lies somewhere in between these two extremes.

Figure 1: Example of the splitpoint clustering method



The heuristic h_{cm} is an adaptation of a previously introduced heuristic called *relative cost measure* [Fürnkranz and Flach, 2005]. Its formula is given by

$$h_{cr} = c_r \cdot \frac{p}{P} - (1 - c_r) \cdot \frac{n}{N}.$$
(7)

where p is the positive coverage of the rule, n is the negative coverage of the rule, P is the total number of positives and N is the total number of negatives.

It was designed for evaluating classification rules thus relying on coverage statistics. In previous work [Janssen and Fürnkranz, 2010a] an optimal setting for the parameter of h_{cr} was found ($c_r = 0.342$). It encodes a clear favor of the consistency (explained by $\frac{p}{P}$) over the coverage (denoted by $\frac{n}{N}$). It achieved good performance among different classification heuristics as shown in [Janssen and Fürnkranz, 2010a] since it was the second best heuristic of all. Thus the motivation to modify exactly this heuristic was the good performance and that it is best suited to be adapted to regression.

3.3 Splitpoint processing

As noted above, the generation of all possible splitpoints would be too costly. To avoid this a method to restrict the splitpoints for an attribute was developed. The basic idea comes from supervised clustering. Thus, we try to identify regions in the data of the current attribute that share a small error computed on the target variable. The aim of the clustering is to yield partitions of the attribute that share a low error in the hope that the error of a rule that covers these regions will also be low. Clustering stems from the same motivation because it also guarantees that each cluster has the lowest possible error. The user has to define how many clusters and hence how many splitpoints are desired. We experimented with different settings but surprisingly a rather low number of splitpoints seemed to be sufficient (cf. Section 5.1).

Figure 1 displays how the cluster algorithm works. In the example in Figure 1 the attribute has 10 values moving equidistantly from 1 to 10. The values depicted in blue are those of the target attribute of the respective example. In the first step the attribute values are ordered ascending and each value becomes a cluster containing exactly this value. Then two adjacent clusters are searched for which the error when using the mean of the two target values as prediction is the lowest. In the example these are the clusters 2 and 3, 7 and 8, and 8 and 9. Though the objective in the first step is to join two adjacent clusters both 2, 3 and 7, 8 are joined (its arbitrary whether to join 7 and 8 or 8 and 9). The mean of the first cluster is $3.5 = \frac{4+3}{2}$ and the second one has a mean of 1.5 (depicted in black in Figure 1 above the number ray). If the mean absolute error is taken, both clusters have an error of 0.5, which is shown in brackets and in red in the corresponding figure. An error of $0.5 = \frac{|4-3.5|+|3-3.5|}{2}$ is also the lowest error that can be achieved given the example data.

³In the remainder of the paper abbreviated with *rrmse*.

⁴Note that this holds only in a scenario where a rule may cover a single example.

In the second step the function is executed recursively and again those clusters are joined that have the lowest error among all possible clusters. So, in this step, cluster 1 is joined with the second cluster and the cluster with a value of 9 is joined with the third cluster. The error of both clusters grows to 0.67 because adding the respective example does yield a raise of the error (i.e., L_{MAE} = $\frac{|2-3|\bar{+}|4-3|+|3-3|}{2} = 0.67$ for the first cluster). Joining any of the untouched clusters leads to a higher error which means that the cluster with next lowest error is built in step 3. After the second step two clusters containing at least 2 examples were built and therefore 5 splitpoints exist. In the example the user given number of splitpoints is set to 4. Hence another cluster has to be built until the algorithm is finished. This last cluster is derived by joining the clusters with the values 4 and 5 and it yields an error of 0.75.

