Layerwise Training of Deep Rule-Based Networks Using Stacking

Schichtenweises Training von tiefen regel-basierten Netzwerken durch Verwendung von Stacking

Bachelor thesis by Daniel Jung

Date of submission: June 30, 2020

1. Review: Prof. Dr. Johannes Fürnkranz
2. Review: Dr. Eneldo Loza Mencía
3. Review: Michael Rapp

Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department
Knowledge Engineering Group

Abstract

Models that are created by machine learning algorithms to perform classification can benefit from a network structure. This is among others the case for feed-forward deep neural networks that process the data instance layer by layer before the last layer predicts a class as the model's output.

This prediction is only based on the intermediate data received by the previous so called hidden layer. Therefore such a network does not only learn a mapping from a data instance to a class but also a mapping to at least one intermediate feature representation of that instance. Since deep neural networks that have several hidden layers and therefore several different intermediate feature representations often outperform shallow neural networks with just one hidden layer for many data domains it follows that these feature representations can progress from hidden layer to hidden layer in terms of their usablity for more accurate class predictions. In other words using the feature representations of deeper layers the model generalizes better beyond the training data to unseen data.

Some research suggests that this increase of the capability to generalize is not restricted to specific types of networks like deep neural networks and their specific training methods like backpropagation but is to some extent a property of the network structure of the model itself.

In this thesis it is investigated whether a network of rules can exhibit some indication of this property of increasing generality of the intermediate feature vectors of succeding layers. Rules in one layer can predict classes and the class predictions of this layer can make up an intermediate feature vector for the rules in the next layer. Each of theses rules in the next layer can then serve as a meta classifier of the rules in the preceeding layer beeing base classifiers as defined by the ensemble method stacking. It must also be ensured that the rules that are always base classifiers for each meta classifier rule in the succeding layer have diversity of an extent necessary for an ensemble method like stacking to improve the predictive accuracy.

Different methods to create a performance enhancing kind of diversity are explored in various combinations experimentally. The results show no favorable method in terms of a significantly higher accuracy of the network prediction.

Nevertheless, it cannot be concluded that these methods cannot be successful. The network configurations investigated in this thesis often produce relatively good results already when using two layers.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

Classification models represented as a network such as a deep neural network can profit in terms of predictive performance on unseen data from feature transformations that are performed from network layer to network layer. I. e. the model's generalization capability can in many cases improve by an increased depth of the network. Each layer outputs then a feature vector to the nodes of the next layer that is more suitable for the classification task then the one it received from the previous layer. In general, it is not clear what the meaning of these generated intermediate features is in terms of the semantics of the respective data domain. But since they enable a more accurate separation of the classes than the original feature vector a semantically meaningful representation of such feature vectors has advantages.

They could guide a human user of the model to understand the reasons why a specific class is predicted. The fixed features of the original data might correspond to higher level features that are more interesting for the interpretation why an instance is predicted as a member of a specific class. If such a higher level feature exists and would be captured in a rule that can be interpreted in terms of preceding rules of preceding layers, then new higher level features can easily be expressed by making use of the network structure. The features of the rule would then have to be recursively substituted by the features of the preceding rules in the next preceding layer until they are expressed in terms of the low level features of the original data.

If rules that correspond to higher level features can be induced in deeper layers of the network those rules could potentially expose the reason why a hypothesis generalizes better.

2. Foundations of Network-Based and Rule-Based Binary Classifiers

This chapter covers foundations underlying the learning of a binary classifier by supervised training. The goal of this machine learning task is to find a classifier that outputs a binary value representing the class of an object after a data representation of that object is given as input to the classifier. From the perspective of the use case to classify given objects this is a predictive data mining task and the learned classifier can be described as a model, a theory or a hypothesis about the mapping of objects of unknown class to the class they belong.(Fürnkranz, Gamberger, and Lavrac 2012). In terms of analyzing why different hypotheses yield different predictive performance a description of possible patterns in the data representation and their role in the classification procedure can be of interest.

We refer to the algorithm that learns and then outputs the classifier by supervised training as training algorithm. The classifier is outputted as an executable program that performs a classification. This program is a representation of the learned hypothesis that is used by a machine to perform the classification task. It performs a data transformation from the data representation of an object to the binary representation of the class this object belongs to. The training algorithm conducts the learning task essentially as a search through the space of possible hypotheses (Mitchell 1982). The hypotheses considered during this search in the form of data structures of the training algorithm can be represented in a hypothesis description language(Fürnkranz, Gamberger, and Lavrac 2012), a symbolic language that model the semantic of these data structures.

In the following sections we first will lay out the general functionality of binary classifiers followed by an overview of the modularity of the data transformations in network-based and not explicitly network-based binary classifiers in the form of rule sets including single rule classifiers. Finally, we will outline the basic algorithmic structure of how members of the family of artificial neural networks ANN as binary classifiers and classifiers in the form of single rules, rule sets and ensembles of such classifiers search through the space of possible hypotheses.

2.1. The Task of Learning a Binary Classifier

In this section we lay out the fundamental concepts involved in the supervised task of learning a binary classifier.

2.1.1. Binary Class Representation as Classifier Output

For a binary classification task a set of two nominal values that represent the two classes is predefined that specifies to which two classes objects can belong. This set is referred to as the binary class attribute $\mathbb{C}$. The

output of a binary classification $\hat{c}$ that the classifier outputs for an object, also referred to as prediction, predicts one of the two values of the binary class attribute to be the class the object belongs to. If one of the two classes represents the fact that an object belongs to a concept and the other class represents the fact that the object doesn't belong to the concept the binary classifications task is equivalent to the task of concept learning.

$$\hat{c} \in \mathbb{C}, with\ |\mathbb{C}| = 2 \qquad \text{(Binary Prediction)} \qquad (2.1)$$

2.1.2. Object Representation as Classifier Input

Specified Set of Objects

For the task of binary classification we must assume a specified set of objects that can be divided into two subsets, i. e. the two classes. This means the objects must be different, the difference must be observable and the observation of the differences must be represented. The specified set of all possible representations for a given task is referred to as data description language (Fürnkranz, Gamberger, and Lavrac 2012).

