Attribute Value Selection Considering the Minimum Description Length Approach and Feature Granularity

Kemal Ince[1] and Frank Klawonn[2]

[1] Volkswagen AG, Komponenten-Werkzeugbau
Gifhornerstr. 180, 38037 Braunschweig, Germany
kemal.ince@volkswagen.de
http://www.volkswagen-braunschweig.de/
[2] Data Analysis and Pattern Recognition Lab
Ostfalia University of Applied Sciences
Salzdahlumer Str. 46/48, 38302 Wolfenbüttel, Germany
f.klawonn@ostfalia.de,
http://public.ostfalia.de/~klawonn/

Abstract. *In this paper we introduce a new approach to automatic attribute and granularity selection for building optimum regression trees. The method is based on the minimum description length principle (MDL) and aspects of granular computing. The approach is verified by giving an example using a data set which is extracted and preprocessed from an operational information system of the Components Toolshop of Volkswagen AG.*

Key words: Minimum Description Length, Granular Computing, Regression Tree, Decision Support, Intelligent Decision Algorithm

1 Introduction

The ideas presented in this paper are motivated by an application in the Components Toolshop of Volkswagen AG in Brunswick. This business area is responsible for producing tools in other divisions of Volkswagen AG for the the serial production. The Components Toolshop has approximately 700 members of staff and includes a 30.000m^2 production area so that it can be considered as one of the biggest tool shops in the world. The product range includes forming tools, (like gearbox cases and engine boxes), injection moulds, casting moulds and production lines and other machined tools.
These tools are denoted in this paper as products. In the Components Toolshop a very large data set is available describing the different processes of manufacturing the products. This data set is mainly obtained from operational information systems. A subset of this data set contains the production time of the products. Every product contains an allocated time δ_s and an actual time δ_i which can

differ from each other. The subset contains additional information on the manufacturing process which is used later in the analysis phase.
This paper describes how a regression tree is built to predict the relative deviation between these two time values. By building the model the regression tree must fulfill the following two criteria as good as possible:

- The predicted deviation should deviate as little from the true deviation **and**
- the complexity of the constructed regression tree Δ_{K} should be as small as possible.

Another aspect which must be considered is that the input values have different granularities. An example is the feature *component* which specifies the automotive part, the feature *component assembly* in which the *components* are aggregated and the feature *component category* in which the *component assemblies* are combined.
It is obvious that identified rules containing features with fine granularity are less general than rules which are composed of features with coarse granularity. The developed algorithm has to decide in favour of the feature which delivers the best result for both criteria described above.
The paper is organised as follows. Section 2 provides a brief overview on the basics of regression trees and the minimum description length principle. In Section 3, the motivation and discussion of the approach is presented in detail. Section 4 describes how the generated model is evaluated and Section 5 concludes with a discussion of the results and an outlook on future work.

2 Theoretical Background of Regression Tree and MDL

This section provides a brief introduction to regression trees and the minimum description length principle. Further details can be found in [1, 2].

2.1 The Regression Tree Idea

Regression is besides classification one of the most important problems in predictive statistics [1]. It deals with predicting values of a continuous variable from one or more continuous and/or categorical predictor variables [3]. In general the regression tree method allows input values to be a mixture of numerical and nominal values. The output value has to be numerical. The result of this approach is that a tree is generated where each decision node contains a test on some input values. The terminal nodes of the tree contain the predicted output values [4]. In [5] an example is given how to build a regression tree using the program XMLMiner with an example data set.

The CART algorithm is an example for building classification and regression trees. This algorithm was developed by Leo Breiman in 1984. An important property of this algorithm is that it delivers only binary trees. This means every

node of the tree is either a terminal node or followed exactly by two successor nodes [6].
The basic regression tree growing algorithm which is used in the different approaches works in the following way:

1. The starting point is the initial node which contains the whole data set. Here, the values m_c, the regression value for the node and the error S are calculated as defined below.
2. If all the points in the node have the same value for all the independent variables, stop the algorithm. Otherwise, search over all binary splits of all variables for the one which will reduce S as much as possible. If the largest decrease in S is less than some threshold δ, or one of the resulting nodes would contain less than q data objects, stop the algorithm. Otherwise, take that split and create two new nodes.
3. Go back to step 1, in each new node.