After the third step 3 clusters are built and the splitpoints are simply derived by taking the mean between the values of two adjacent clusters or two values if the cluster contains only one example. The 4 splitpoints are 3.5, 5.5, 6.5, and 9.5 (depicted in red in Figure 1 in the number ray). We have evaluated the effectiveness of the splitpoint method by comparing it to the usage of another splitpoint method where *n* splitpoints are selected equidistantly. The results of this comparison are shown in Section 6.1. For the computation of the error the *mean absolute error* was used. This choice is arbitrary but experiments with the *root mean squared error* did not yield any performance difference.

3.4 Parameters of the algorithm

There are 3 parameters the user has to specify.

- The parameter of the heuristic,
- the number of splitpoints (splitpoint-parameter), and
- the percentage of examples that are left uncovered (*left-out*-parameter).

The parameter of the heuristic is optimized with a greedy procedure that narrows down the region of interest. This procedure is described in detail in Section 5.

The number of splitpoints is crucial for the runtime of the algorithm. This value was optimized by testing different values (cf. Section 5.1).

The last user-given parameter is the percentage of examples that are left uncovered by the outer loop of the algorithm. This parameter clearly depends on the dataset. During the experiments there was some evidence that we had included databases that basically encode randomness and for those learning anything results in worse performance (e.g. the dataset *quake*).

4 Experimental setup

To optimize the 3 parameters some datasets were used for tuning and were split into 2 folds of equal size. On the first fold of each dataset, all steps of the optimization procedure were done and afterwards the best model was evaluated on the second fold. This is also done vice versa. Hence, the experiments yield two configurations of the same algorithm that only differ in the parametrizations. A test of the parametrizations on the hold-out folds of the tuning datasets is the first step of the evaluation. Additionally some insights are gained by evaluating the two variants also on those datasets that were used during the optimization. To complete the evaluation, the two resulting configurations were also evaluated on some datasets that were not used for any optimization purposes. The aim of the experiments was to optimize the parameters of the algorithm on a set of diverse datasets to capture characteristics of a wide variety of different datasets. Our hope was that by taking a set of datasets that are very different the parameters would be more stable. For this reason, we selected 29 databases in total from the UCI-Repository [Asuncion and Newman, 2007] and from Luis Torgos website⁵. The datasets were divided into 20 sets that were used during the tuning phase and 9 sets that were only used for evaluation purposes. The tuning datasets were

abalone, auto-mpg, auto-price, breast-tumor, compressive, concrete-slump, cpu, deltaailerons, echo-month, forest-fires, housing, machine, pbc, pyrim, quake, sensory, servo, strike, triazines, winequality-white

As mentioned above the main motivation to select these datasets was to capture a lot of different learning problems. Thus, the number of nominal and numerical attributes should be different among the databases and the domains from which they origin should be as diverse as possible.

The 9 datasets that were used to evaluate the algorithms were

auto93, auto-horse, cloud, delta-elevators, meta, r_wpby, stock, veteran, winequality-red

The distribution among the 20 tuning databases in terms of nominal and numerical attributes as well as in terms of size should be approximately the same as in the testing datasets. Therefore both bags of data contain some small, some medium and some big databases. For a detailed overview of the datasets see [Janssen and Fürnkranz, 2010b].

4.1 Evaluation methods

The primary method to evaluate the algorithm was the *rrmse*. The advantage of this evaluation measure clearly lies in its domain-independency. For some of the experiments it would take too much space to include results on every single dataset. In those cases our means for evaluating the different algorithms was to average the results over all datasets. We are aware of the problems that come with averaging results over many different domains (i.e., some databases may be outliers with huge variance compared to the majority of the other datasets) and hence include a Friedman-Test with a post-hoc Nemenyi-Test as suggested in [Demsar, 2006]. The resulting CD-charts give insights how good the algorithms perform by evaluating their ranking independently from using average accuracy.

There are other ways to evaluate regression algorithms domain-independently. The correlation coefficient for instance is also widely used. But there are some drawbacks from using this method regarding rule learning algorithms. For results including the correlation coefficient and a discussion of the drawbacks see [Janssen and Fürnkranz, 2010b].