Instance Space

We consider only the case where the set of objects is predefined by a given data format for the representation of each object specified in the following data description language. We refer to objects also as instances I and to the set of possible objects as instance-space $\mathbb{I}$. An object representation is predefined as a vector of fixed dimension l where each component is an element of a set called attribute $\mathbb{A}$. The attributes can be different sets of either continuous or discrete values v. Continuously valued sets are also referred to as numerical attributes. A training algorithm can usually assume a relevant numerical ordering of those values. Discretely valued sets are referred to as nominal attributes for which a relevant ordering is also possible, e. g. an integer valued set, but is generally not assumed at the time before training. The instance space is the cartesian product of the fixed number of attributes.

$$\mathbb{I} = A_1 \times ... \times A_l \qquad \text{(Instance Space)} \qquad (2.2)$$

2.1.3. Examples as Training Algorithm Input

Before the training algorithm begins its search for the hypothesis it can be supervised by training examples. In this thesis we only consider a supervised training algorithm in which case for a fraction of the predefined objects of the instance space the class is known. This knowledge is what is passed from the supervisor to the machine as training data input for the training algorithm that learns the hypothesis. We refer to this input as training data $\mathbb{T}$.

$$\mathbb{T} \subset \mathbb{E} \qquad \text{(Training Data)} \qquad (2.3)$$

Each example E is given in the form of a vector v of l numerical or nominal values v_1 to v_l that identifies an instance from instance space and a class representation c that represents the known class of this instance.

$$E := \langle v_1, ..., v_l, c \rangle \in \mathbb{T}, with\ \mathbb{T} \subset \mathbb{E} = A_1 \times ... \times A_l \times \mathbb{C} \qquad \text{(Training Example)} \qquad (2.4)$$

We refer as already implied to unclassified objects as instances and to those pre-classified by the supervisor as examples (Fürnkranz, Gamberger, and Lavrac 2012). Test classification for evaluation purposes is performed on instances I of withhold examples of the training data whose class value c is kept invisible to the classification algorithm but stored to evaluate the classifier after classification.

$$I := \langle v_1, ..., v_l \rangle \in \mathbb{I}, with\ \mathbb{I} = A_1 \times ... \times A_l \qquad \text{(Test Instance)} \qquad (2.5)$$

The cartesian product of the set of instances $\mathbb{I}$ and the set of classes $\mathbb{C}$ is known as example space $\mathbb{E}$ (Fürnkranz, Gamberger, and Lavrac 2012).

$$h \subset \mathbb{E} = \mathbb{I} \times \mathbb{C} \qquad \text{(Hypothesis in Example Space)} \qquad (2.6)$$

A binary example space for a two-dimensional numeric valued instance space is shown in Figure 2.1. The size of the example space of a binary classification task is $\mid \mathbb{I} \times \mathbb{C} \mid = \mid \mathbb{I} \mid \cdot \mid \mathbb{C} \mid = \mid \mathbb{I} \mid \cdot 2$.

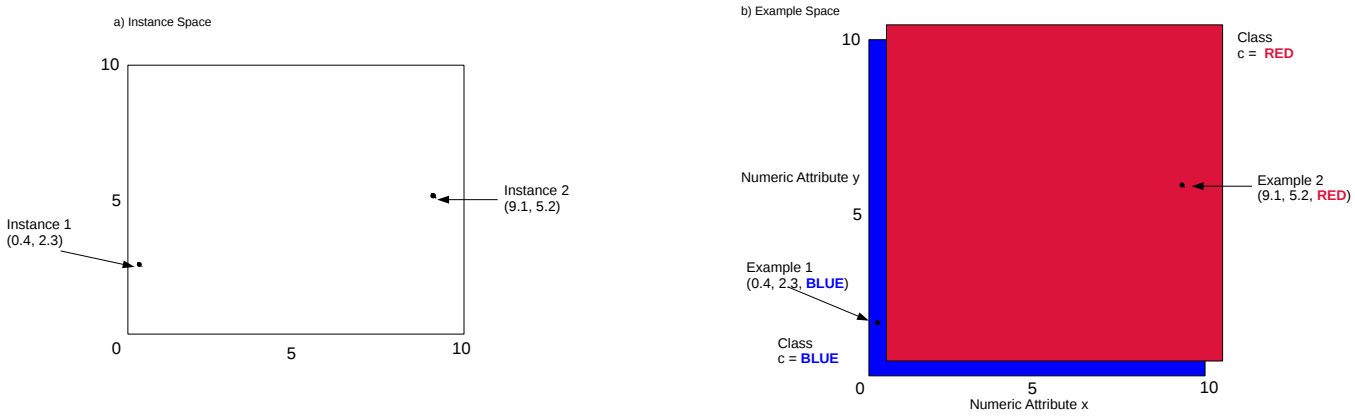

Figure 2.1.: a) Two-dimensional Numeric Instancespace
b) Binary Class Example Space

2.1.4. Semantic Hypothesis as Learning Goal

The goal of the training algorithm is to learn a hypothesis h that associates the correct class representation $\hat{c}$ to any object representation within the instance-space I. Usually, the existence of the true class for each possible object is assumed. The hypothesis is a function h that models the classifier's functionality to predict the correct class for each given object of the instance space. Two different hypothesis representations that represent the same function h are semantically equivalent and only syntactically different. The hypothesis

description language is specified to represent all or a subset of the semantically different hypotheses h syntactically (Fürnkranz, Gamberger, and Lavrac 2012). In this thesis we use the term hypothesis semantically and hypothesis representation for the hypothesis' syntactic form.

$$h : \mathbb{I} \rightarrow \mathbb{C} \qquad \text{(Hypothesis)} \tag{2.7}$$

A function h is a subset of the example space $\mathbb{E}$. Each semantically possible hypothesis consist of half of the elements of the example space, i. e. any possible hypothesis has the size of the instance space $\mid \mathbb{I} \mid$ implying also that each instance of the instance space occurs exactly once in the elements of the hypothesis.

We refer to the set of all semantically possible hypotheses as hypothesis space $\mathbb{H}$. It has $\mid \mathbb{H} \mid = \mid \mathbb{C} \mid^{\mid \mathbb{I} \mid}$ elements. This is the model capacity of a memory model that memorizes the mappings of all possible instances to their class individually. The example space is equal to the union of all possible hypotheses.

$$\mathbb{H} = \{h_1, ..., h_{|\mathbb{H}|}\}\ , with\ \mathbb{E}\ = \bigcup_{i=1}^{i=|\mathbb{H}|} h_i \qquad \text{(Hypothesis Space)} \tag{2.8}$$

Often, only a subset of the hypothesis space limited by the hypothesis description language can be selected or constructed by the training algorithm. The difference in structure and size of the hypothesis space created by the hypothesis description language defines a language bias. Such a bias can restrict the search for the hypothesis to those hypotheses in the hypothesis space that are more likely to generalize to unseen data.(Fürnkranz 1999).

We refer to the remaining capacity as representational capacity.

2.1.5. Impact of Hypothesis Representation on Hypothesis Search

The representation of the hypothesis h in the hypothesis description language is a semantically equivalent abstract symbolic resemblance of the data transformations in the classification routine. E. g., it can be expressed as a set of composite functions in the form of equations in the case of ANN classifiers or in a syntax resembling propositional logic in the case of rule set classifiers. Such representations can have different degrees of descriptiveness with respect to possibly relevant patterns in the data. If the learning task is not purely predictive but has a secondary descriptive nature this descriptiveness can be a deciding factor for the choice of the hypothesis representation. But only because some representations are more descriptive about the instance space pattern that influence the decision what class is predicted by the classifier, doesn't mean that these pattern couldn't be less descriptive encoded in the representation. In the latter case these pattern would still be the basis for the correct prediction.

