Supplementary Information on: Interpretation of complex pain-related phenotype clusters using explainable artificial intelligence (XAI)

Jörn Lötsch[1, 2] and Sebastian Malkusch[1]

[1]Institute of Clinical Pharmacology, Goethe - University, Theodor-Stern-Kai 7, 60590 Frankfurt am Main, Germany
[2]Fraunhofer Institute for Molecular Biology and Applied Ecology IME, Project Group Translational Medicine and Pharmacology TMP, Theodor-Stern-Kai 7, 60590 Frankfurt am Main, Germany

Contents

Chapter 1

Explainable artificial intelligence

The operations performed by artificial intelligence (AI) during classification are often complex. Therefore, the procedure of AI-based decision making remain a black-box. The aim of explainable artificial intelligence is the traceability of the mechanisms that underly the procedure of AI-based classification. The difficulty in designing an explainable artificial intelligence (XAI) is finding a compromise between intelligibility and performance. Unfortunately, high classification accuracies are often achieved using complex AI models such as random forests and deep artificial neural networks which decisions are hard to understand. Here we focus on simpler AI models based on a single decision tree to make their process of classification understandable by generating a set of classification rules that reproduce their decisions

Chapter 2

Metrics

Tree learning is mainly a top-down approach. It is realized by identifying the observation parameter that best splits the set of observations. However, the definition of the split quality can be defined in multiple ways based upon the metrics used for tree learning. The most prominent metrics are introduced in the following.

2.1 Gini impurity

The Gini Impurity is defined as the probability of choosing an element in the dataset by random times the probability of falsely classifying this element based on the class distribution in the dataset.

$$G(p) = \sum_{i=1}^{\hat{C}} p_i \sum_{k \neq i} p_k \tag{2.1}$$

With $\hat{C}$ being the number of unique values within the class attribute C and p_i the fraction of elements labeled with class i within the training data. The probability of falsely labeling the chosen element is calculated as

$$\sum_{k \neq i} p_k = 1 - p_i \tag{2.2}$$

which leads to a Gini impurity of

$$G(p) = \sum_{i=1}^{\hat{C}} p_i \left(1 - p_i\right) \tag{2.3}$$

For the case that $\sum_{i=1}^{\hat{C}} p_i = 1$ the Gini impurity simplifies to:

$$G(p) = 1 - \sum_{i=1}^{\hat{C}} p_{i,k}^2 \tag{2.4}$$

2.2 Entropy

By definition entropy is a measure of disorder. It calculates to

$$H(S) = \sum_{i=1}^{\hat{C}} -p_i \log_2 p_i \tag{2.5}$$

Here, p_i defines the probability of drawing an instance with the classification value i from the data set S.

2.2.1 Information

The information content I of a data subset after splitting by attribute α is defined as the weighted average Entropy of all features.

$$I(S, \alpha) = \sum_i p\left(i \mid \alpha\right) H\left(S \mid \alpha\right) \tag{2.6}$$

Here, the weighting factor $p\left(i \mid \alpha\right)$ is defined as probability of randomly drawing an instance with a class value i from the data subset resulting from the split α. Alternatively, it can be rewritten using the weighted size:

$$p\left(i \mid \alpha\right) = \frac{\hat{S}_i\left(\alpha\right)}{\hat{S}\left(\alpha\right)} \tag{2.7}$$

Here, $\hat{S}_i(\alpha)$ defines the number of instances from the data set S after a split by attribute α that are classified with a class value i and $\hat{S}(\alpha)$ defines the total number of instances in the data set S after a split by attribute α .