In the above described algorithm S is the sum of squared errors for the regression tree R_T measured as follow:

$$S = \sum_{c \in leaves(R_T)} \sum_{i \in c} (y_i - m_c)^2 \tag{1}$$

where $m_c = \frac{1}{n_c} \sum_{i \in c} y_i$ is the prediction for leaf c [7].

2.2 The Minimum Description Length Principle

The minimum description length principle (MDL) is based on the fundamental idea that any regularity in a data set can be used to compress it [2]. Compression means to describe the data set with fewer symbols than the number of symbols which are needed to describe the data set literally. Such a data set can for example be described by a decision tree which has fewer symbols as the initial data set. The more regularities in the data set exist, the more the data set can be compressed. Folowing this idea, it is possible to understand 'learning' as 'finding regularities' in the data.
Therefore the MDL principle can be used in different ways for inductive inference such as to choose a model that trades-off the goodness-of-fit on the observed data set with the complexity of the model (in statistical questions) or in a predictive interpretation where MDL methods can be used to search for a model with good predictive performance on unseen data sets [2].
In the following example, the idea is illustrated that learning can be interpreted as data compression. In the sample below a 2000 bits long sequence S_1 is shown, where just the beginning and the end of it is listed.

$$'01110011100111001110.....01110011100111001110' \tag{2}$$

It seems that S_1 is a 400-fold repetition of '01110'. A decription method which maps descriptions $\overline{D}$ in a unique manner to a data set D is needed to compress

S_1. A programming language can be used as description method to carry out the compression of S_1. In the sample below such a computer program is displayed in the programming language $C\#$. It describes the regularity in S_1 and is shorter than S_1 itself.
Example of a computer program in C# describing the regularity in sequence S_1

```
string sequence = "";
for (int i = 1; i <= 400; i++)
{
    sequence = sequence + "01110";
}
Console.WriteLine("The sequence = " + sequence);
```

The example above is very theoretical, since in practical applications such highly compressible data seldom exist. Usually sequences with lower compressibility such as described in the sample below are given.

$$'00110000001100100001.....11001000000100110010000' \tag{3}$$

The sequence S_2 has a recognizable regularity because it contains approximately twice as many 0's as 1's. But the regularity S_2 is more of statistical than of deterministic character. So it seems possible to find a description which is able to generate future sequences that is similar to S_2.
If we consider that n is the length of the sequence (in both samples above is $n = 2000$ bits long), S_1 can be compressed to $O(\log\ n)$ and S_2 can be compressed to αn with $0 < \alpha < 1$.
This fact allows to make the following case:

$$\exists\ \overline{D}(s)\ where\ n(\overline{D}(s)) \leq n(s) \quad with\ s \subseteq S \tag{4}$$

where S is the initial sequence, s is a subsequence of S, $\overline{D}(s)$ is the description of the subsequence s, $n(\overline{D}(s))$ is the length of $\overline{D}(s)$ and $n(s)$ is the length of s.

3 Motivation and Solution

As mentioned initially, in the Components Toolshop of Volkswagen AG different operational information systems are in use, which are required to support the manufacturing process. This means several of these systems are concerned with the manufacturing process directly and some of them, for example organizationally attached, indirectly. The application of these systems delivers large amounts of data which can contain interesting and hidden coherences. It is suspected that the type of specific events depends on various facts and could not be detected by a manual inspection of the large data set. The cycle time of a product could, for example, depend on the milling machine which is used to manufacture it. Due to this problem data mining approaches were used to detect these coherences in

the data.
The considered application area has to deal with data containing divers information like the machine on which the product was manufactured, by whom the product was processed, to which greater category the product belongs etc. These features are the input values for the decision model which has to be generated. Furthermore, information about the real time which was needed to manufacture the product δ_i and the expected time δ_s which is estimated initially by the planner are available. The ratio Δ_r of these two features constitutes the output value, which has to be predicted.

$$\Delta_r = \frac{\delta_i}{\delta_s} \tag{5}$$

Both time values are numerical. Therefore the output value has a continuous character. Table 1 shows a fictitious data set with the following features:

- The *machine* which was planned to be used during manufacturing process: M_s,
- the *machine* which was used in the real manufacturing process: M_i,
- the *machine category* of M_i: MC_i,
- the *component* which has to be manufactured: C,
- the *component assembly* of C: C_a,
- the *component category* of C: C_c and
- the *output value*, the ratio between the two time values: Δ_r

A row (data object) in the data set to be analysed is characterised by the above value and the data set might look as in Table 1. Of course, the real data sets are much larger.