Memory-based vs Computational Data Transformation

As a machine readable program the classification procedure utilizes memory and computation resources to perform the classification. A purely memory oriented representation would be a kind of look up table where the elements of the instance space I are the keys and the classes c are the values. This can be impractical for large instance spaces. But even for infinite instance spaces it is not impossible in principle unless it is expected that the instances to be classified can be given with unbounded precision. The mapping for all

unseen instances however, would in this case still have to be calculated hypothetically during training of the classifier. Therefore a computational data transformation from unseen instances to the class representation must take place at this time that is capable to find and utilize the patterns in the training data that generalize to the complete instance space.

This memory-based hypothesis representation is an extreme example for a strictly predictive hypothesis with no descriptiveness. For each instance the class can be looked up, i. e. be predicted, but no reason for that prediction is exposed. The classifier has no structure but the immediate mapping from instance to class. For a hypothesis to be descriptive the modeled classification must include the performed data transformations that reveal patterns in the data relevant for the prediction, i. e. the instance representation is not immediately associated with its class as in a look up table but transformed by a series of arithmetic and / or logical operations into one or more intermediate representations before the transformation into the class representation takes place.

Decision Boundary

One way to visualize a hypothesis in the example space is to separate the instances by the class the hypothesis predicts for them. If the instance space is made of numeric attributes the hypothesis can be represented as two unions of continuously connected instances of the same class forming hyper volumes in instance space. The surface of one of these two unions of hyper volumes is called decision boundary. The dimension of the boundary is $L - 1$, i. e. one less than the dimension L of the instance space. For instance spaces of low dimensionality and the one-dimensional binary class space the hypothesis can be visualized.

As an example for such a visualization let first an object be represented by one numeric attribute $A_1 = [0, 10]$ with one-dimensional instance space I being $I = A_1$ and all instances $X \in [0, 2] \cup [8, 10]$ belong to class RED and all instances $Y \in (2, 8)$ belong to class $BLUE$. The decision boundary is the set of objects $< 2 >, < 8 >$. If a second numeric attribute not correlated with the class attribute is added to the object representation, creating a two-dimensional instance space, the second attribute would be irrelevant for classification as shown in Figure 2.2. This results in the extension of each zero-dimensional point of the decision boundary to a one-dimensional vertical or horizontal straight line as decision boundary consisting of all the objects for which the relevant attribute value is equal to the one dimensional attribute value in the previous example.

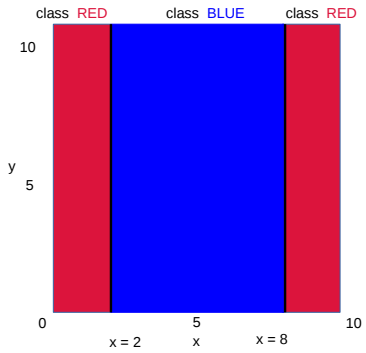

Figure 2.2.: Decision Boundary for One Relevant Attribute

In the general case however, both attributes correlate with the class attribute and are therefore both relevant for the hypothesis. But the correlation of objects with the two classes could allow division of the instance space into subsets in such a way that for each subset there is only a correlation between one attribute and

the class attribute and the other attribute is for this subset irrelevant. Then unions and intersections of these subsets whose elements can be classified based on one attribute alone can be formed. The surface of these hyper volumes, i. e. rectangles, are straight vertical and horizontal lines that don't cross and always end at the borders of the instance space.

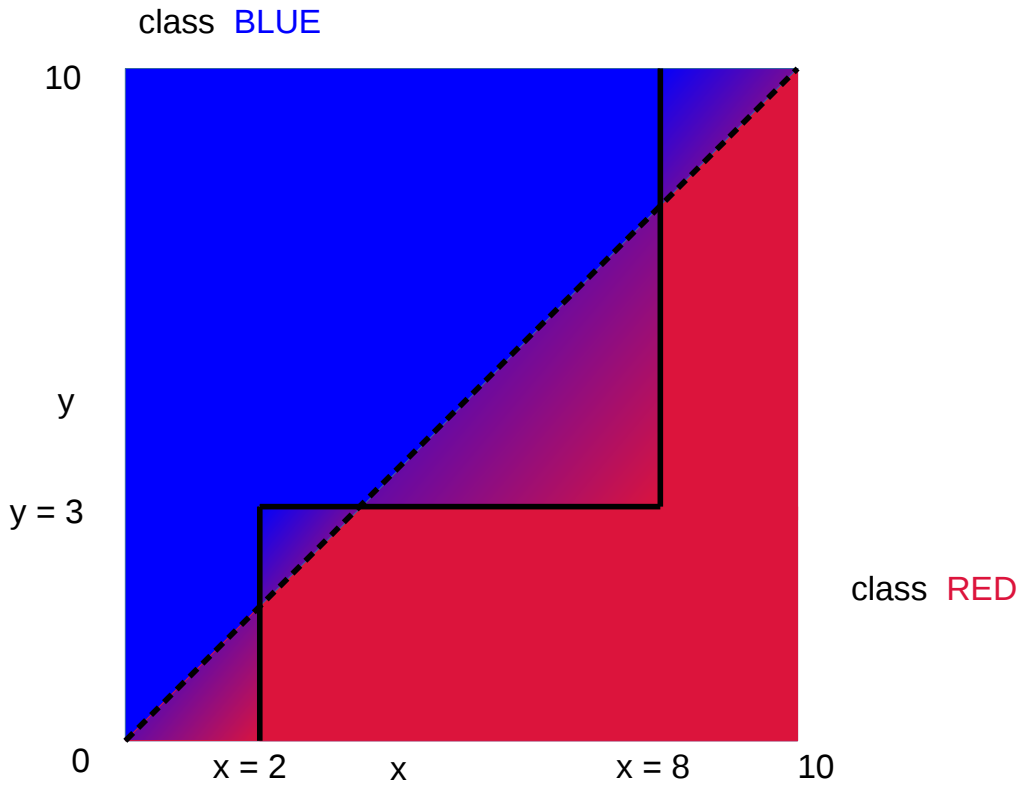

Figure 2.3.: a) Decision Boundary for United and Intersected Subsets Each Based on Single Attribute (Continuous Line)
b) Decision Boundary Based on Correlated Attributes $x = y$ (Dashed Line)

On the other hand it is also possible that the correlation of both attributes with the class attribute cannot be resolved in this way for the complete instance space because the possible subsets with one relevant attribute become so small that infinite many would have to be combined by unions and intersections. This corresponds to diagonal straight decision boundaries for constant correlations as shown in Figure 2.3 as the dashed line. The correlation with class $BLUE$ is based on the correlation $x < y$ between x and y, with class RED is based on the correlation $x > y$ between x and y. Also the boundary can only be expressed in terms of both attributes x and y, namely as $x = y$.