5 Optimizing parameters

For the split into the 2 folds, all datasets were randomized in advance using the unsupervised randomize function of *weka* [Witten and Frank, 2005]. All evaluation measures were computed using one run of a 10-fold cross validation.

⁵These databases can be downloaded at http: //www.liaad.up.pt/~ltorgo/Regression/ DataSets.html.

Table 1: Results for the splitpoint computation and leftout-parameter (average *rrmse* over the 5 parametrizations of the heuristic)

| parameter (splitpoint) | folds 1 | folds 2 | parameter (left-out) | folds 1 | folds 2 |
|---------------------------|---------|---------|-------------------------|---------|---------|
| 1 | 1.0675 | 1.0540 | 0 | 0.9929 | 1.0209 |
| 3 | 0.9929 | 1.0256 | 0.01 | 0.9787 | 1.0221 |
| 5 | 1.0132 | 1.0261 | 0.02 | 0.9776 | 1.0182 |
| 7 | 1.0067 | 1.0245 | 0.03 | 0.9759 | 1.0156 |
| 9 | 0.9992 | 1.0209 | 0.05 | 0.9739 | 0.9940 |
| 11 | 1.0126 | 1.0427 | 0.1 | 0.9704 | 0.9835 |
| 19 | 1.0163 | 1.0240 | 0.2 | 0.9736 | 0.9701 |

5.1 Optimization of the splitpoint and the left-out parameter

Though these two parameters are likely to have a small deviation in performance among different databases, we decided to optimize them first and fix them before we start optimizing the parameter of the heuristic. We believe that the parameter of the heuristic has a stronger influence on the performance of the algorithm than the other two parameters. This is mostly because the heuristic is used to evaluate every single candidate rule and therefore is the most important factor in the algorithm. Note that the heuristic also has a strong influence on the quality of the rules and on the total number of rules found by the algorithm. The other two parameters are also influencing the performance of the algorithm but rather in an indirect way by assuring to provide splitpoints of good quality and by leaving those examples untouched that are hard to learn.

For this reason, we focussed more on the heuristic parameter than on the other two. If we had concentrated more on these two the performance of the algorithm could have been become a bit better but our main idea was to derive stable parameters for the heuristic and we believe that the gain in performance depends stronger on the heuristic parameter than on the other two (cf. the experimental results in section 5.2).

To start optimizing the splitpoint parameter the other two had to be fixed. In advance it is not known how to determine these values. Thus, the left-out-parameter was fixed to 0, therefore all examples have to be covered. In this case the default rule is built by using the mean of all examples. In other cases where examples remain uncovered it is built by using the mean of all uncovered examples. On the contrary, it is not obvious what parameter value can be used for the heuristic. For this reason, 5 different values were used during the optimization. To make a choice, the two extremes were included ($\alpha = 0$, and $\alpha = 1$), and some values in between, namely 0.4, 0.5, and 0.6. These values were used to include different preferences of the heuristic. Clearly, using only two parameters would be suboptimal because there is some evidence that the optimal parameter rather would lie somewhere in the middle of the domain than at the beginning or the end of it [Janssen and Fürnkranz, 2010a]. We expected the curve yielded by plotting the parameter over the error to be shaped like a U, where the two extreme values would results in a rather bad performance and the optimal value lies somewhere around 0.5 (cf. Section 5.2). To have a combined error measurement for the optimization procedure the mean over the rrmse of these choices was taken.

In the beginning, the *left-out*-parameter was fixed to a value of 0 yielding a starting point for the optimization of the *splitpoint*-parameter. To find the best value some intu-

itive values (1, 3, 5, 7, 9, 11, and 19) were used. All values bigger than 19 were skipped because a clear gain in runtime performance should be achieved. Using huge values would result in practically using all possible splitpoints and thus would not improve the algorithm's runtime⁶.

