Universiteit Leiden
The Netherlands

Opleiding Informatica

Numeric subgroup discovery by MDL-optimal histograms

Tim Opdam

Supervisors:
Matthijs van Leeuwen & Lincen Yang

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

17/07/2022

Abstract

Subgroup discovery is a data mining task that looks for interpretable subgroups in a dataset. An example of a subgroup discovery algorithm is the SSD++ algorithm. SSD++ is an subgroup discovery algorithm for both numeric and nominal targets that generates a rule list. This algorithm works with the minimal description length (MDL) principle and uses Gaussian statistics for describing the data. This means that the algorithm assumes that the samples in each subgroup are normally distributed in order to score how good the subgroup is, however when the actual distribution is not Gaussian, describing the subgroup using Gaussian statistics is not optimal. Therefore we propose HistRule, a subgroup discovery algorithm based on SSD++. Instead of using Gaussian statistics to describe the data like SSD++, HistRule uses MDL-optimal histograms, which can fit the data no matter what the actual distribution is.
In this thesis we will be comparing the effectiveness of HistRule to SSD++ and several other regression measures by measuring the mean squared errors of their predictions on several datasets, and looking at the average rule lengths and complexities of the interpretable models. These experiments have shown that on average HistRule has a lower mean squared error than SSD++, meaning that it functions better as a predictor, however this is not a major difference. We have also seen that HistRule takes significantly longer the learn the model than SSD++ meaning it is not suitable for larger datasets as it is now.

Contents

1 Introduction

Subgroup discovery[Kl6][Atz15] is a data mining technique that is used for finding interesting subsets of a dataset that stand out from the rest of the target values in the data. Subgroup discovery is possible for targets of both numeric and nominal values, but we will be focussing only on numeric subgroup discovery.
In this thesis we are going to work with subgroup discovery by means of rule lists. A rule list is an interpretable predictive model that is built from a series of `IF` statements with at the end the `ELSE` statement that is the default rule describing every sample not covered by any rule.
An example of a rule list can be seen in Figure 1. For each rule in this figure the rule itself, the usage (i.e., how many samples are covered by each rule), the mean value of the samples in each subgroup, and the standard deviation of the samples in each subgroup are reported. Each rule (`IF` statement) forms its own subgroup.

Rule	Usage	Mean	Std
IF 143.0 <= thalach < 162.0 AND oldpeak >= 0.8 AND restecg = left_ventricular_hypertrophy AND sex = female	10	350.5	109.4
ELSE IF thalach < 125.0 AND 1.0 <= num < 2.0	5	271.6	4.2
ELSE IF 120.0 <= trestbps < 130.0 AND thal = reversal_defect	20	239.4	34.9
ELSE	202	242.5	46.4

Figure 1: Example of a rule list predicting cholesterol level generated by HistRule

An algorithm used for generating such rule lists is the SSD++ algorithm[PGBvL21a][PGBvL21b]. The SSD++ algorithm is a subgroup discovery algorithm that works with the MDL principle [Ris78][Gr7][GR19] and uses Gaussian statistics for describing the data. However when the data is not normally distributed, describing it with Gaussian statistics is not optimal.
Therefore we introduce HistRule, a subgroup discovery algorithm based on the SSD++ algorithm, but instead of using Gaussian statistics for generating the rules, it uses MDL-optimal histograms[KM07] that are not bound by any assumption of how the data is distributed and can thus better describe a wider range of data distributions than Gaussian statistics.
To see if HistRule performs better than the SSD++ algorithm it is based on, we will be comparing the mean squared errors for predicting the values of samples of several datasets alongside several other regression methods.
Contributions. In this thesis we propose Histrule. For the creation of the HistRule algorithm we have replaced the Gaussian statistics from the SSD++ algorithm with MDL-optimal histograms and a Python implementation of HistRule was made. A Python implementation of an algorithm for generating MDL-optimal histograms was also made from an existing version in R.
Overview. Section 2 elaborates on some terms used in this thesis and the algorithms to which we will compare HistRule. Most of these algorithms will be regression algorithms since we will be comparing how well HistRule can predict numeric values. Section 3 talks in more detail about how the SSD++ algorithm works and how it learns each rule, this section also talks about what MDL-optimal histograms are. Section 4 is about the problem statement and Section 5 is about how the SSD++ algorithm was modified to achieve HistRule. It also talks about how it was implemented in Python. And finally in Section 6 we will compare the predictive performance of Histrule with SSD++ and several other regression models.

2 Related Work

Several regression models have been chosen to compare HistRule to. In the experiments section (Section 6) we compare the predictive performances of these algorithms with Histrule. This section will briefly go over each chosen regression algorithm and their main differences, followed by an elaboration on some terms used in this thesis.

SSD++. The SSD++ algorithm[PGBvL21a] [PGBvL21b] generates a rule list for nominal and numeric targets. A rule list is an interpretable model that splits the data into subsets with each rule. It uses a Gaussian model for generating each rule, unlike HistRule which uses MDL-optimal histograms. See Section 3.1 for more details on how the SSD++ algorithm generates a rule list.
CART. Classification and Regression Trees (CART)[BFOS84], are interpretable tree-based models. For the regression trees the values of the leaf nodes are the average values of the samples in those nodes.
M5. The M5 algorithm[Qui92] is also a tree-based regression model but different from CART it has linear regression functions as leaf nodes.
RuleFit. Rulefit[FP08] is a rule-based model that is learned through an ensemble of regression trees which then fits a sparse linear model to the rule ensemble.
Random Forest regression. Random Forest[Bre01] is a model for classification and regression that is built from an ensemble of decision trees. For regression tasks random forest outputs the average value from each tree in the ensemble as the predicted value. Unlike a decision tree, a random forest model is not interpretable.
Gradient Boosting regression. Gradient boosting[Fri01] is an ensemble model for both classification and regression. Like random forest, a gradient boosted trees model is built from an ensemble of decision trees, but the main difference from random forest is how the tree ensemble is generated. In gradient boosting each tree is built after each other, one tree at a time, to improve the overall model. Random forest builds each tree independently in parallel.