M_s	M_i	MC_i	C	C_a	C_c	Δ_r
a	b	B	10201	1020	10	0.76
a	a	A	10202	1020	10	0.74
a	b	B	10301	1030	10	0.75
c	d	D	20301	2030	20	0.44
a	c	C	20302	2030	20	0.46

Table 1. A fictitious data set.

We need an algortihm to build a regression tree which predicts Δ_r. During building the regression tree, it is necessary to decide which granularity for the input values makes sense to predict the output value as good as possible. The simpler the constructed model (regression tree) is and the smaller the errors it delivers in predicting the output value the better it is. In the following section we describe our approach of the implemented algorithm in more detail in the form of pseudo-code.

Pseudo-code of the combined MDL and RegTree algorithm

```
program RegtreeMDL (Output)
  var
     mean; outputValue: double;
     meanList: Dictionary<string, double>;
     ds: DataSet;
     dt1, dt2: DataTable;
     dcc: DataColumnCollection;
     rows: DataRow[];
     colName, colValue, filter: string;
  begin
     ds = GetInitialData();
     for int i = 0 to ds.rows.Count
     step
        outputValue = outputValue +
          ds.rows[i][ds.IndexOf(lastColumn)];
     next
     mean = outputValue / ds.rows.Count;
     meanList.Add("wholeDataSet", mean);
     dcc = ds.dt1.Columns;
     for int i = 1 to dcc.Count - 1
     step
        colName = dcc[i].ColumnName;
        dt2 = SelectDistinct("tbl_AttValue",ds.dt1,colName);
        for int j = 0 to dt.rows.Count
        step
           colValue = dt.rows[j]-ItemArray[0];
           filter = colName + " LIKE " + colValue;
           rows = ds.dt1.Select(filter);
           for int k = 0 to rows.Count
           step
              outputValue = outputValue +
                rows[k].ItemArray[rows[k].ItemArray.Length - 1];
           next
           mean = outputValue / rows.Count;
           meanList.Add(colName + " _ " + colValue , mean);
        next
     next
     for int i = 0 to meanList.Count - 1
     step
        if SumOfFailureRegtree(meanList[i]) <=
           SumOfFailureRegtree(meanList[i+1])
        then break;
        else
```

```
    next
end
```

The above pseudo-code describes the functionality of the implemented algorithm. In first step, the *mean* of the initial data set is calculated. First the numbers of *rows* in the data table *dt*1 is identified. For each *row* in *rows* the sum of the $outputValue$ is computed. Finally the *mean* of the numeric output value is calculated by using the sum of the output values and dividing it by the number of *rows*. Afterwards, in the second step all other possibilities for splitting are calculated. Therefore, all the $columnNames$ of the input values have to be considered. The codomain of the input values are identified by using these $columnNames$. In the following step, these $columnNames$ and codomains were used to $filter$ the data set and calculate the means for the subset. The last loop of the algorithm deals with the method $SumOfFailureRegtree()$ to calculate, in which step the algorithm delivers the 'best' result and has to terminate.

4 Validation with generated data

In this section, a brief example is given, how the algorithm handles the data and which results it delivers. Therefore, the input value A with the value set $A = \{a_1, a_2\}$ is defined. The output value Z consists of continuous values. In this fictitious case we define the target value Z as follows:

$$Z = \left\{ +\frac{\Delta}{2} + R(0,1), -\frac{\Delta}{2} + R(0,1) \right\}$$

with $R(0,1)$ constituting a minimal random noise between 0 and 1 and Δ is a constant. Furthermore it is suggested that both values of A occur equally often in the whole data set. In Table 2 such a data set is displayed.

A	Z
a_1	$+\frac{\Delta}{2} + R(0,1)$
a_2	$-\frac{\Delta}{2} + R(0,1)$
a_1	$+\frac{\Delta}{2} + R(0,1)$
a_2	$-\frac{\Delta}{2} + R(0,1)$
a_2	$-\frac{\Delta}{2} + R(0,1)$
a_1	$+\frac{\Delta}{2} + R(0,1)$

Table 2. Abstract data set containing random noise.