Table 1 (left table) shows the results for the two optimization procedures (for the two folds of the tuning datasets). As can be seen the best number of splitpoints was 3 on the first folds and 9 on the second folds (the lowest error is depicted in bold in the figure). On the first folds, however, using 9 splitpoints yields the second best *rrmse* which lacks only 0.0063 behind the best performing number of splitpoints. On the second folds, using 9 splitpoints performed best followed by using 19 splitpoints. Using 3 splitpoints lacks 0.0047 in terms of *rrmse* behind the best one and therefore is the fourth best method. Nevertheless, the gap between different parametrizations seems to be bigger on the first folds than on the second ones. Regarding the split of all tuning datasets into 2 folds, these results seem to reflect the randomness in splitting the datasets.

After the optimization of the splitpoint parameter the same procedure was employed to the left-out-parameter of the algorithm. Here, the splitpoints were already fixed to 3 for the algorithm tuned on the first folds and to 9 for the variant tuned on the second folds. To find the best value also some intuitive parameters were used. Thus, the values 0, 0.01, 0.02, 0.03, 0.05, 0.1, and 0.2 were tested during this optimization. The setting where all examples are covered by rules was included to make sure that is more effective to leave some parts of the data uncovered. Clearly, an optimal setting is dataset-dependent. But it also depends on the quality of the induced rules. For numerical target variables it can be useful to cover only those parts of the data that share some common characteristics. For the remainder of the data it could be beneficial to treat them independently from their characteristics.

As can be seen in Table 1 (right table) two different parameters performed best on the two folds. Practically this can be attributed to the same reasons that were already discussed during the optimization of the splitpoint parameter. Thus, on the one hand the randomized split of the data into 2 folds of equal size could have manipulated the characteristics of the datasets. On the other hand it could also be possible that there is no unique best value for leaving examples uncovered. The results also show that leaving examples uncovered is mandatory for the performance of the algorithm.

5.2 Optimization of the heuristics parameter

For the optimization of the heuristics parameter a framework similar to the one introduced in [Janssen and Fürnkranz, 2010a] was used. It employs a binary search to find the best parameter and was proven to yield stable parameters for classification heuristics as shown in [Janssen and Fürnkranz, 2010a].

The search is started with a range of intuitively appealing parameters. Thus, the two extremes of 0 and 1 are tested together with some values in between (0.1, 0.2, ..., 0.9). All settings are evaluated by taking the average of the *rrmse* on the 20 datasets presented in Section 4. Then the best performing parameter is used for further inspection. Therefore, an area around this parameter is inspected in more

⁶Note that the number of disjunct values for an attribute in the data is rather small.



detail. There are several choices to do this, but we decided to evaluate 6 parameters around the best one. Those are distributed equidistantly around the best parameter with decreasing the step size from 0.1 to 0.01. This procedure is executed recursively, so in the next step the 6 parameters around the next best value are tested. The search stops if the *rrmse* improvement falls below a threshold of t = 0.0005. This choice was arbitrary but we believe that the effort that has to be made to narrow down the parameter for the next step of the search procedure is too high compared to the performance gain the next execution may yield.

Figure 2 shows a graphical interpretation of the search for both experiments. For both of them very low parameter settings result in bad performance. When the parameters are increased the performance becomes better as long as the optimal setting is reached. After that it decreases again.

For the parameters that are optimized on the first folds of the datasets (Figure 2 (a)) the curve shows some fluctuations in the part located left of the best parameter. In spite of this behavior the curve depicted in (Figure 2 (b)) is monotonically decreasing in this area. For parameter settings that are bigger than the best parameter the curve in the left figure is now showing a monotone increase whereas it shows more fluctuations when the parameter is tuned on the second folds of the partitioned datasets.

Interestingly, the best parameters are very similar in both experiments. This means that the parameters are stable among different splits of the datasets. On the first folds of these datasets the best parameter lies at 0.59 and on the second folds it was 0.591. For the first folds the parameter 0.591 lacks only 0.007 behind in terms of *rrmse*. For the second folds the difference in performance was 0.001.