2.1.6. Hypothesis Search Bias

The training algorithm that performs the search through the hypothesis space is usually initialized by some hypothesis represented in the biasing hypothesis description language and as an executable classifier in accordance with that hypothesis description language. This initial hypothesis is implemented as an adjustable

program that can be adjusted by the training algorithm. Only a specified set of possible adjustments can change the state of the hypothesis in the biased representational hypothesis space in the specified order. The restriction of possible adjustments is referred to as search bias(Fürnkranz 1999).

The now remaining subset of reachable hypotheses restricts the representational capacity to the effective capacity of the model.

Evaluation of Hypothesis

The training algorithm also applies an evaluation function that takes a set of predictions made by the initial hypothesis and the known classes of the training data as input and calculates an evaluation value according to the implemented evaluation function. This is the attempt to quantify the difference between the current hypothesis and the desired hypothesis which is also often referred to as error in which case the evaluation function is called a error function. The quantification measure is referred to as metric. This creates an order between different hypotheses indicated by the metric value of each hypothesis.

Update of Hypothesis

After calculating the metric value for each hypothesis that can be reached in the next adjustment step as a function of the possible adjustments the highest rated hypothesis can be chosen. If the search bias doesn't exclude infinitesimal adjustments of the hypothesis that function is derivable with respect to the possible adjustments and the derivative can be used to find the highest gradient or in the case of an error value the lowest gradient.

Constructive Induction

Instead of updating the hypothesis only within the given hypothesis description language, the language can be treated as open and allow new representations to be constructed by applying arithmetic and logical operators to compute new attributes from existing ones. (Fürnkranz 1999).

2.2. Modularity of Data Transformations in Networks

When the structure of a network is used, the data transformation applied to perform the classification can be split into a set of parallel data transformations that transform the original object representation into multiple intermediate data representations as shown in Figure 2.5, i. e. the transformation is performed in a modular structure.

The data representation of an object enters a network of data processing units. The set of necessary data transformation modules that produce the intermediate data representations is sometimes called a hidden layer, e. g. in the context of Neural Networks. Since this multiplicity of data representations has now to be mapped to the representation of a class, at least one consequential, i. e. serial data transformation is also necessary to perform this mapping. This structure is a feed-forward network because the transformed data is only propagated forward towards the final data transformation unit of the last layer and never backwards into the units of layers that the data have already past. The network shown in Figure 2.5 is a so called shallow

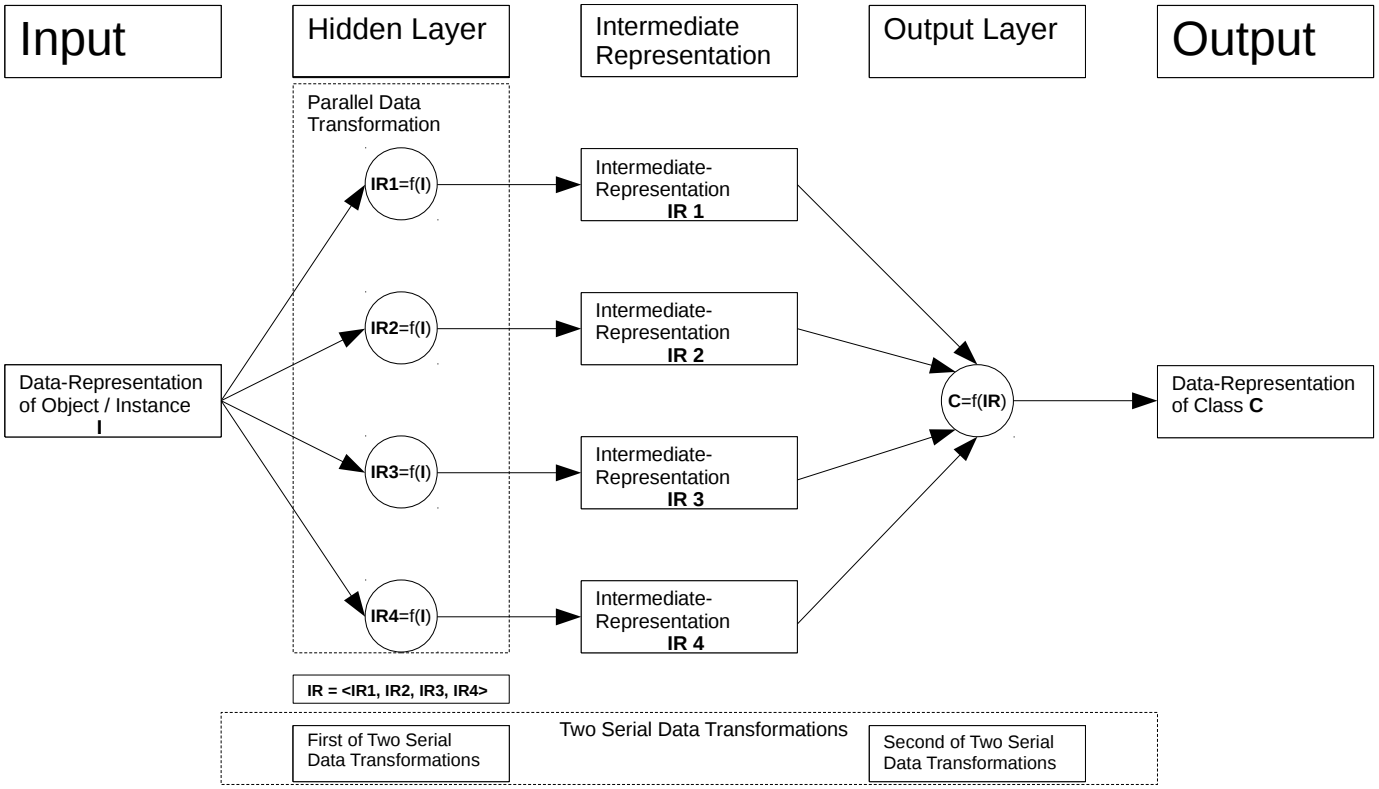

Figure 2.4.: Data Transformations of a Network-Based Binary Classifier with One Hidden Layer (Shallow Network)

network because it has only one hidden layer. If the outputs of the first hidden layer are connected to the inputs of a consequential hidden layer consisting of several parallel data transformation units or even more hidden layers are added to the series of data transformations then the network is called a deep (neural) network.

The network in Figure 2.5 could represent a Neural Network. Then each data transformation module would be implemented as a neuron that implements a data transformation according to a composition of a linear function with a non-linear activation function. This will be laid out in more detail in section 2.3. In the context of rules as binary classifiers on the other hand the data transformation modules could be implemented as rules or other rule-based data transformation modules. This will be laid out in more detail in section 2.4.

2.3. Artificial Neural Network ANN Classifier

One already mentioned type of network-based classifier model that is successfully applied to many interesting and relevant classification tasks is an ANN or multi-layer perceptron. This is a machine learning model that can be trained as a binary classifier (Murphy 2012). It combines layer-wise adaptable linear functions followed

by a fixed non-linear activation function that makes the transformation altogether non-linear as network data transformation modules called neurons. See Figure 2.6. The weight parameter for the linear functions define the space of possible adjustments for hypotheses considered by the training algorithm(Bishop 2007).