2.1 Subgroup Discovery

Subgroup discovery[Kl6][Atz15] is a data mining technique that is used for finding interesting subsets of a dataset that stand out from the rest of the target values in the data. It aims to describe these subsets of the data in an interpretable way.
Subgroup discovery has a wide range of applications, it can be used for predictive tasks such as classification and regression, or it can be used in exploring the data.
Herrera, Carmona, González and del Jesus[HCGDJ11] show a more in depth overview of subgroup discovery and its applications.

2.2 MDL

The MDL principle[Ris78][Gr7][GR19] is a model selection principle that states that the model that compresses the data the most, is the best model. This means that the more the data is compressed, the more is learned of the data, and the better it can be predicted.
The MDL score is calculated in two parts, the encoding the model, and the encoding of the data

using the model. The formal definition of the two-part description length is:

$$MDL = L(data|model) + L(model) \tag{1}$$

Here $L(model)$ is the code length of the model in bits, and $L(data|model)$ the code length of the data, given the model, in bits.

2.3 Histograms

A histogram is a way to visualize the distribution of the data. The data is distributed over 'bins'. A bin is a small range over the data in which all samples of the data that have a value in that range, fall. In a histogram the entire range of the data is covered by a number of non overlapping bins. Usually bins in a histogram have the same width as each other, but it is also possible for histograms with bins of variable width. Figure 2 shows an example of a histogram with 10 bins of equal widths.

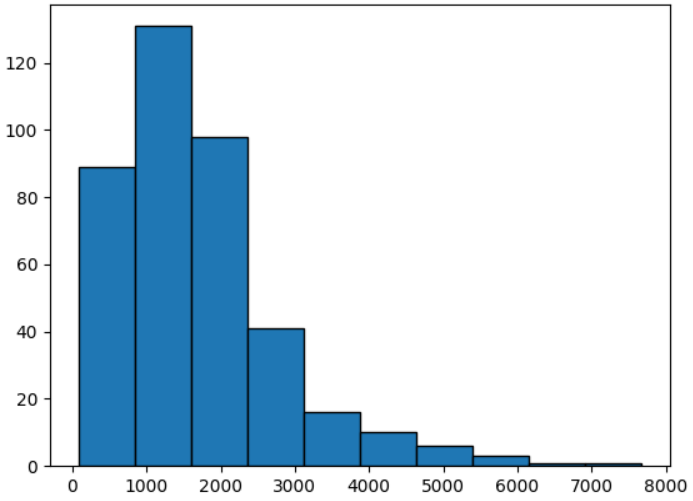

Figure 2: Example of a histogram

3 Preliminaries

A rule list is a series of Boolean expressions that splits the data into several subsets. A single rule can have multiple Boolean expressions that are joined together with AND statements. A general example of a rule with multiple conditions is: IF a AND b THEN $P_1(y)$, where a and b are Boolean expressions (e.g. $x_1 < 1$ AND $x_2 == male$) and $P_1(y)$ is some distribution of the data in the subset. A rule list can have any number of rules, with at the end of the list the default rule (the ELSE statement). The default rule covers all the samples that are not already covered by an earlier rule. So a generalised rule list of $n-1$ rules is:

IF $condition_1$ THEN $P_1(y)$
ELSE IF $condition_2$ THEN $P_2(y)$
...
ELSE IF $condition_{n-1}$ THEN $P_{n-1}(y)$
ELSE $P_n(y)$

3.1 SSD++

SSD++ evaluates candidate rules based on the minimal description length (MDL) principle. The MDL principle states that the model that compresses the data the most, and thus have the shortest code length, is the best model. So the best rule, has the shortest code length when encoded.
The MDL score is based on two parts: encoding the data, and encoding the model. For encoding the data SSD++ uses Shannon-Fano encoding. Shannon-Fano encoding is a data compression method that uses probabilities to encode the data. These probabilities can be gained from the assumption that the data in each subgroup is normally distributed. For the encoding of the model in bits, see the SSD++ papers[PGBvL21a][PGBvL21b] for more details as this will not be discussed in this thesis.
In the SSD++ algorithm rules are iteratively formed in a seperate-and-conquer way, one rule at a time. This means that after adding each rule, the samples covered by this rule are removed from the dataset (seperate), and the following rule will be based on the remaining uncovered samples (conquer). To form a rule with the SSD++ algorithm, candidates rules are learned by a beam search on the uncovered samples. The rule found by the beam search that decreases the code length the most, gets added. The beam search stops when there is no new rule that would decrease the code length.

3.2 MDL-optimal Histograms

MDL-optimal histograms are histograms with variable bin width and allow different densities, meaning that not every bin has to be the same size and not every bin has to have a similar number of samples, making them very flexible. The cut points, which are where the boundaries for each bin are, are chosen based on the MDL principle. Figure 3 shows an example of an MDL-optimal histogram, where the bins have varied widths and densities together with the actual distribution.

In the figure the we can see that the data is not normally distributed but the histogram can fit the distribution well. The histogram has wider bins where the data does not change much, and thinner bins where the value changes quickly, allowing it accurately describe the data.