Assumed that the best parameter lies somewhere in the region of 0.6, consistency should be preferred over coverage for regression rules. This also holds for classification rules where the preference of consistency is even stronger than in regression. Nevertheless it is an interesting result that the evaluation of a rule's quality follows similar standards in classification and in regression.

6 Results

6.1 Splitpoint processing

Table 2 shows a comparison of the runtime of 2 different splitpoint methods. At first, 3 equidistant splitpoints per attribute were used. Then, 3 clustered splitpoints were employed. Evaluating all splitpoints was too costly⁷. All run-

Table 2: Runtime of different splitpoint methods on the test set

| method | runtime (in sec.) |
|---------------------------|-------------------|
| 3 equidistant splitpoints | 2625.4 |
| 3 clustered splitpoints | 1234.3 |

times depicted in Table 2 are the averages of 10 independent runs on a dual Pentium 4 2.8 GHz processor with 2 GB RAM on the 9 datasets used for testing (cf. Section 4).

As can be seen in Table 2 the clustered splitpoint computation is more efficient than the equidistant method. At first sight this may appear contrary to what could be expected. Due to the much more simpler computation of equidistant splitpoints this method should be faster than the clustering method. But note that this evaluation was done by letting the whole algorithm run on the 9 test datasets. Not surprisingly the quality of the equidistant splitpoints is worse compared to the clustered splitpoints. This results in a significantly higher number of candidate rules that have to be evaluated during the search for the best rule which can be drastically reduced by using clustered splitpoints.

6.2 Comparison with other systems on the tuning datasets

The main focus of the comparison is how well the algorithm performs against other regression algorithms. Table 3 gives an overview of the different algorithms compared to each other on the two folds using the *rrmse*. From now on the tuned algorithm is referred by the name *SeCoReg*.

There are 4 other algorithms that are all implemented in *weka* [Witten and Frank, 2005] which were used to compare our system with. Clearly, some of them are much more complex than our rather simple algorithm⁸. On the other hand most of them employ more complex models, i.e., hyperplanes like the *multilayer perceptron* (MLP) or support vectors like the *SVMReg*. The *linear regression* (Linear Reg.) is also a rather simple algorithm that nevertheless employs quite a good trade-off between runtime and error. *M5Rules* uses rules to explain the data. These rules predict linear models which makes the algorithm much more flexible because each rule is able to map the examples on many different outcome values.

The parameters of all *weka* algorithms were left at default values. The reasons to select these 4 algorithms were that our implementation had to prove that it is comparable

⁷This is due to some huge datasets.

⁸Note that the algorithm neither has a pruning functionality nor an optimization phase.

 Table 3: Results in terms of average *rrmse* for different algorithms on the tuning datasets

| | folds 1 | folds 2 |
|-------------------|---------|---------|
| M5Rules | 0.7425 | 0.8058 |
| Linear Reg. | 0.8145 | 0.9116 |
| MLP | 1.0154 | 1.3890 |
| SVMreg | 0.7917 | 0.8500 |
| SeCoReg (folds 1) | 0.8736 | 0.9291 |
| SeCoReg (folds 2) | 0.8976 | 0.8903 |

Figure 3: Comparison of all algorithms against each other with the Nemenyi test. Groups of algorithms that are not significantly different (at p = 0.05) are connected.



in terms of error to other state-of-the-art systems. Another reason to select these particular algorithms for benchmark was the lack of freely available regression rule learning algorithms. The only free system we found was REGEN-DER and a comparison is given in Section 6.3.