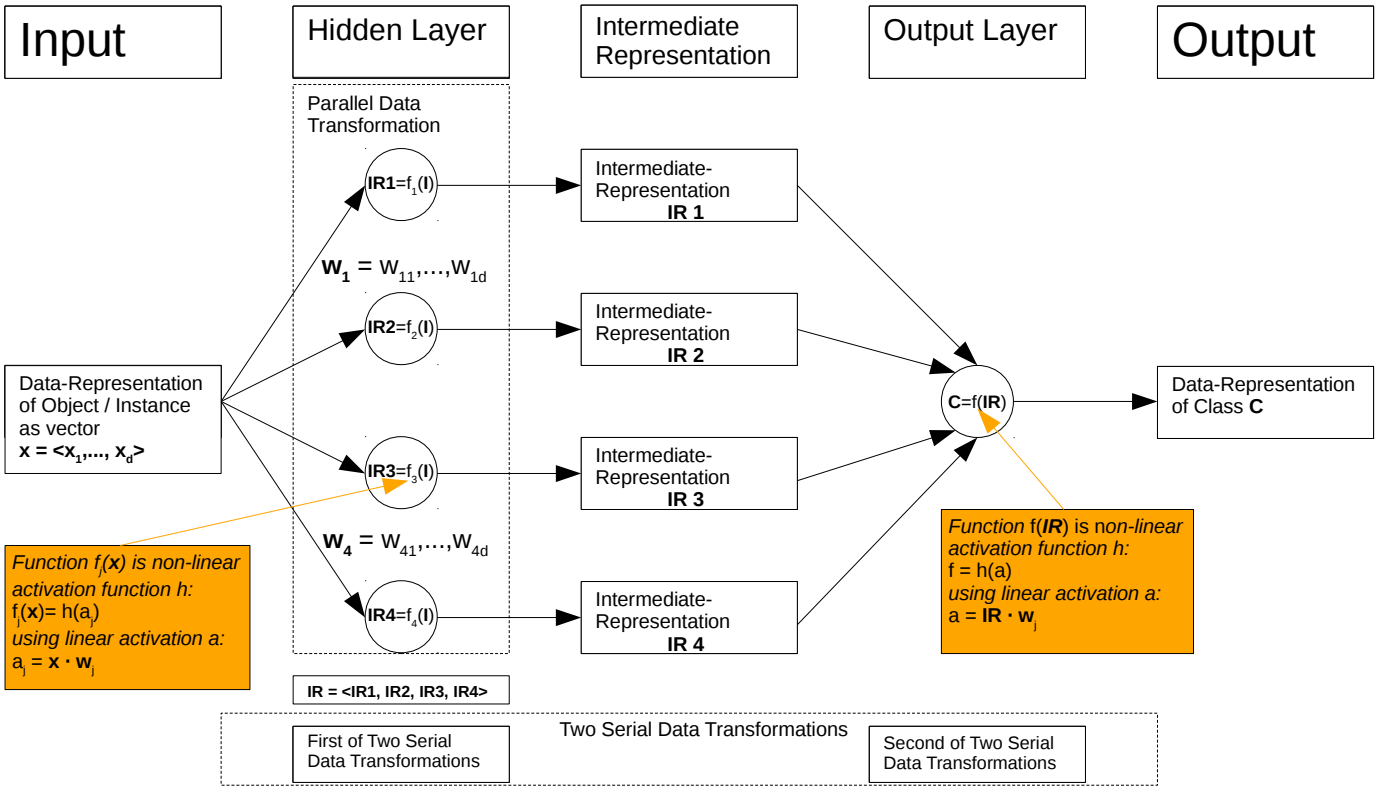

Figure 2.5.: Neural Network

2.3.1. Data Transformation

The training data, usually numeric, i.e. a vector of attribute values from continuous domains is classified after being layer-wise non-linear transformed. Each neuron's output range of a hidden layer can be interpreted as one attribute or dimension in an object's intermediate representation of that layer. This means that the instance space dimensionality for the intermediate representation of each layer depends on the number of neurons of that layer. In each hidden layer the data is often still represented as a vector of attribute values from continuous domains. So the attributes of the intermediate object representation are numerical. Only the last transformation in the output layer maps the data into the discrete domain of the binary class attribute.

2.3.2. ANN Classifier Hypotheses Representation

ANN classifier hypotheses transform the instance representation into intermediate representations before they transforms the last intermediate representation into the representation of the predicted class. Intermediate representations have not the descriptiveness of predicates of logically connected input values. They are numeric encodings of patterns relevant for the classification but not descriptive to a human observer and resemble the probability of exclusive or overlapping participation of instances in subsets of the instance space.

2.3.3. ANN Classifier Hypotheses Search

The goal of training an ANN for binary classification is to iteratively adjust the weight vectors w of each layer towards a global minimimum of the cross-entropy error function E (Murphy 2012).

$$E = \qquad \text{(Cross-Entropy Error Function)} \tag{2.9}$$

The update of the weight vectors for each layer is performed backwards starting at the last layer, a procedure called back-propagation. The updates are based on the gradient of the error function with respect to the respective continuous and derivable weight vector(Bishop 2007). This update selection is called gradient descent.

2.4. Rule-based Classifiers

For rule-based classifiers the training algorithm learns so called rules as classification model. A single rule can model a classifier. But generally the classification algorithm is modeled by more than one rule, i. e. a rule set.

2.4.1. Single Conjunctive Rule Classifier

A rule includes a representation of a subset of the instance space as its body B and a representation of a class that is predicted for all the instances of this subset as its head H.

$$H \leftarrow B \qquad \text{(Single Rule)} \tag{2.10}$$

If the rule condition is met for an instance of this subset then such an instance is said to be covered by that rule (Fürnkranz, Gamberger, and Lavrac 2012).

Hypothesis Representation

The covered instances are represented by statements of propositional logic. Rules in first order logic are not considered here. The basic building block for propositional rules is a propositional feature. This is a logical predicate that is conditioned by the truth of whether a pair of a variable and a constant attribute value is an element of one of the relations " $=$ ", " $<$ " or " $>$ ". This predicate evaluation takes place as a parallel data transformation of a given instance representation. The attribute values of the input vector are processed in place of the respective variables in the feature of the rule.

The set of features of a conjunctive rule conditions a predicate to be true if the feature predicates conjunctively are true. Each feature represents a subset of the instance space and the conjunctive rule's body represents the intersection of these subsets. An example for conjunctive rule representations is given in Table 2.1.