For generating one-dimensional, MDL-optimal histograms we will be using the algorithm by Yang, Baratchi and van Leeuwen[YBvL20], which is an improved version of the algorithm by Kontkanen and Myllymäki[KM07].

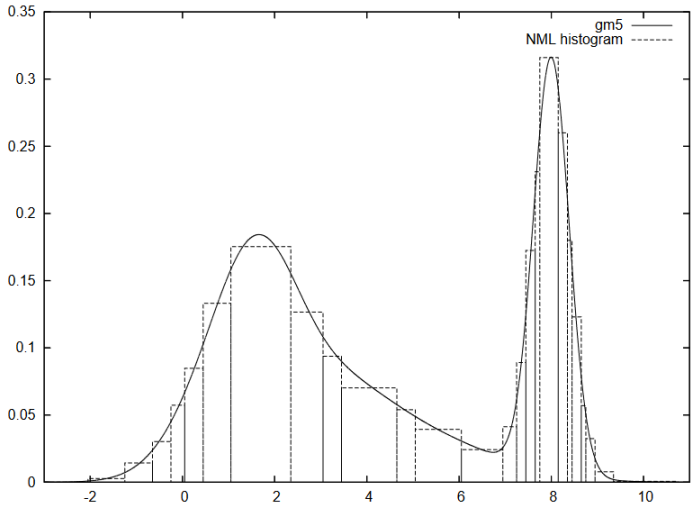

Figure 3: A one-dimensional MDL-optimal histogram by Kontkanen and Myllymäki[KM07]

4 Problem Statement

In this thesis we want to make a rule list with a histogram model instead of a rule list with a Gaussian model. We will again work with the MDL principle and thus want to encode the data and the model to get the MDL score. Like SSD++, the data points are encoded with Shannon-Fano encoding, to get these probabilities a model is necessary. SSD++ used a Gaussian model, but if the distribution in a subgroup is not normally distributed, the quality of the evaluation will be harmed. This is why in this thesis we will use a histogram model to see if we can select better rule lists using a histogram model instead of the Gaussian model used by SSD++. Since a rule list that compresses the data more, is a better rule list, we want to see if using histogram models will be able to compress the data more, and thus, create a better rule list. We will test this if it is better by comparing the predictive performance of HistRule, since a better model should have a better predictive performance

5 Method

In order to compare SSD++ with HistRule, the SSD++ algorithm is used as the base for HistRule with only the Gaussian model changed to a histogram model. For the histogram model used in HistRule we have chosen MDL-optimal histograms that can compress data better than regular histograms.
MDL-optimal histograms are far more flexible and require no assumptions about the distribution of the data, meaning that they will fit the data that is not normally distributed much better. If the histograms can fit the data better, it can also compress the data more, and following the MDL principle: the model that compresses the data the most is the best model.
To encode the data, Shannon-Fano coding is used, where the probabilities are gained from histograms instead of Gaussian statistics. To calculate the probability of a data point, given a histogram, you need the probability that the data point falls in the bin with the corresponding value in it. You also

need the probability that it is that specific value, given that it did fall in that bin. The probability that the data point x is the value a would then be:
$P(x = a) = P(x$ in corresponding bin$) * P(x = a|x$ in corresponding bin$)$. The probability for the entire dataset would then be $P(x_1, x_2, ..., x_n) = P(x_1) * P(x_2) * ... * P(x_n)$. For the final code length there is also a penalty term introduced called the *regret*, for more details on the code length calculation see the papers for MDL-optimal histograms [KM07][YBvL20].
The goal is to replace the code length calculation for encoding the data of each subgroup of the SSD++ algorithm to the code length calculation of MDL optimal histograms. For a fair comparison the encoding of the model will remain the same.

5.1 Implementation

The implementation of HistRule is based on the Python implementation of the SSD++ algorithm[1] and the R implementation of the MDL histograms by Yang, Baratchi and van Leeuwen [YBvL20]. The source code for the MDL-optimal histograms were gained by request. The functions used to calculate the code length in the Python implementation are `length_rule_free_gaussian` and `length_rule_fixed_gaussian`, and the code length calculation for the MDL histograms in R is the function `mdlhist1d`.
Figure 4 shows a flowchart of the program, note that this flowchart does not show all function calls but only the functions and created classes related to the code length calculation of the SSD++ algorithm. In the flowchart `length_rule_free_gaussian` and `length_rule_fixed_gaussian` are highlighted in green.

As seen in the flowchart the functions are called in a few different places in the program, `length_rule_fixed_gaussian` is called by `compute_default_length` in the class `GaussianRuleList`. `compute_default_length` calculates the code length of the default rule, that is, the samples that are not covered by any rule.
Both functions are called by `compute_delta_data` which is used to calculate the score of a subgroup proposed by the beam search, and both are called by `add_rule` which adds the rule to the rule list and calculates the code length of the entire rule list so far.
In those three places, the code length calculation has been replaced by the `mdlhist1d` function.

5.2 R version

The first version made use of the R implementation of the MDL-optimal histograms. Because the algorithm for MDL histograms has only been implemented in R and the SSD++ algorithm only in Python, they must somehow be integrated. For this, the rpy2 library for Python was used as a bridge between the languages to call the `mdlhist1d` function that was implemented in R. For convenience a wrapper function was used instead that set all the required arguments to their proper values with more ease. The arguments in question are `left_bound`, `right_bound`, `num_candidate_cuts`.
For the `left_bound` the value given to it is the minimal value of the data, minus 1% of the range of the data (here range refers to max(data) - min(data)).
For the `right_bound` it is the maximum value of the data, plus 1% of the range of the data.