A regression tree for this simple data set can have the following two forms displayed in Figures 1 and 2. The first tree delivers the arithmetic mean $\overline{x} = 0$. The error F_1 in predicting Z is

$$F_1 = \frac{\Delta}{2} + R_1(0,1) + \frac{\Delta}{2} + R_2(0,1) + \frac{\Delta}{2} + R_3(0,1) + \frac{\Delta}{2} + R_4(0,1) + \frac{\Delta}{2} + R_5(0,1) + \frac{\Delta}{2} + R_6(0,1)$$

$$= 3 * \Delta + \sum_{i=1}^{6} R_i(0,1)$$

Provided that the random noise $R_i(0,1)$ is close to zero, the error F_1 becomes

$$F_1 = 3 * \Delta$$

The length of the regression tree L_1 is 0. It has only one predicting value 0 which is generaly valid for the whole data set. The MDL-value M_1 consists of the sum of F_1 and L_1. The result is that the MDL-value of the first tree is

$$M_1 = F_1 + L_1 = 3 * \Delta.$$

Fig. 1. Regression tree with one node including the whole data set and predicting Z as $\overline{x} = 0$.

The second regression tree delivers a model which separates the data set into two subsets. The error in predicting Z in this case is

$$F_2 = R_1(0,1) + R_2(0,1) + R_3(0,1) + R_4(0,1) + R_5(0,1) + R_6(0,1)$$

$$= \sum_{i=1}^{6} R_i(0,1)$$

Provided like above that the statistical noise $R_i(0,1)$ (almost) zero, the error becomes $F_2 = 0$. To get the MDL-value the length of the regression tree L_2 is needed to be calculated.

$$\left|\frac{\Delta}{2}\right| + \left|\frac{-\Delta}{2}\right| = \Delta$$

The MDL-value of the second tree is calculated as followed.

$$M_2 = F_2 + L_2 = \Delta.$$

Because of minimizing the MDL-value a split like described by the second regression tree below makes sense.

The above considerations are only valid when the noise is small compared to the constant Δ. When the noise becomes larger, the more complex decision tree might not be favoured anymore. Let $\frac{\Delta}{2}$ be the predicted value of the model and assume $\Delta = 0.002$ and therefore $\frac{\Delta}{2} = 0.001$

In binary code this means

$$(0.002)_2 = 0.0000000010000011$$

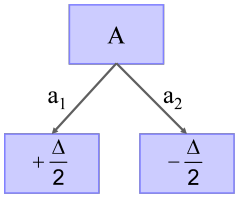

Fig. 2. Regression tree with one root node and two leafs.

and

$$(0.001)_2 = 0.0000000001000001$$

If the noise has the value $N = 0.123$

$$N = 0.123 \rightarrow (0.123)_2 = 0.0001111101111100$$

$$0.001 + 0.123 = \underline{0.124}$$

The sum of both binary values is calculated in binary coding as follows:

$$\begin{array}{r} 0.0000000001000001 \\ + \underline{0.0001111101111100} \\ 0.0001111110111101 \end{array}$$

The error is much higher than the difference between the prediction of the Δ. This means every bit in the predicted value has to be corrected so that the deviation in the predicted Δ is not relevant any more.

5 Conclusion

Decision support is getting more and more important even for industrial application areas such as the Components Toolshop. It has a wide range of topics where the field of building models by generating decision and regression trees is a less but itself established detail. If only the aspect of minimizing the predicting error is considered, the model might exhibit a very high complexity. Additionally considering the fact to reduce the complexity of the model delivers a result, which can be used to answer universally valid questions in decision support. With the described approach the possibility to select automatically the 'best' input values by predicting a continuous output value is accomplished. This approach can be adapted in different analysis problems to resolve decision support problems.

References

1. Wei-Yin, Loh: Classification and Regression Tree Methods. In: Encyclopedia of Statistics in Quality and Reliability, pp. 315–323. Wiley-VCH, (2008)

2. Grünwald, P.: A Tutorial Introduction to the Minimum Description Length Principle.Centrum voor Wiskunde en Informatica, Netherlands.
3. Overview Classification and Regression Trees, http://www.statsoft.com/textbook/classification-and-regression-trees/
4. Online Help of XMLMiner, http://www.resample.com/xlminer/help/Index.htm
5. Example of a Regression Tree, http://www.resample.com/xlminer/help/rtree/rtree_.htm
6. Wikipedia The Free Encyclopedia, http://de.wikipedia.org/wiki/CART_(Algorithmus)
7. Shalizi, C.: Classification and Regression Trees. In: 36-350 Data Mining, Lecture 10. (2009)