Rules		
Conditions as Literals c	Logical Conjunction of Literals	Conjunction as Rule Body
$c_1 = if(x < 2), c_2 = if(y < 3)$	$\neg c_1 \wedge c_2$	$RED \leftarrow x > 2 \wedge y < 3$
$c_1 = if(x < 2), c_2 = if(y < 3)$	$c_1 \wedge \neg c_2$	$BLUE \leftarrow x < 2 \wedge y > 3$
$c_1 = if(class = 1st), c_2 = if(age = adult)$	$c_1 \wedge \neg c_2$	$yes \leftarrow class = 1st \wedge age \neq adult$
$c_1 = if(class = 1st), c_2 = if(age = adult)$	$\neg c_1 \wedge c_2$	$no \leftarrow class \neq 1st \wedge age = adult$

Table 2.1.: Examples of Single (Conjunctive) Rules

Induction of Single Conjunctive Rules

To initialize the search hypothesis the training algorithm creates candidates for rules. For binary classification it can only create candidates for each of the two classes. For which of the two classes a rule is learned can be specified independently from the training data or as a function of some information retrieved from the training data. It is possible to learn a rule for the majority/ minority class which is the class that is associates to most/ the least instances in the training data. The choice for the class in the rule's head can also be based on the rule's evaluated performance by evaluating a rule for each class and choosing the one that performed better.

After the class for the candidate rule's head is determined the instances of all the training examples are classified with the candidate rule and the predictions are compared to the true class that the example relates to the instance. This is part of the evaluation step in the hypothesis search. If the true class of the example is predicted we have a true positive. The total number of true positives is referred to as TP. If the false class of the example is predicted we have a false positive, their total number is referred to as FP. If the candidate rule doesn't cover the instance, but the class the rule would have predicted is the true class we have a false negative, counted as FN. Finally, if the false class is correctly not predicted we have a true negative, counted as TN. These four counts form a so called two-dimensional confusion matrix CM.

$$CM = \begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix} \qquad \text{(Confusion Matrix)} \qquad (2.11)$$

Evaluation Function for Single Rule

The candidate rule can now be evaluated by an evaluation function $Eval$ that uses the confusion matrix CM resulting from the classification of the training data examples by the candidate rule.

$$Eval(CM_{rule}) = RankingValue \qquad \text{(Evaluation Function)} \qquad (2.12)$$

This allows to compare the chosen candidate rule and refinements of the candidate rule and to make a choice between them based on the ranking value. The refinements are the possible adjustments that bias the found hypothesis according to the search bias.

An initial most general rule guarantees maximum TP. Specializing refinements should for the sake of completeness never sacrifice TP and to maintain consistency avoid covering any FP. But if such refinements are not sufficient to don't exist or can not be found with reasonable effort a trade off has to be found for a good heuristic solution where completeness and consistency are only maximized. An evaluation function has to be chosen as a trade off between the goal of a complete hypothesis, that is capable to make predictions for every instance in the instance space and a consistent one that predicts the correct class for each instance.

Therefore the rule must at the same time cover many positive instances, i. e. $TP/(TP + FN)$ or recall should be maximized, i.e. FN should be minimized.

$$Recall(CM_{rule}) = TP/(TP + FN) \qquad \text{(Recall - Evaluation Function)} \qquad (2.13)$$

But the rule must also be precise, i. e. $TP/(TP + FP)$ should be maximized, i. e. FP should be minimized.

$$Precision(CM_{rule}) = TP/(TP + FP) \qquad \text{(Precision - Evaluation Function)} \qquad (2.14)$$

But a refinement can at the same time decrease FP as intended and TP as not intended. A high decrease of FP might be worth a low decrease of TP. If refinements can only specialize then lost TP can never be regained. So if there are still many refinements ahead in the training algorithm then the chance is higher that FP can be reduced later without loosing to many TP. This implies that loss of TP should be less tolerated in the beginning.

But depending on whether the rule is a single classifier or the rule's classification is combined with the classifications of other rules that can compensate for shortcomings of this rule like low coverage the trade off shall be chosen differently. The f-measure evaluation metric calculates the weighted harmonic mean of recall and precision with a variable weight parameter β.

$$F{-}measure(CM_{rule}) = \frac{(\beta^2 + 1) \cdot Precision(CM_{rule}) \cdot Recall(CM_{rule})}{\beta^2 \cdot Precision(CM_{rule}) + Recall(CM_{rule})} (F - measure - Evaluation\ Function) \qquad (2.15)$$

The algorithm to find the best single rule according to a given evaluation function is shown as Algorithm 2.1.

The single rule's missing capability to represent disjunctions of arbitrary value sets restricts its expressiveness to represent arbitrary subsets of the given instance space $\mathbb{I}$.

Algorithm 1 FindBestRule Algorithm (cf.Fürnkranz 1999).

Require: $T = P \cup N$ {required predicted class $\hat{c}$ splits the training examples into T positive examples with $\hat{c} = c$ and negative examples with $\hat{c} \neq c$}, evaluation function $eval$, class $\hat{c}$
Ensure: r is the best rule for given training data T
  $r := INITIALIZERULE(T)$ {body: $\emptyset$, head: (fixed class c, minority/majority class c or best class c)}
  $T_{covered} := T$
  while $GETNEGATIVES(T_{covered}, \hat{c}) \neq \emptyset$ **do**
    for each possible condition **do**
      $r_{refined}.body \cup c$
      if $EVAL(r_{refined}, T) > EVAL(r, T)$ **then**
        $r = r_{refined}$
      end if
      $T_{covered} = GETCOVERED(r, T)$
    end for
  end while
  return best rule r

2.4.2. Rule Sets

The restricted expressiveness of single rules can be extended to arbitrary subsets of the instance space $\mathbb{I}$ by combining single rules to form a rule set as a classification model. The conjunctive rules of the set are interpreted as disjunctively covering the subsets of the instance space that they individually cover. I. e. an instance is covered by the rule set if it is covered by any rule. To add a rule to the set means to generalize the hypothesis the rule set describes, i. e. the rule set covers a larger or equal amount of instances.

Rule Set Hypothesis Representation

Conjunctive rules that form a rule set interpreted as a disjunction form statements of propositional logic in disjunctive normal form (DNF). Each rule condition or feature as a predicate can be true or false therefore it can be interpreted as a binary variable. Each rule is then a conjunction of theses binary variables and the rule set is a disjunction of these conjunctions, hence a DNF. Such a rule set classifier representation format as hypothesis description language is capable of describing any possible hypothesis of any given example space with arbitrarily increasing accuracy. But in cases where the instance space has to be divided into infinite many subsets due to a relevant correlation between attributes as described for a diagonal decision boundary the hypothesis cannot be accurately represented.

This hypothesis representation implies the following data transformations. A rule set classifier transforms a given instance representation into a predicate, i. e. binary representations of whether the instance belongs to explicitly described subsets of the instance space and then transforms those into predicate representations of intersections of these subsets performed by logical conjunctions and unites those intersections using logical disjunctions to new sets again represented as a predicate. This predicate is directly associated with the predicted class. This often makes the rule set hypothesis descriptive in exposing what subsets of the instance space are relevant for an instance to be of a certain class. These facts are not only exposed to the user in form of the representation of the rule set but also actually are relevant for any performed classification that this specific hypothesis models. Therefore the chosen kind of data transformation has not only an impact on the descriptiveness but also on the predictive performance of the classifier.