[1] https://github.com/HMProenca/RuleList

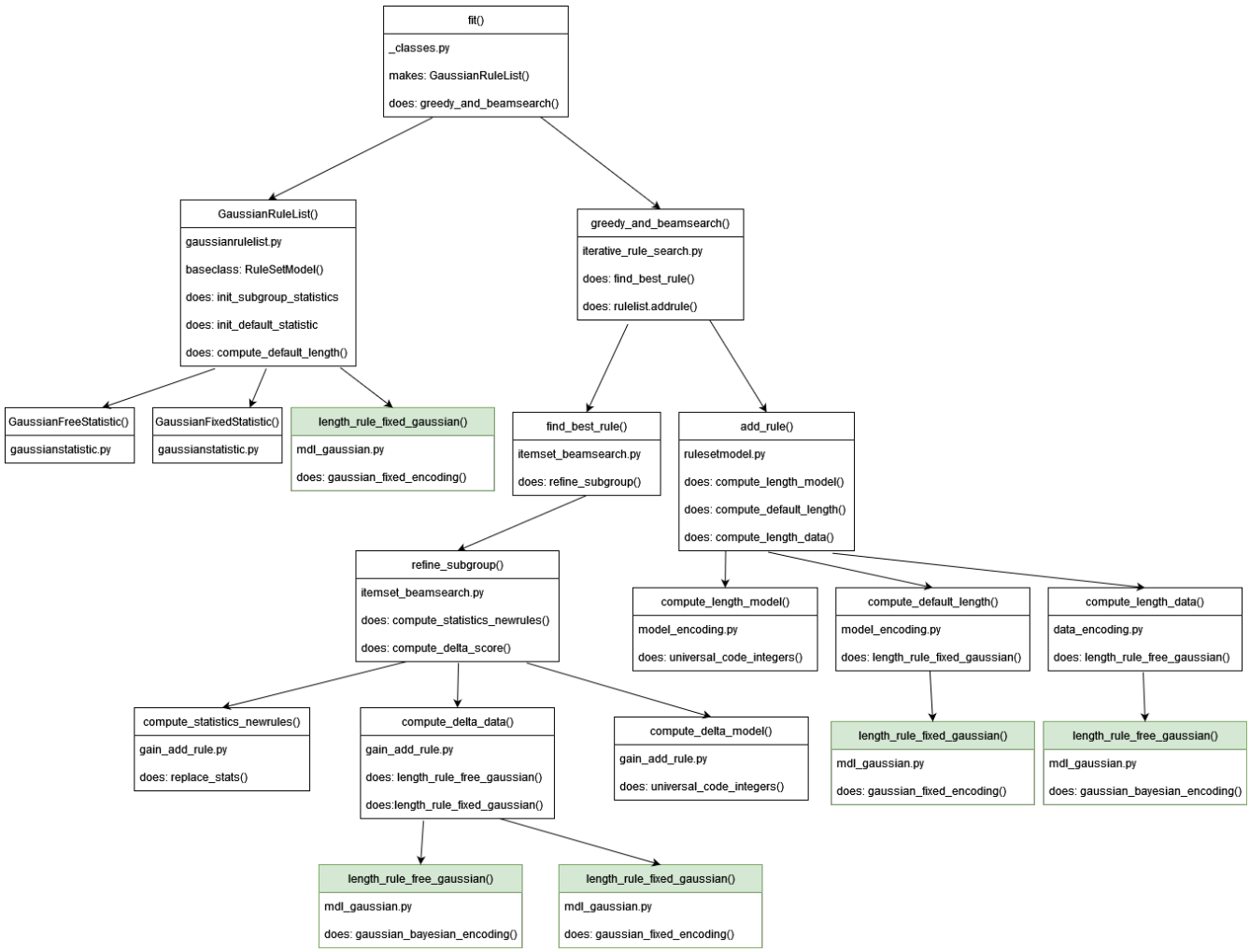

Figure 4: Flowchart of the Python implementation of SSD++ algorithm

`num_candidate_cuts` is assigned the number of unique values in the data.

Figure 5 shows a comparison between the runtimes of the `mdlhist1d` function when called via the Rscript command on the command line and when called via the rpy2 library in Python. For this experiment the parameters for the `mdlhist1d` function were: `Kmax = 50`, `max_cut_search = 1000` and `cl_model = True`, the data was the array `[0, 1, 2, ..., 99]`. The data only has 100 samples due to the fact that only smaller datasets were used during the experiments (see Section 6) allowing a better insight into expected runtimes than a larger testing data. The results from R were gained through an R script that called the `mdlhist1d` function in a for loop, and the rpy2 results were gained by calling the `mdlhist1d` function in a for loop with use of the rpy2 library in Python. It is immediately clear that there is a significant difference in runtime, where calling via rpy2 is significantly slower than in a normal R environment. A significant portion of the time spent running `mdlhist1d` was spent linking packages and compiling the C++ code the function uses with every call, however since all the calls use the same R environment it is only necessary to link and compile once in the first call and not in subsequent calls. Therefore a global variable was introduced to