2.2.2 Information gain

In tree-based machine learning the information gain IG, also termed Kullback-Leibler divergence, defines the amount of information that is gained about the system after splitting

the observations by a specific observation parameter. It is therefore also termed mutual information. It is defined as the difference in information entropy from the state before the split to the weighted sum of the conditional information entropy of all children after the split by attribute α.

$$IG\left(S,\alpha\right) = H\left(S\right) - \sum_{v\in\alpha} p(v) H\left(S \mid \alpha\right) \tag{2.8}$$

$$= \sum_{i=1}^{\hat{C}} -p_i \log_2 p_i - \sum_{v\in\alpha} p(v) \sum_{j=1}^{\hat{C}} -p\left(j \mid \alpha\right) \log_2 p\left(j \mid \alpha\right) \tag{2.9}$$

2.2.3 Intrinsic Attribute Information

The intrinsic attribute information holds the share of an attribute on a splitting procedure. For the splitting attributeα it calculates to:

$$\mathrm{Int}I\left(S,\alpha\right) = -\sum_{v\in\alpha} \frac{\mid S_{\alpha=v} \mid}{\mid S \mid} \log_2 \left(\frac{\mid S_{\alpha=v} \mid}{\mid S \mid}\right) \tag{2.10}$$

Attributes that are characterized by high intrinsic information consist of partitions that are evenly distributed in size. Attributes that are characterized by low intrinsic information, on the other hand, have a heterogeneous distribution of partitions, within which a few partitions contain the most tuples. Therefore, the proportion of attributes with higher intrinsic information about the classification is generally low.

2.2.4 Information Gain Ratio

The information gain ratio defines the ratio of the information gain to the intrinsic information.

$$IGR = \frac{IG\left(S,\alpha\right)}{\mathrm{Int}I\left(S,\alpha\right)} \tag{2.11}$$

It reduces the bias against multi-value attributes by taking into account the number and size of the branches when selecting an attribute. As the intrinsic attribute information can be zero the information gain ration may not always be defined.

2.3 Misclassification error

With p being the frequencies of k observation parameters at node t are defined as:

$$P_t = (p_{1,t}, ..., p_{k,t}) \tag{2.12}$$

so that

$$\sum_{i=1}^{k} p_{i,t} = 1 \tag{2.13}$$

the misclassification error (MCE) is defined as

$$M = 1 - p_{\hat{k}(t),t} \tag{2.14}$$

with

$$\hat{k}(t) = \text{argmax}\,(P_t) \tag{2.15}$$

2.4 Metrics comparison

The introduced metrics shall be compared upon the example of a binary problem with p being the frequency of class label 1 and $1-p$ being the frequency of class label 2. For the example the metrics calculate to:

- misclassification error:

$$M = 1 - \text{argmax}\,(p, 1-p)$$

- Entorpy:

$$H(S) = -p \log_2 p - ((1-p) \log_2 (1-p))$$

- Gini impurity

$$\begin{aligned} G(p) &= p\,(1-p) + (1-p)\,(1-(1-p)) \\ &= 2p\,(1-p) \end{aligned}$$

Compared to the misclassification error, the entropy and the Gini impurity are differentiable. Therefore, both entropy and the Gini impurity lead to similar trees. However, Gini impurity is faster to compute, while entropy based trees tend to be slightly more balanced.