Decision List

A so called decision list can be used to define an order by which the rules of a rule set are applied to an instance. The first rule in the list that covers the instance makes the final prediction.

A rule that performs a binary classification takes a object representation as input and outputs either nothing, shown in the example in Figure 2.8. labeled as "No Class", the representation of the first of the two classes labeled "Positive Class" or of the second class labeled "Negative Class". The outputs of the rules that form a hidden layer are transformed into the final class representation of the decision-list classifier according to a function that outputs the output of that rule of the hidden layer that is the first one which doesn't output "No Class" as shown in Figure 2.8. This is the positive class if any rule covers the instance to be classified or the negative default class if the last rule was reached.

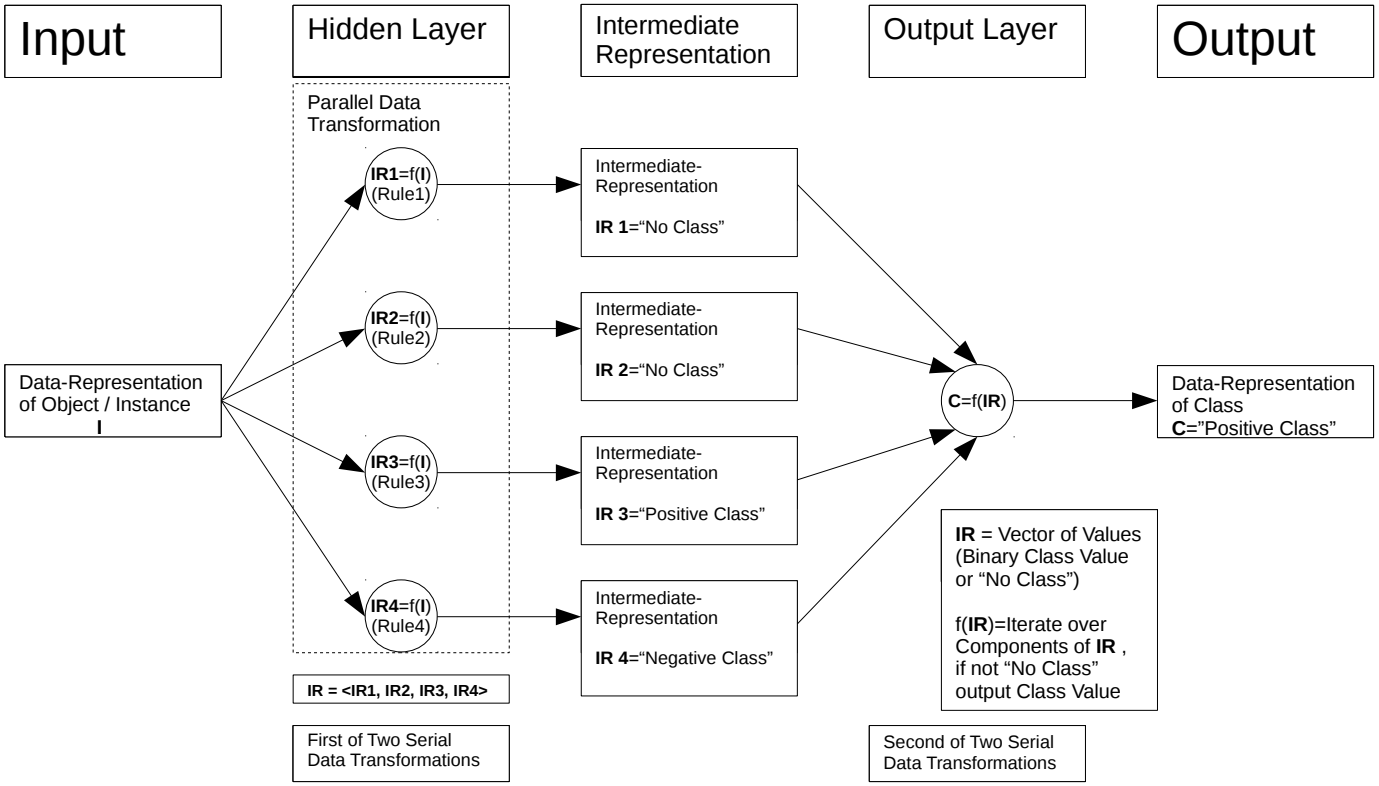

Figure 2.6.: a) Modularity of the Data Transformation in the Case of a Decision List

Hypothesis Search by Separate and Conquer Algorithm

After searching the hypothesis space by specializing the most general rule by consecutive refinements in the form of adding conjunctive conditions to single rules finding the best, i. e. most complete and consistent rule the search for a more optimal hypothesis continues here by generalizing the hypothesis again by adding a

further rule to the decision list. The separate and conquer algorithm, SeCo, restricts this search to that part of the instance space corresponding to the examples from the training data that were not covered by previous rules. Nevertheless, the rules induced from these examples can still cover any part of the instance space.

Algorithm 2 Basic Structure of Separate and Conquer Algorithm - SeCo (cf.Fürnkranz 1999).

Require: $T = P \cup N$ {required predicted class $\hat{c}$ splits the training examples into T positive examples with $\hat{c} = c$ and negative examples with $\hat{c} \neq c$}, evaluation function $eval$, class $\hat{c}$
Ensure: rules in R cover all examples in training data T
  $R := \emptyset$
  $T_{uncovered} := T$
  while $T_{uncovered} \neq \emptyset$ **do**
    $r := FindBestRule(T_{uncovered})$
    $R := R \cup r$
  end while
  return The learned rule set R

2.4.3. Single Disjunctive Rule Classifier

It is possible to form single rules using disjunctions instead of conjunctions of conditions as rule body. An empty rule that initially covers no instance can be generalized by adding alternative conditions for which instances will be covered by the rule. As in the case of single conjunctive rules, disjunctive rules by themselves are not a hypothesis description language of full expressiveness. But disjunctive rules that form a rule set interpreted as a conjunction could be interpreted as a expression in propositional logic in conjunctive normal form (CNF). Such a rule set as a hypothesis description language is in its expressiveness equivalent to the mentioned conventional rule set in DNF but poses different challenges for the hypothesis search.

2.4.4. Ensemble Methods

When different classifiers are combined to perform the classification task they form an ensemble. The hypothesis description language of the different classifiers is extended by the representation of the defined combination method. The hypothesis search differs by the fact that each component classifier is not optimized for the best individual performance but the different classifiers must also have diverse classification behavior in order to perform better collectively(Bian and Chen 2019). Similarly to neural networks that create diversity between node functions especially by using different initial weights in ensemble learning different methods such as manipulating instances or features among others are used to generate diverse classifiers. (Bian and Chen 2019).

Stacking

The ensemble method stacking, see (Wolpert 1992), aims at optimizing the individual performance of different classifiers by learning a new meta-classifier that is trained on the classifications of the different classifiers.