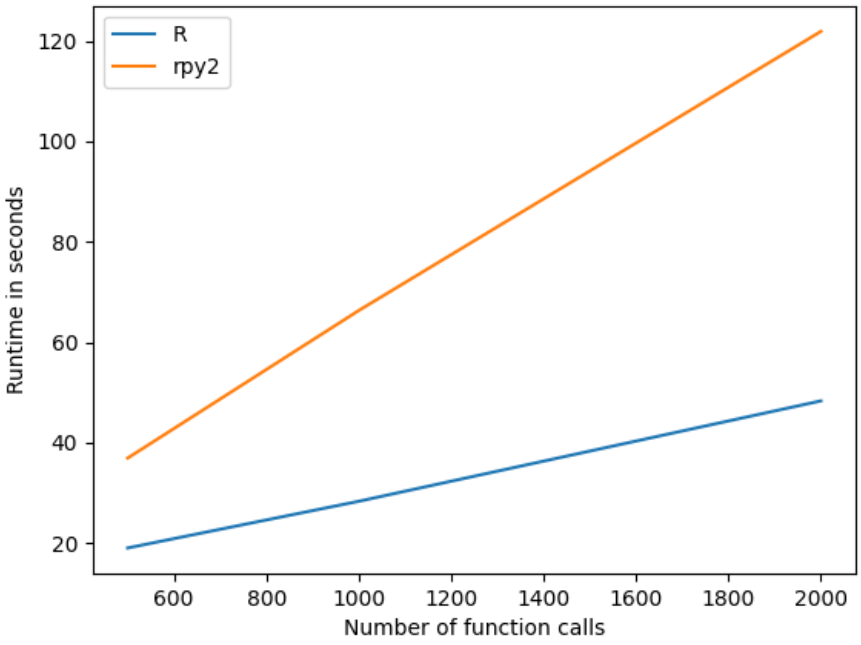

Figure 5: Comparison of runtimes of between rpy2 and Rscript

make sure that only in the first run the packages are linked and the C++ code gets compiled. See Figure 6 for the runtimes of the modified code.
The difference in runtime is now much smaller, and the runtime of the rpy2 version no longer grows much faster than the R version. The figure also shows the runtime of the Python implementation.

5.3 Python version

In order to try to further speed up the code, a Python implementation of the `mdlhist1d` function was made in order to reduce the overhead caused by rpy2. Because the expensive part of the original R program was done in C++, if the C++ code were to be reused in the Python implementation the runtime would not be negatively impacted. To include the required C++ code into the Python implementation pybind11 was used.
The original code in R made use of the SCCI package in order to calculate the regret, however there is no Python implementation of the SCCI package. To solve this, from the C++ source code of the SCCI package[2], the functions necessary to calculate the regret were used together with pybind11 in order to calculate the regret in the Python implementation. Figure 6 shows a comparison between the Python implementation and the improved R implementations.
Considering that the first run of the R implementation takes on average around 8 seconds where it links all packages and compiles the C++ code, the Python implementation performs only slightly slower than the original R code and better than the rpy2 implementation, but without taking 8 seconds on the first run and can thus be considered an improvement.

[2] https://github.com/cran/SCCI

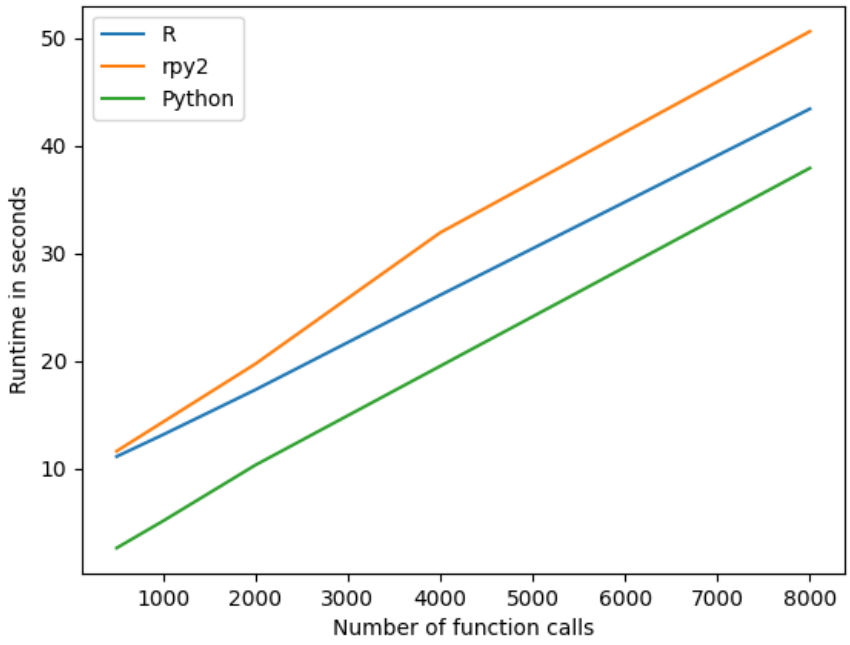

Figure 6: Comparison of runtimes of the modified function between rpy2 and Rscript and the Python implementation

6 Experiments

To test the effectiveness of HistRule, it will be compared with the following models:

- SSD++,
- CART,
- M5,
- RuleFit,
- RandomForestRegressor,
- GradientBoostingRegressor,

In the experiments the predictive power will be compared because a better rule list should compress the data better, leading to better predictive performance, and we want to see if HistRule creates a better model than SSD++. The main focus will be on comparing to the SSD++ algorithm, but several other regression models are also included for comparison.

6.1 Experiment setup

The datasets used for testing are the same as in the SSD++ paper and were sourced from their testing Github repository[3]. A description of the datasets can be found in Table 1, this table shows how many samples these datasets have, how many categorical features and how many numeric features the chosen datasets have. Due to the prohibitive execution time of HistRule on the default settings (see Table 2), only the smaller datasets have been used.