In Table 3 the results of all algorithms on the two folds of the tuning datasets are displayed. Results of both derived *SeCoReg*-algorithms are shown together with their performance on the data sets on which they were tuned (in italics). Not surprisingly both variants of the algorithm that were tuned on the respective folds are better than using them on the left-out folds. The ranking of the algorithms is similar on both experimental variants. The best one was *M5Rules* followed by the *SVMreg* and the *linear regression*. The *SeCoReg* was ranked on the 4th place in both experiments (by average *rrmse*), only slightly behind the *linear regression* (lacking 0.0831 behind on the first folds and 0.0175 on the second folds). The *Multilayer Perceptron* had the worst performance with a rather big gap to the next better algorithm.

Figure 3 shows CD-charts for both experiments. Note that the figure displays ranks averaged on all datasets. Only the algorithm *M5Rules* was significantly better than the *SeCoReg* in both cases.

6.3 Comparison with other algorithms on the test sets

To validate the results on completely different datasets the algorithm was also tested on 9 independent test sets (cf. Section 4). This step is necessary to make sure that the tuning datasets, even though they were split into two disjunct folds, were not overfitted during the parameter tuning phase. Table 4 displays the results in terms of *rrmse* on the test databases for all of the 4 weka algorithms and the two configurations of the *SeCoReg*-learner. The ranking of the

Table 5: Results in terms of *rrmse* compared to *RegENDER* on 7 datasets of the test set

| algorithm | avg. rrmse | avg. rank |
|---------------------------------|------------|-----------|
| SeCoReg (folds 1) | 0.8154 | 3.00 |
| SeCoReg (folds 2) | 0.8538 | 3.13 |
| RegENDER (10 rules) | 0.9008 | 3.88 |
| RegENDER (100 rules) | 0.9291 | 3.88 |
| RegENDER (# rules from folds 1) | 0.9221 | 3.75 |
| RegENDER (# rules from folds 2) | 0.9034 | 3.38 |

algorithms differs slightly compared to the results on the 20 datasets. Hence, on the test sets the *SVMreg* performs best followed by the *M5Rules*-system. On the third place the first *SeCoReg*-learner appears. It was only slightly worse in performance compared to the *M5Rules*-learner. The next best algorithm is the second *SeCoReg*-learner which has achieved a marginal better *rrmse* than the *linear regression*. As in the previous experiments the *multilayer perceptron* was the worst algorithm.

Thus, on the test sets the tuned *SeCoReg*-algorithm achieved better results than in the previous experiment. Here, the best configuration of the algorithm is ranked in third place. Note that the dataset *meta* shows huge standard deviations for some algorithms (*M5Rules, linear regression* and *MLP*). We attribute this to the separation of the data into the 10 folds of the cross validation.

Additionally to the error measurements a Friedman-Test was employed like in the previous Section (cf. Section 6.2). Contrary to the prior results, the Friedman-Test was not rejected at a *p*-value of 0.05 (the critical *F*-value was 2.196 but to reject the test it had to be bigger than 2.492). It would have been rejected at a *p*-level of 0.1, but this was not significant enough to include these results in the paper. For this reason the Nemenyi-Test could also not be done on the test sets. Practically, this means that the *SeCoReg* algorithm does not differ significantly from the 4 weka algorithms at a significance level of 0.05.

Table 5 shows a comparison to *RegENDER* [Dembczyński *et al.*, 2008]. The dataset *auto-horse* contains missing class values which cannot be handled by *RegENDER*. Therefore, this dataset was left out. In addition the results on the dataset *meta* showed strong fluctuations as mentioned before. For this reason this dataset was also left out. *RegENDER* has a parameter to specify the number of rules in the ensemble. To make a choice the algorithm was tested with 10 and 100 rules and with the same number of rules the two *SeCoReg* variants had found on each test set. Clearly, using more rules will result in a lower error (cf. [Dembczyński *et al.*, 2008]) but we think it is fair to run the algorithm with the same number of rules as used in the *SeCoReg*-learner.

The SeCoReg-algorithm was slightly better in average *rrmse* and the average rank was also better. Nevertheless, a Friedman Test was rejected (p = 0.05) but the Nemenyi Test showed that all algorithms were in the same equivalence class (the critical distance extends over all algorithms) and therefore do not differ statistically significant.