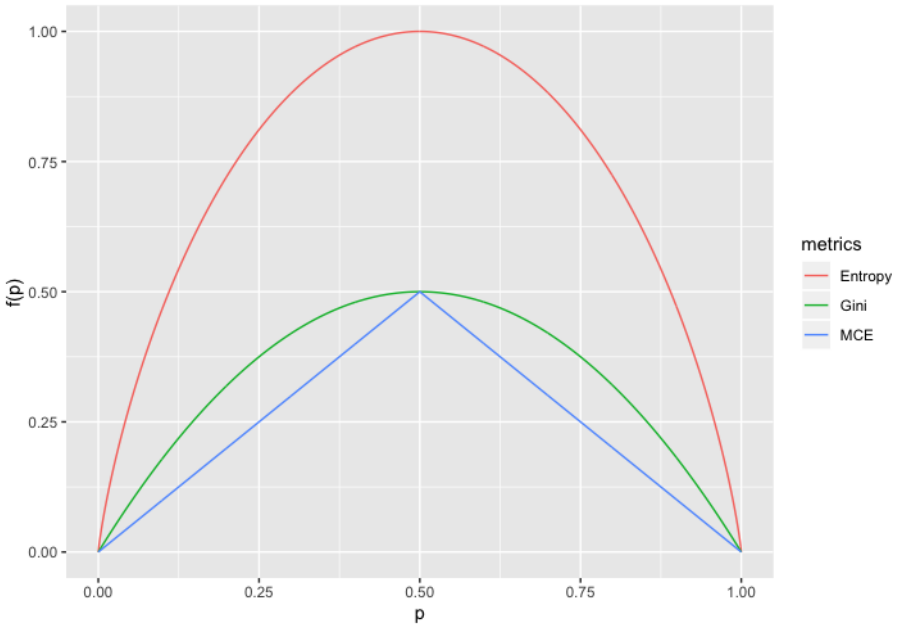

Figure 2.1: Metrics comparison for the case of a binary split, with p being the frequency of class one. Red: Entropy, Green: Gini impurity, Blue: MCE

Chapter 3

Pruning

A main problem that arises with decision tree based classification is the determination of the right tree size for a distinct classification problem. If the decision tree grows too large it will overfit the data. On the other hand, if growing ist stopped too early the tree will lack nodes that are necessary for high accuracy classification. A phenomenon termed underfitting of the data. A second disadvantage of large trees is the fact, that they become hard to interpret by humans which will directly reduce their value for XAI. This problem can be handled by the usage of pruning algorithms. Pruning is defined as the reduction of the decision tree size by the termination of unnecessary sub-trees and transformation of splitting-nodes into leaf-nodes. Depending on the time of execution, Pruning is classified either as pre- or post-pruning.

3.1 Pre Pruning

Pre-Pruning takes place during the growing process of the decision tree. It aims at the termination of possible sub-trees, that would lead to overfitting, before they are grown. This way, pre-pruning reduces the training time of the algorithm. A main disadvantage of using pre-pruning is the risk of missing splits that would increase classification accuracy. This could happen if a sub-tree is cut during pre-pruning because the split deteriorates the tree model but the subtree would improve the accuracy of the tree in later splits. This phenomenon is also termed horizon effect. A list of simple top-down pre-pruning rules are:

- Stop if all indices belong to the same class.
- Stop if all attribute values are the same

- Stop if a user defined threshold of splittings is reached.
- Stop if splitting would not increase the purity.

(Qui93, Ber73)

3.2 Post Pruning

Post-Pruning takes place after the decision tree is fully grown. By which means the leaf notes are the product of the native decision tree termination rules. Therefore, they are not prone to horizon effect induced loss of accuracy. However, as the tree needs to be fully grown for post-pruning, this advantage comes at the cost of longer training times.

3.2.1 REP

One of the most simple pruning algorithms is termed Reduced Error Pruning (REP) and was introduces by Quinlan 1987. It is a bottom up approach that prohibits overfitting by replacing complex sub trees with leaf-nodes. The checking procedure is done using an extra testing data set. The REP algorithm starts from the leaf nodes and checks iteratively for each higher internal node whether the replacement of the sub-tree by the majority vote based classification of the internal node would lead to an increase in classification accuracy. If so, the sub-tree is pruned. The pruning-procedure continues until no further increase in classification accuracy is achieved by further pruning steps.