Dataset	Samples	Categorical	Numeric
cholesterol	297	7	5
autoMPG8	392	0	6
dee	365	0	6
ele-1	495	0	2
forestFires	517	0	12
concrete	1030	0	8
abalone	4177	0	8

Table 1: Description of benchmark datasets

For the experiment each dataset was split into a training set 80% of the size of the full dataset 10 times, resulting in 10 different training sets, and 10 corresponding testing sets consisting of the remaining 20% of the data. Each model was trained on the same 10 training sets for each dataset. For each algorithm the mean squared error (MSE), given by $MSE = \frac{1}{n}\sum(y_{actual} - y_{predicted})^2$, is measured on the test sets to test its predictive power, as well as its execution time.

The training sets for this experiment were generated with the `train_test_split` function from sklearn, for the 10 training sets the random states were: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
The settings for SSD++ are the default settings, for a fair comparison HistRule is also run with the same settings for these experiments. Additionally the settings for generating the MDL-optimal histograms are: `Kmax = 50`, `max_cut_search = 100`, `cl_model = false`. These are lower settings than in the small experiment in Section 5 to try to speed the code up.
For CART the `DecisionTreeRegressor` from sklearn was used with its default settings except for `ccp_alpha`, which performs post-pruning on the generated tree. The value for `ccp_alpha` was chosen by means of cross validation on the training set. For this, the training set was further split into another training and testing set, on which the values for `ccp_alpha` were tested. The value that had the highest score on the testing set, was chosen. For more details, see the sklearn tutorial for post pruning decision trees[4].
M5 was run with `rpart` in R.
The Python implementation of RuleFit[5] was used with its default settings.
For the RandomForestRegressor the sklearn implementation, `RandomForestRegressor`, was used with its default settings.

[3] https://github.com/HMProenca/SSDpp-numeric
[4] https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html
[5] https://github.com/christophM/rulefit

For the GradientBoostingRegressor the sklearn implementation, `GradientBoostingRegressor`, was used with its default settings.

6.2 Results

Table 2 shows the average runtime of HistRule and SSD++ for each dataset in seconds, Table 3 shows the average MSE of each model with the 10 different training sets for each dataset, and table 4 shows the standard deviation of these MSEs.

Dataset	**HistRule**	**SSD++**
cholesterol	961.2	87.7
autoMPG8	814.8	52.5
dee	1716.9	22.2
ele-1	83.6	2.6
forestFires	8164.8	360.0
concrete	9037.0	300.7
abalone	2231.4	1400.7

Table 2: Average execution time in seconds

Dataset	**HistRule**	**SSD++**	**CART**	**M5**	**RuleFit**	**GradientBoosting**	**RandomForest**
cholesterol	3287.72	3286.13	4081.33	1868.91	3394.80	3445.52	3144.34
autoMPG8	12.26	12.35	13.14	9.07	7.70	7.95	7.40
dee	0.64	0.89	0.26	0.16	0.16	0.16	0.16
ele-1	458790.85	481793.78	536584.54	376128.84	419324.91	444438.20	431864.94
forestFires	6520.97	7147.07	4447.27	4683.64	3900.69	3781.25	3555.58
concrete	72.19	74.86	50.66	67.59	19.89	27.01	24.53
abalone	6.09	6.31	5.69	6.17	4.63	4.71	4.83

Table 3: Average test MSE of each algorithm per dataset

Dataset	**HistRule**	**SSD++**	**CART**	**M5**	**RuleFit**	**GradientBoosting**	**RandomForest**
cholesterol	959.34	871.43	2609.52	288.90	1027.00	988.32	833.81
autoMPG8	2.95	2.51	2.64	0.72	2.04	2.04	2.21
dee	0.09	0.21	0.05	0.01	0.02	0.02	0.018
ele-1	83542.49	109727.78	154830.41	36534.18	114420.56	148035.80	110138.24
forestFires	4007.54	4239.30	5816.26	1199.28	5528.72	4807.02	4787.90
concrete	5.58	3.95	7.36	7.55	2.73	3.73	3.32
abalone	0.57	0.65	0.41	0.25	0.34	0.31	0.29

Table 4: Standard deviation of MSE for each algorithm per dataset

From Table 3 it is seen that HistRule performs similarly to the SSD++ algorithm, and is for nearly all of these datasets slightly better than SSD++. Also the standard deviations of HistRule and

SSD++ are similar. However HistRule is also significantly slower than SSD++, as seen in Table 2. The table also shows that like SSD++, HistRule does not outperform the other regression models.

Another aspect to compare is the average number of rules and the complexity of these rules, for this comparison only the interpretable models, so HistRule, SSD++ and CART, are compared. Table 5 shows the average number of rules on the same 10 training sets as in the previous experiment, Table 6 shows the average length of the rules. The average rule length of both HistRule and SSD++ is the average number of conditions in the rules, for the CART model the number of rules are the number of leaf nodes and the average rule length the average depth of these leaf nodes.

Dataset	**HistRule**	**SSD++**	**CART**
cholesterol	5.8	0.8	8.3
autoMPG8	13.8	6.7	31.6
dee	13.8	2.9	16.2
ele-1	15.1	7.0	13.7
forestFires	28.3	17.6	5.7
concrete	43.1	16.4	245.5
abalone	43.8	24.1	25.9

Table 5: Average number of rules

Dataset	**HistRule**	**SSD++**	**CART**
cholesterol	2.1	2.4	3.23
autoMPG8	2.15	2.11	5.17
dee	1.8	1.59	3.93
ele-1	1.74	1.91	3.8
forestFires	3.01	3.48	2.68
concrete	2.94	3.29	9.02
abalone	2.77	2.61	5.09

Table 6: Average length of rules

As seen in Table 5, HistRule has a much greater number of rules than SSD++ but on average they have a similar rule length that is shorter than the average rule length of CART.