To sum up, both tuned variants of the presented algorithm are not able to beat state-of-the-art systems. They are rather situated in the middle of the performance of the other algorithms (this holds at least for the test sets). Especially on the 9 test sets it became clear that the *SeCoReg* rule learners are able to achieve a performance comparable to the results of the 4 *weka* algorithms and *RegENDER*.

Table 4: Results in terms of *rrmse* for different weka algorithms and the SeCoReg-learners on the test set

| dataset | SVMreg | M5Rules | Linear Reg. | MLP | SeCoReg (tuned on folds 1) | SeCoReg (tuned on folds 2) |
|-----------------|-----------------|-----------------|-----------------|-----------------|----------------------------|----------------------------|
| auto-horse | 0.32 ± 0.08 | 0.37 ± 0.14 | 0.32 ± 0.11 | 0.34 ± 0.10 | 0.52 ± 0.18 | 0.61 ± 0.11 |
| auto93 | 0.66 ± 0.12 | 0.58 ± 0.19 | 0.67 ± 0.20 | 0.57 ± 0.19 | 0.65 ± 0.17 | 0.85 ± 0.29 |
| cloud | 0.39 ± 0.12 | 0.42 ± 0.16 | 0.40 ± 0.13 | 0.62 ± 0.33 | 0.61 ± 0.19 | 0.67 ± 0.15 |
| delta-elevators | 0.61 ± 0.01 | 0.60 ± 0.01 | 0.61 ± 0.01 | 0.63 ± 0.01 | 0.78 ± 0.03 | 0.77 ± 0.03 |
| meta | 0.92 ± 0.08 | 1.86 ± 1.58 | 2.33 ± 1.72 | 1.40 ± 0.90 | 1.00 ± 0.02 | 1.01 ± 0.03 |
| r_wpbc | 1.03 ± 0.16 | 1.14 ± 0.19 | 1.04 ± 0.13 | 2.20 ± 0.56 | 1.35 ± 0.20 | 1.27 ± 0.18 |
| stock | 0.37 ± 0.05 | 0.14 ± 0.03 | 0.36 ± 0.04 | 0.20 ± 0.04 | 0.25 ± 0.03 | 0.26 ± 0.04 |
| veteran | 0.93 ± 0.15 | 1.23 ± 0.61 | 1.07 ± 0.36 | 3.01 ± 1.78 | 1.09 ± 0.22 | 1.21 ± 0.33 |
| winequality-red | 0.82 ± 0.03 | 0.81 ± 0.03 | 0.81 ± 0.03 | 0.95 ± 0.08 | 0.98 ± 0.09 | 0.95 ± 0.04 |
| averages | 0.6739 | 0.7942 | 0.8456 | 1.1017 | 0.8040 | 0.8438 |

Due to the rather simple design of the *SeCoReg*-algorithm these results seem to be promising.

7 Conclusion and further work

In this paper a new rule learning algorithm for the task of regression was presented. It was shown that the algorithm performs comparable to different state-of-the-art algorithms implemented in *weka* and *RegENDER*, a rather new algorithm.

A new splitpoint generation method was introduced. This method proved to support the quality of candidate rules and even results in lower runtime compared to naive methods like the generation of equidistant splitpoints. Nevertheless, the number of generated candidate rules directly depends on the number of splitpoints. But as shown in the experiments at least for one configuration of the algorithm a number of 3 splitpoints per numerical attribute was enough.

A novel rule learning heuristic was introduced that clearly improves the algorithms performance due to its flexibility in weighting the error of a rule with its coverage. An optimal setting for this regression rule heuristic was presented and it proved to be stable since the parameter values are nearly the same. An interesting observation is that, as known from classification, in regression the rules consistency also should be preferred over its coverage.