3.2.2 EBP

Another bottom up pruning approach by Quinlan (1993) is the Error?based Pruning (EBP) algorithm. It differs from the REP algorithm by two facts: First, no separate test data set is needed to calculate the necessity of pruning. Second, sub trees can either be replaced by leaf nodes or by the sub tree with the strongest instance population. The EBP algorithm estimates the classification error by calculating the error rate of a node. The error rate is estimated to follow a binomial distribution. The pruning is controlled by a threshold that is termed certainty factor (CF). The CF is introduced as the confidence limit of the error rate. It can therefore be used to estimate the upper limit of the probability that a classification error will occur at the node. Therefore, the EBP algorithm is also classified as a pessimistic approach. The EBP algorithm calculates the classification error of a sub tree as the sum of estimated classification errors of its leave nodes. Similar to the REP algorithm the EBP algorithm substitutes an internal node by a leaf node if its majority vote based classification error is exceeded by the cumulative estimated classification error of its

sub tree's leaf nodes. In an additional step the EBP algorithm tests whether the estimated classification error of an internal node and the estimated classification errors of its leave nodes exceeds the estimated classification error of its sub tree with the strongest instance population, the internal node is replaced by its strongest populated sub tree.

3.2.3 MCCP

The Minimum Cost-Complexity-Pruning (MCCP) is a post pruning algorithm that minimizes the cost-complexity measure $R_\alpha(T)$ of a decision tree T. It is parameterized by the complexity parameter $\alpha \geq 0$ and calculates to:

$$R_\alpha(T) = R(T) + \alpha|T| \tag{3.1}$$

With $R(T)$ being the total classification error rate of all leaf nodes of the tree T and $|T|$ being the absolute number of leaf nodes within the tree T. For a single leaf node t the complexity measure would simplify to:

$$R_\alpha(T) = R(t) + \alpha \tag{3.2}$$

A subtree that has its origin at the internal node t is defined as T_t. Therefore, the complexity measure of a sub tree T_t which root node is indicated by the suffix t is calculated by:

$$R_\alpha(T_t) = R(T_t) + \alpha|T_t| \tag{3.3}$$

With $|T_t|$ being the absolute number of leaf nodes within the sub tree T_t. As defined before, the impurity of an internal node $R(t)$ always exceeds the cumulative impurity of its sub tree leaf nodes $R(T_t)$.

$$R(T_t) < R(t) \tag{3.4}$$

Thus, there exist an effective complexity parameter α_{eff} that satisfies the condition:

$$R_\alpha(T_t) = R_\alpha(t) \tag{3.5}$$

$$R(T_t) + \alpha_{\text{eff}}|T_t| = R(t) + \alpha_{\text{eff}} \tag{3.6}$$

$$\alpha_{\text{eff}} = \frac{R(t) - R(T_t)}{|T_t| - 1} \tag{3.7}$$

At the beginning of MCCP a threshold value for the effective complexity parameter α_{eff} is defined. The MCCP algorithm starts at the leaf nodes of the tree and walks through the decision tree in a bottom up manner and calculates the effective complexity parameter for every inner node. If the inner node with the smallest effective complexity parameter $\alpha_{\text{eff}}(t)$ is smaller than the pre defined threshold value the sub tree T_t will be pruned by substituting the inner node t with a leaf node that classifies the instances by majority vote. This procedure is applied recursively until the the smallest effective complexity parameter $\alpha_{\text{eff}}(t)$ exceeds the threshold value.

Chapter 4

Decision Trees

A decision tree is a visualization of an algorithm that consist from conditional control statements only. In machine learning it is for example used to solve classification problems, by which means as a predictive model to draw discrete conclusions from multi-dimensional observation parameters. These classification trees consist of leaves representing class labels and branches representing conjunctions of observations that lead to distinct class labels. Due to their simple structure classification tree-based decision making is a traceable procedure compared to more complex classification models like random forests or deep artificial neural networks. Therefore, classification trees are very popular in the arising field of explainable artificial intelligence although they are sometimes inferior to the more complex models in terms of accuracy. The construction of decision trees from class-labeled training observations is termed learning. Tree learning is a complex problem that misses an absolute solution until now. Therefore, several tree learning algorithms have been invented during the last decades. The decision tree models used within this research article are

- iterative Dichotomiser 3 (ID3)
- c4.5 trees
- c5.0 trees
- classification and regression tree (CART)
- conditional inference tree (ctree)
- partial decision trees (PART)

and shall be introduced in more detail in the following.