An example of the top four rules generated by SSD++ on the autoMPG8 dataset can be seen in Figure 7. Figure 8 shows the top four rules on the same dataset, with the same training set, generated by HistRule.

Rule	Usage	Mean	Std
IF Cylinders >= 318.0 AND Acceleration < 78.0	55	13.8	1.6
ELSE IF Horse_power < 2451.0 AND Acceleration >= 78.0	46	35.3	4.3
ELSE IF Cylinders >= 250.0 AND Acceleration < 80.0	51	16.8	2.6
ELSE IF Horse_power >= 2870.0 AND Acceleration < 80.0 AND Displacement >= 85.0	35	19.6	1.7

Figure 7: Top four rules on the autoMPG8 dataset by SSD++

Rule	Usage	Mean	Std
IF Horse_power >= 4055.0 AND Acceleration < 74.0	33	12.8	1.5
ELSE IF Horse_power >= 3415.0 AND 72.0 <= Acceleration < 74.0	7	13.6	1.3
ELSE IF Horse_power >= 4055.0	20	15.5	1.3
ELSE IF Horse_power < 2451.0 AND Acceleration >= 78.0 AND Displacement >= 85.0	45	35.6	4.0

Figure 8: Top four rules on the autoMPG8 dataset by HistRule

From these rules we can see that HistRule generates smaller rules than SSD++. Because the rules are smaller, HistRule generated rule lists also have more rules. This can also be seen in Table 5 where HistRule has a noticeably larger average number of rules than SSD++.

For a deeper look into these rules, the data distributions of the samples in each rule are plotted. In Figure 10 are the distributions of the top four rules by SSD++, and in Figure 11 are the distributions of the top four rules by HistRule. Figure 9 shows the distribution of the entire autoMPG8 dataset. In these figures it can be seen that both SSD++ and HistRule form rules where the samples in that rule distributed similarly to a normal distribution.

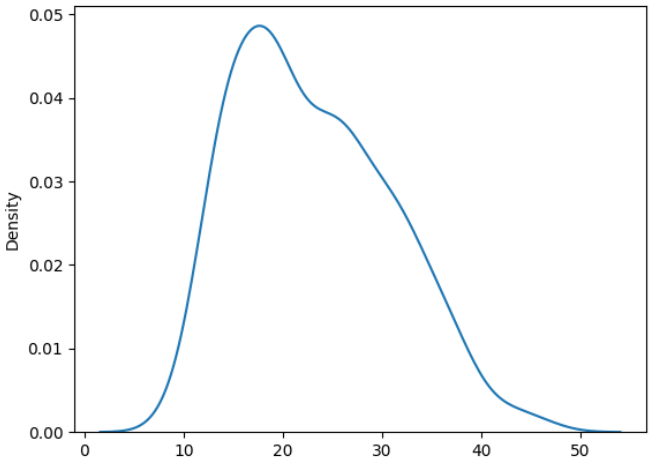

Figure 9: Distribution of target values of the autoMPG8 dataset

7 Conclusions and Further Research

In this thesis we have proposed HistRule, a rule list based subgroup discovery algorithm for numeric targets that was based on the SSD++ algorithm. The SSD++ algorithm uses a Gaussian model to describe the data, and thus assumes that the data in each subgroup is normally distributed. In contrast, HistRule uses MDL-optimal histograms instead. By removing the need for a Gaussian assumption in the SSD++ algorithm, and in its place using MDL-optimal histograms we have created the HistRule algorithm. Histrule should, due the adaptive nature of MDL-optimal histograms, be far more flexible for describing the data and would thus no longer need any assumption of how the data is distributed. This allows HistRule to compress non normally distributed data more, resulting in a better model according to the MDL principle.
In the experiments we have shown that it indeed performs slightly better as a predictor than the original SSD++ algorithm. We have done this by comparing the mean squared errors for their predictions, however it is only a slight difference, and it also takes significantly longer to learn the rule list which is less than ideal.