A promising path to optimize the algorithm would be to adapt the advantages of algorithms like *M5Rules* which predicts linear models in the head of each rule. On the one hand the performance of the algorithm should be drastically improved when using linear models instead of single target values. On the other hand, much of the interpretability of the rule set would be lost when doing so.

Acknowledgments

This research was supported by the *German Science Foundation* (*DFG*) under grant no. FU 580/2-2.

References

- [Asuncion and Newman, 2007] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.
- [Clark and Niblett, 1989] Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.
- [Cohen, 1995] William W. Cohen. Fast effective rule induction. In *ICML*, pages 115–123, 1995.
- [Dembczyński et al., 2008] Krzysztof Dembczyński, Wojciech Kotłowski, and Roman Słowiński. Solving regression by learning an ensemble of decision rules. In *ICAISC '08*, pages 533– 544, Berlin, Heidelberg, 2008. Springer-Verlag.
- [Demsar, 2006] Janez Demsar. Statistical comparisons of classifiers over multiple datasets. *Machine Learning Research*, 7:1– 30, 2006.

- [Demšar, 1999] D. Demšar. Obravnavanje numericnih problemov z induktivnim logicnim programiranjem. Master's thesis, Faculty of Computer and Information Science, University of Ljubljana, Slovenia, 1999. In Slovene.
- [Friedman and Popescu, 2008] Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *Annals Of Applied Statistics*, 2:916, 2008.
- [Fürnkranz and Flach, 2005] Johannes Fürnkranz and Peter Flach. ROC 'n' rule learning – Towards a better understanding of covering algorithms. *Machine Learning*, 58(1):39–77, 2005.
- [Fürnkranz, 1999] Johannes Fürnkranz. Separate-and-conquer rule learning. Artificial Intelligence Review, 13(1):3–54, February 1999.
- [Holmes et al., 1999] Geoffrey Holmes, Mark Hall, and Eibe Frank. Generating rule sets from model trees. In *Twelfth Australian Joint Conference on Artificial Intelligence*, pages 1–12. Springer, 1999.
- [Janssen and Fürnkranz, 2010a] Frederik Janssen and Johannes Fürnkranz. On the quest for optimal rule learning heuristics. *Machine Learning*, 78(3):343–379, March 2010. DOI 10.1007/s10994-009-5162-2.
- [Janssen and Fürnkranz, 2010b] Frederik Janssen and Johannes Fürnkranz. Separate-and-conquer regression. Technical Report TUD-KE-2010-01, TU Darmstadt, Knowledge Engineering Group, 2010.
- [Karalič and Bratko, 1997] Aram Karalič and Ivan Bratko. First order regression. *Machine Learning*, 26(2-3):147–176, 1997.
- [Quinlan, 1992] Ross J. Quinlan. Learning with continuous classes. In 5th Australian Joint Conference on Artificial Intelligence, pages 343–348, Singapore, 1992. World Scientific.
- [Torgo and Gama, 1996] Lus Torgo and Joo Gama. Regression by classification. In *In Proceedings of SBIA96, Borges*, pages 51–60. Springer-Verlag, 1996.
- [Torgo, 1995] Luis Torgo. Data fitting with rule-based regression. In In Proceedings of the 2nd international workshop on Artificial Intelligence Techniques (AIT'95, 1995.
- [Ženko et al., 2005] Bernard Ženko, Saso Džeroski, and Jan Struyf. Learning predictive clustering rules. In In 4th Intl Workshop on Knowledge Discovery in Inductive Databases: Revised Selected and Invited Papers, volume 3933 of LNCS, pages 234–250. Springer, 2005.
- [Wang and Witten, 1997] Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes. In *Poster papers of the 9th European Conference on Machine Learning*. Springer, 1997.
- [Weiss and Indurkhya, 1995] Sholom M. Weiss and Nitin Indurkhya. Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research*, 3:383– 403, 1995.
- [Witten and Frank, 2005] Ian H. Witten and Eibe Frank. *Data Mining* — *Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers, 2nd edition, 2005.