4.1 Growing decision trees

Growing the optimal tree is an NP-Complete problem. It requires O(exp(m)) time. Thus, the problem is intractable even for small data sets. Therefore, tree-based classification will most likely not find the optimal but rather a reasonably good solution. As it is not possible to say in advance which training algorithm will lead to the best result it is always mandatory to compare the performance of different learning algorithms applied to the same classification problem. Once trained the prediction complexity of the tree is $O(\log_2(m))$ which is independent of the number of features.

4.1.1 Top Down Induction of decision trees

One of the most common category of decision tree-based machine learning algorithms is known by the terminus of Top Down Induction of decision trees (TDIDT). All of the TDIDT are characterized by a similar growing procedure during training. Their growing procedure is:

- based upon the Top-Down Principle, stating at the root node.
- sensitive to the frequency of information in the training data.
- not influenced by the order of the data.
- is not incremental.

During TDIDT the tree is grown from the root node and branches into further child nodes until finally leaf nodes are constructed. As this procedure is influenced by the frequency of information, TDIDT algorithms need full access to the training data set at any time. therefore TDIDT algorithms are not incremental.

4.1.2 Recursive Partitioning Procedure

The growing of decision trees follows the recursive partitioning procedure. Lets consider a data set with j instances. Each instance is characterized by i attributes and in case of the training data set by a classification attribute c. Tree growing is started by identifying the attribute x that splits the complete data best with respect to a given metrics. In case x being categorial the split will result in n child nodes. With n being the number if individual categorical values within attribute k. In the case of x being a continuous numerical attribute splitting is performed upon a threshold parameter that is defined with respect to the metrics. The splitting procedure results in two data subsets subsets with $i(x, y) < y$ and $i(x, y) > y$. The node produced by the first split is termed the root node of

the tree. Starting at the root node, splitting is recursively repeated on all emerging data subsets until a stop criteria is reached.

4.2 Iterative Dichotomiser 3

The ID3 is a decision tree algorithm that was introduced in 1986 by J. Ross Quinlan [1]. The ID3 algorithm is entropy-based. It handles feature attributes as categorical values and does not tolerate missing values. It splits the root node by the feature k that results in the largest information gain (see equation (2.8)). During the splitting procedure ID3 creates n child nodes. With n being the number if individual categorical values within attribute k. After the split the feature k is removed from the list of possible splitting features, thus guaranteeing that each feature is only used once for splitting. The process is recursed on the child nodes using the remaining features until the node is pure and therefore, transformed into a leaf or until each feature is used for classification. The ID3 algorithm has the following termination criteria:

- The node is pure (it comprises only instances of the same class).
- The node is empty.
- There are no attributes left for splitting.

If one of the termination criteria is met, it creates a leaf. If the leaf node is pure, it is characterized by the class of its instances. If the leaf node contains instances of several classes it is characterized by a majority vote. If it is empty it is not able to characterize its instances correctly. Therefore, it will be classified as null. The ID3 algorithm tends to preference attributes with many choices (> 2) which results in the growing of predominantly small and wide trees.

4.3 C4.5

The C4.5 algorithm is the improved successor of the ID3 algorithm [2]. In comparison to the ID3 algorithm, the C4.5 algorithm can handle more generalized data including continuous numeric values and missing values. The C4.5 algorithm uses gain ratios instead of gains to quantify the information gain of a split. It handles categorical attribute values similar to the ID3 algorithm by splitting an attribute k that contains n individual categorical values into n child nodes. This results into broad trees if the parameter space of nominal attributes is large. The algorithm transforms continuous numerical attributes to categorical ones by performing a threshold-based binary split. The optimal splitting parameter is found by maximizing the information gain ratio for the splitting attribute with