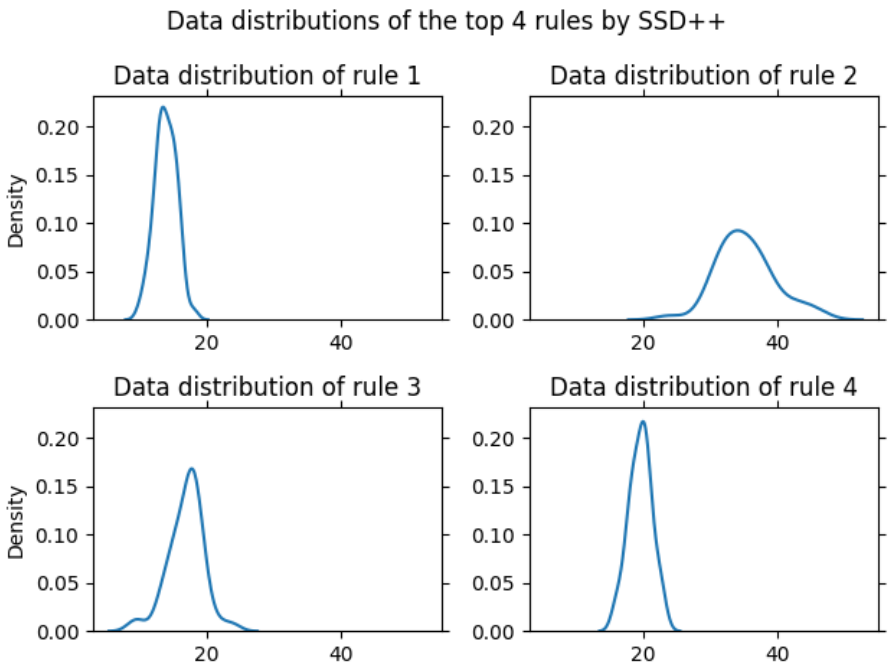

Figure 10: Data distributions of the top four rules by SSD++

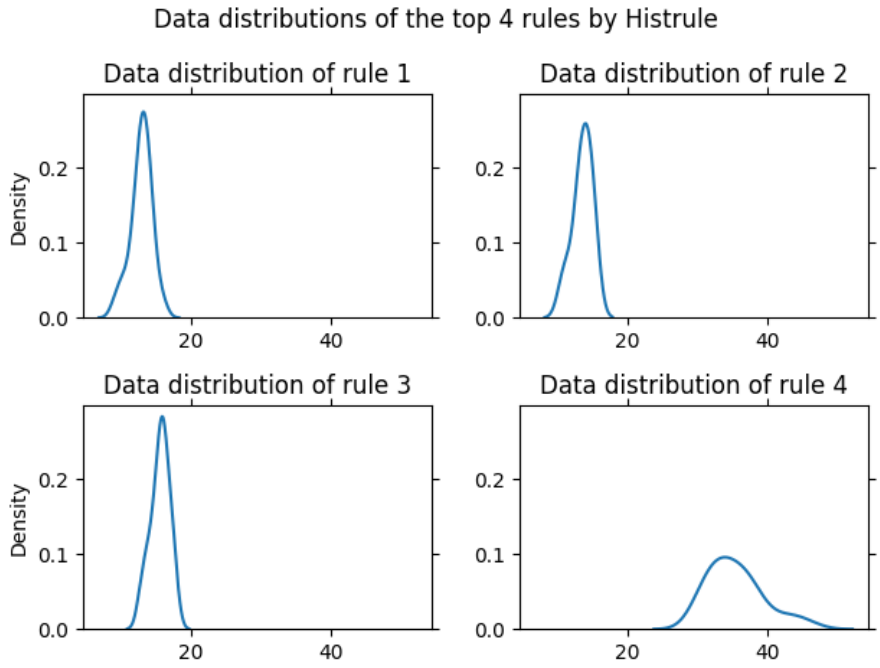

Figure 11: Data distributions of the top four rules by HistRule

7.1 Future Work

Due to the significantly slower runtime, the HistRule algorithm is not viable for larger datasets as it is now with the default settings, and would thus need to be sped up.

Making a full C or C++ implementation of the algorithm for calculating the MDL-optimal histograms, is a possible suggestion for speeding up HistRule, since most of the execution time of HistRule is in calculating these histograms.
A second option to speed up the code would be to fine tune the settings of HistRule, since it currently has the same default settings as SSD++. These new settings should try to lower the runtime without impacting the performance too much, an obvious candidate here would be lowering the `beam_width` for the beam search, which as its default value has 100, thus limiting the number of candidate subgroups that need to be evaluated.
A more in-depth optimization for the parameters of the algorithm for generating the MDL-optimal histograms is also an option, with as goal trying to find a good balance between speed and performance.

References

[Atz15] Martin Atzmueller. Subgroup discovery. *WIREs Data Mining and Knowledge Discovery*, 5(1):35–49, 2015.

[BFOS84] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. Classification and regression trees. *Journal of the American Statistical Association*, 81, 01 1984.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[FP08] Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3), sep 2008.

[Fri01] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001.

[GR19] Peter Grünwald and Teemu Roos. Minimum description length revisited. *International Journal of Mathematics for Industry*, 11(01):1930001, dec 2019.

[Gr7] Peter D Grünwald. *The minimum description length principle*. MIT press, 2007.

[HCGDJ11] Francisco Herrera, Cristóbal J. Carmona, Pedro González, and María José Del Jesus. An overview on subgroup discovery: Foundations and applications. *Knowledge and Information Systems*, 29:495–525, 12 2011.

[Kl6] Willi Klösgen. Explora: A multipattern and multistrategy discovery assistant. In *Advances in Knowledge Discovery and Data Mining*, pages 249–271. American Association for Artificial Intelligence, 1996.

[KM07] Petri Kontkanen and Petri Myllymäki. Mdl histogram density estimation. In Marina Meila and Xiaotong Shen, editors, *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pages 219–226, San Juan, Puerto Rico, 21–24 Mar 2007. PMLR.

[PGBvL21a] Hugo M. Proença, Peter Grünwald, Thomas Bäck, and Matthijs van Leeuwen. Discovering outstanding subgroup lists for numeric targets using MDL. In *Machine Learning and Knowledge Discovery in Databases*, pages 19–35. Springer International Publishing, 2021.

[PGBvL21b] Hugo Manuel Proença, Peter Grünwald, Thomas Bäck, and Matthijs van Leeuwen. Robust subgroup discovery, https://arxiv.org/abs/2103.13686, 2021.

[Qui92] J. R. Quinlan. Learning with continuous classes. pages 343–348. World Scientific, 1992.

[Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[YBvL20] Lincen Yang, Mitra Baratchi, and Matthijs van Leeuwen. Unsupervised discretization by two-dimensional mdl-based histogram, https://arxiv.org/abs/2006.01893, 2020.