respect to the splitting threshold. Missing values are handled by the C4.5 algorithm by simply ignoring them during information gain ratio calculation. Like its predecessor C4.5 starts with splitting the data at the root node by the attribute that results in the highest information gain ratio. It recurses the splitting procedure at the subordinate nodes until a further splitting of the nodes would not lead to a further increase in the information gain ratio. At these end points a leaf is finally created. The C4.5 algorithm has similar termination criteria as its predecessor, except for the case of an empty node. For the latter the leaf node is not characterized as null but by the majority voting of its parental node.

4.4 C5.0

The C5.0 algorithm is an optimizes version of the C4.5 algorithm [3, 4]. It brings several improvements on C4.5:

- It is faster than C4.5.
- It is more memory efficient than C4.5.
- It produces smaller trees compared to C4.5.
- It supports boosting.
- It supports weighting.
- It supports winnowing.

4.5 Classification and regression trees

The CART algorithm was first introduced by Breiman in 1984 [5]. It is Gini impurity-based and can handle categorical as well as continuous numerical values. It produces binary trees only. By which means, that non-leaf nodes will always have two children. During training, the algorithm splits the data into two subsets based upon a single attribute k. In the case of categorical attributes the split is performed by answering the question whether $k \in t_k$. With t_k being defined as the attribute threshold. In the case of continuous numerical attributes the split is performed by answering the question whether $k \leq t_k$. The algorithm identifies the optimal split by screening for the pair $(k,\ t_k)$ that results in the purest children.

$$J\left(k, t_k\right) = \frac{m_{\text{true}}}{m} G_{\text{true}} + \frac{m_{\text{false}}}{m} G_{\text{false}} \tag{4.1}$$

Equation (4.1) ist termed the CART cost function. With m being the total number of instances at the node before splitting and m_{true} being the number of instances that fulfill the threshold criterion t_k and nm_{false} being the number of instances that do not. Once the optimal pair (k, t_k) for the node split is found, the procedure is repeated on both of the child nodes until one of the following termination criterions is reached:

- the child node is pure
- the split does not lead to a significant increase in purity

If one of the termination criteria is met, it creates a leaf. If the leaf node is pure, it is characterized by the class of its instances. If the leaf node contains instances of several classes it is characterized by a majority vote.
As suggested by its name, the CART algorithm is also capable of growing regression trees. The CART cost function for regression calculates similar to CART cost function for classification (see eq. (4.1)).

$$J\left(k, t_k\right) = \frac{m_{\text{true}}}{m}\text{MSE}_{\text{true}} + \frac{m_{\text{false}}}{m}\text{MSE}_{\text{false}} \tag{4.2}$$

4.6 Conditional Inference Trees

Conditional inference trees [6] differ from the decision trees introduced so far by separating the identification of a suitable split attribute from the procedure of threshold optimization. The learning procedure of a ctree can be described in three steps. In a first step, The optimal splitting attribute is identified from the set of available attributes. The ctree performs a hypothesis test for each available attribute against the null hypothesis of being independent from the classification attribute. The procedure will stop here, if the null hypothesis cannot be rejected. Otherwise, the ctree will select the attribute with the strongest association to the classification attribute based on the attributes' p-values. In the second step, the splitting is performed. This can be done by any metrics. Finally the third step ist the recursive repeat of steps one and two on the data subsets emerged from former splittings until a stop criterion is reached.

4.7 Partial Decision Trees

Sometimes it is not necessary to let a complete tree grow completely to get the information that is sought. In such a case a PART is a resource-saving alternative. A PART contains nodes that split up into undefined sub trees. In this way, sub trees of interest can be grown quickly without constructing a whole tree. In order to quickly identify a subtree that cannot be further simplified within a PART, the construction procedure and the pruning

need to take place at the same time. The growing procedure of the tree is then terminated as soon as the sub tree of interest has been identified.

4.8 Algorithm Comparison

The main differences between the presented algorithms are summarized in table 4.1

Table 4.1: Algorithm Comparison

Algorithm	ID3	C4.5	CART
categorical attributes	+	+	+
continuousl attributes	-	+	+
Metrics	Entropy	Entropy	Gini impurity
Split Criterium	information gain	information gain ratio	CART cost function
$\sum$ child nodes (categorical)	$\sum$ attribute values	$\sum$ attribute values	2
$\sum$ child nodes (continuous)	-	2	2
missing values	-	+	+
post-pruning	-	EBP	MCCP
empty leaf node handling	null	parental majority vote	no empty leafs

Chapter 5

Rule Based Learners

In general machine learning (ML) algorithms that learn a set of conditional rules from a data set are summarized under the heading of rule based machine learning (RBML). The rule set identified by the RBML algorithm represents the total amount of contextual knowledge learned from the training data.

5.1 Decision rules

Decision rules as used for RBML are in the form of if(condition) → then(prediction) expressions. In this way, decision rules structure a classifier's split decision in a similar way to human thinking. If the set of rules used for classification is small enough and the underlying attributes have a comprehensible meaning, RBML can therefore create models that are easy to explain.

The usefulness of a decision rule can be defined quantitatively by two characteristics: The support and the accuracy. The support defines the coverage of a rule. It is calculated as a fraction of the number of instances that are checked positively under the conditions, in relation to the total number of instances checked. The accuracy defines the confidence of the rule. It is calculated as the fraction of the number of classes that are predicted correctly by the rule in relation to the number of instances that are checked positively under the conditions.

A decision rule set comprises multiple decision rules. The combination of decision rules can lead to several problems. A prominent problem is the case that for a certain instance no decision rule from the set applies. This problem can be solved by introducing a default

rule that for example classifies the instance simply by a majority vote. Another common problem that may arise is the case of overlapping rules, by which means that two or more rules apply to a certain instance. Rule combinations strategies can be used to clarify which decision rule applies in such a case. Two prominent rule combination strategies are the organization of a rule set in the form of a decision list and a decision set. Rules organized in a list are applied in order in case of an overlap. The first rule whose condition is true is applied. Within a decision set overlapping applicable rules are handled by multiple strategies. A simple example would be a majority based classification that is weighted by the rule accuracy. A main drawback of this approach is that the explainability suffers if multiple rules are are applied in combination.

5.1.1 Sequential Covering

Sequential covering comprises algorithms that iteratively learn single rules to create a decision list or set. It is based upon the concept of divide and conquer. At the beginning, the whole data set is covered by the algorithm in order to learn the first rule. Subsequently, all instances covered by the introduced decision rule are removed from the data set. This procedure of rule learning is repeated on the remaining data until no more instances are left or alternative stop criteria are met. A prominent representative of this superior class is the repeated incremental pruning to produce error reduction (RIPPER) algorithm. Sequential covering can also be utilized in order to learn rules from decision trees.

5.2 Learning Rules from Trees

Next to RBML algorithms, decision rules can also be learned from all kind of decision trees using the method of sequential covering. For this purpose a decision tree is grown by its learning procedure. Post processing algorithms like pruning may be applied. The rule learning procedure starts at the root node of the tree by selecting recursively the purest node which is classified by the lowest misclassification error until a leaf node is reached. The resulting path throughout the tree from root node to leaf node represents a single rule. The prediction of the rule is calculated from the majority-based classification of the leaf node. The conditions consist of the split parameters of the individual inner nodes. This procedure is repeated until each leaf node has been covered by a rule.

5.3 RBML Algorithms

Next to the possibility to rules from fully grown decision trees there are several rule based classification algorithms invented within the last decades.

5.3.1 OneR

Probably the most simple rule indication algorithm is the one rule (OneR) algorithm [7]. It creates a set of decision rules for the attribute that is most predictive in classification. In order to identify this attribute it learns rules from every single attribute. It starts with selecting an attribute and creates one rule per unique attribute value, which predicts the class that is most often populated by the instances with that particular attribute value. This procedure is repeated iteratively through all attributes. Finally, the OneR algorithm selects the attribute which rules are that is characterized by the optimal accuracy.

If accuracy comparison ends up in tie, the OneR algorithm either selects an attribute by choice or favors the attribute whose rule set yields the lowest p-value in a Chi-square test.

Due to its simplicity, the OneR algorithm is easy to interpret and often serves as a benchmark for more complex RBML algorithms. The simplicity has its price. The OneR algorithm can only handle categorical data and does not tolerate missing values. Therefore, continuous values need to be transformed to categorical and missing values need to be imputed before decision rules can be learned by the OneR algorithm.

5.3.2 RIPPER

The RIPPER method [8] is actually a combination of multiple RBML algorithms that can be used to create either a decision list or set. The approach can be divided into three phases: Growing phase, pruning phase, and optimization phase. During the grow phase the algorithm uses the method of sequential covering. It identifies the optimal splitting attribute by the increase in information gain and splits the data in order to add a condition to a rule. Each growth step is followed by a pruning step. If the additional condition does not reduce the entropy any more, the procedure is stopped and the rule is pruned. This procedure is repeated on the remaining instances until the remaining data are classified perfectly or the RIPPER runs out of possible splitting attributes. In the latter case, the rule prediction is calculated by the majority vote of the remaining instances. The growing and pruning circle is followed by an optimization phase. This way the RIPPER generates

one rule at a time. Instances that are covered by an invented rule are removed. The rule set is iteratively enlarged until no instances remain in the training set.

5.3.3 PART

The PART algorithm [9] combines the divide and conquer strategy with decision tree approaches. Like the RIPPER it iteratively invents a rules and subsequently removes the instances that are covered by the rule until no instances are left. It differs from the RIPPER in the way the rules are created: PART grows a partial decision tree on the current set of instances. The rule is then learned from the leaf node of the partial decision tree with the greatest coverage. After rule definition, the tree is discarded and the instance continuum is reduced upon the the rules of the divide and conquer method. New rules are iteratively created until no instances are left.

Chapter 6

Parameter Definition

- C: class attribute (a set of discrete class values)
- $\hat{C}$: number of unique class values in C
- A: set of attributes
- α: splitting attribute
- S: set of instances
- $V(x)$: value of variable x
- p: probability
- $G(p)$: Gini impurity
- H: entropy
- I: information content
- IG: information gain
- Int$I$: intrinsic attribute information
- M: misclassification error
- i, j, k: running parameter

Chapter 7

Abbreviations

AI artificial intelligence

CART classification and regression tree

CF certainty factor

ctree conditional inference tree

EBP Error?based Pruning

ID3 iterative Dichotomiser 3

MCCP Minimum Cost-Complexity-Pruning

MCE misclassification error

ML machine learning

OneR one rule

PART partial decision trees

RBML rule based machine learning

REP Reduced Error Pruning

RIPPER repeated incremental pruning to produce error reduction

TDIDT Top Down Induction of decision trees

XAI explainable artificial intelligence

Bibliography

[1] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[2] RC Quinlan. 4.5: Programs for machine learning morgan kaufmann publishers inc. *San Francisco, USA*, 1993.

[3] Max Kuhn and Kjell Johnson. *Applied predictive modeling*, volume 26. Springer, 2013.

[4] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[5] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[6] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics*, 15(3):651–674, 2006.

[7] Robert C Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.

[8] William W Cohen. Fast effective rule induction. In *Machine learning proceedings 1995*, pages 115–123. Elsevier, 1995.

[9] Eibe Frank and Ian H Witten. Generating accurate rule sets without global optimization. 1998.