While the so called level-0 or base classifiers are trained on the training data that is a subset of the example space the meta-classifier or level-1 classifier is trained on meta-examples M_E of a meta-instance set $\mathbb{M}$.

$$M_E := \langle \hat{c}_1, ..., \hat{c}_m, c \rangle \in \mathbb{M}, with\ \mathbb{M} = C \times ... \times C \times C \quad \text{(Training Meta-Example)} \tag{2.16}$$

The same is true for the test instances M_I.

$$M_I := \langle \hat{c}_1, ..., \hat{c}_m \rangle \in \mathbb{M}, with\ \mathbb{M} = C \times ... \times C \quad \text{(Test Meta-Instance)} \tag{2.17}$$

An example is shown in Figure 2.2 b). The meta-classification is the second step in a series of two data transformations.

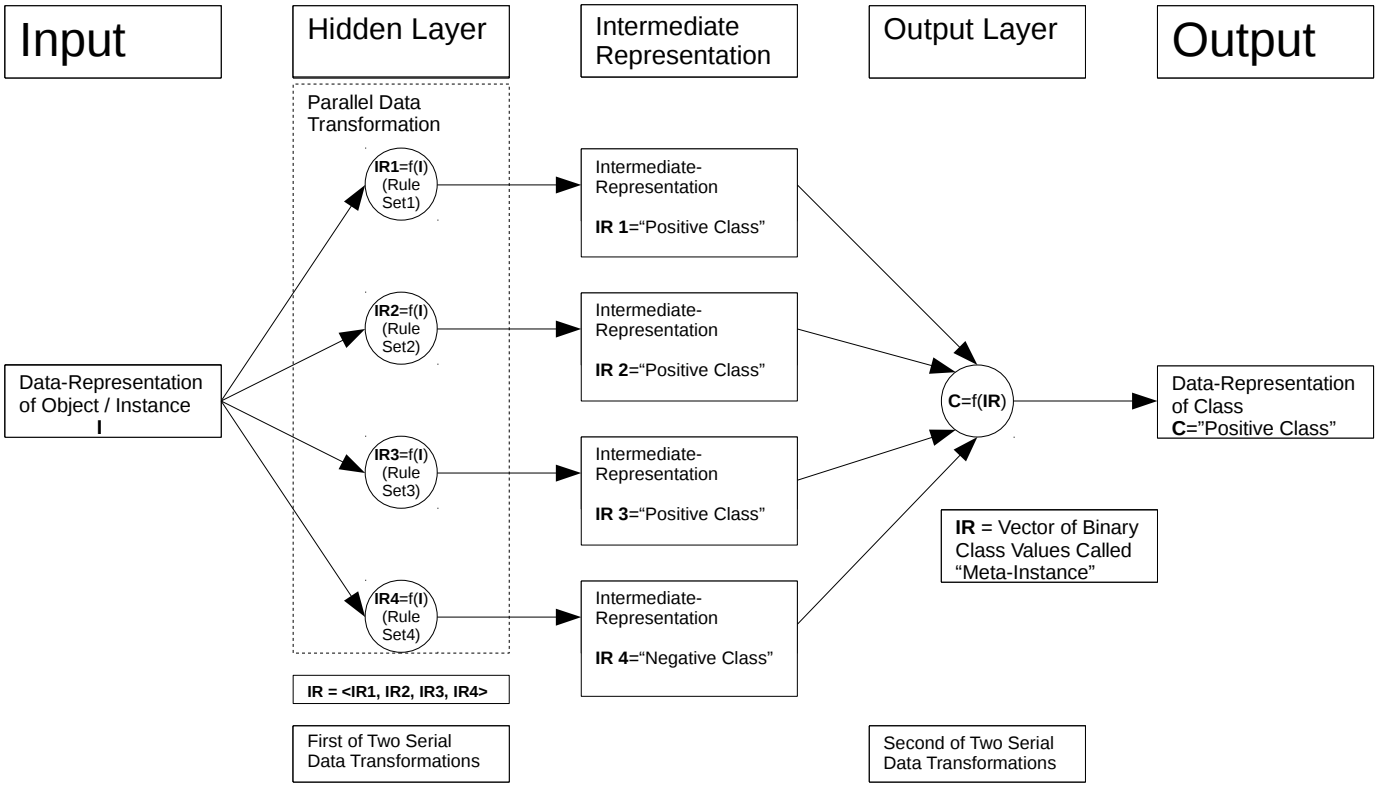

Figure 2.7.: b) in the Case of an Ensemble of Classifiers (Stacking)

Bagging

Bagging is an ensemble method where the diversity between classifiers is achieved by training them with different bootstrap samples created by sampling the instance set with replacement (Breiman 1996). This

method can be combined with stacking by training a meta classifier from base classifiers that were trained with different bootstrap samples. This bag-stacking method has shown to be successful by (Ting and Witten 1997).

3. Related Work

In this chapter the characteristic advantages of the fact that a classifier has a network structure are discussed in the context of relevant research. Afterwards, research on the role and the kind of necessary diversity of classifiers in classifier ensembles is investigated.

3.1. Purpose of Network Structure in ANN

Since neural networks, especially deep neural networks, have proven to be successful prediction models, including classification models, there has been an ongoing research into the question of improvement of the descriptiveness of these models. One obstacle on the way towards higher descriptiveness is the lack of understanding how a hypothesis represented as a neural network derives the predicted class from useful pattern in the data.

Descriptiveness of a classification procedure stems from dividing, i. e. modularizing the procedure into meaningful steps, so that these steps describe how the final classification is derived from useful pattern in the data.

If the classification is only memorized in a look up table for example there is no description. In the usual case where the classes are only known for a subset of the instance space such a memory classifier wouldn't generalize. If the classes were known for the complete instance space, memory is sufficient for a classifier because generalization is unnecessary. But also, there would be no description why a certain instance belongs to a certain class.

This means if a neural network does generalize, indicated for example, by a classification accuracy for a hold out test set above chance, then there must be a derivation of the predicted class from patterns in the data that equally allow the derivation of a specific class for training instances with this pattern and test instances with the same pattern. One possibility is that this derivation takes solely place in the training phase where the hypothesis is searched by using a form of gradient decent and the result (individually for all possible instances in instance space) is then simply memorized in the set of network parameters, the weights. The role of the modularized structure, layers of neurons, could be reduced to providing the necessary memory capacity.

3.2. Memorization in Neural Networks

Neural networks are generally capable of memorizing the training data. They can even memorize random data (Zhang et al. 2017), i. e. data that cannot be grouped by any pattern. In this case the network can only implement its classification function by somehow individually memorizing the output for each individual input. The conclusion was drawn by (Arpit et al. 2017), that neural networks behave differently on random data and

real data. But the difference is mainly found in the search by stochastic gradient descent. Simple pattern are found in earlier epochs, then with increasing number of epochs the hypothesis complexity increases(Arpit et al. 2017). The measure used for hypothesis complexity is based on decision boundary density. This suggests that when the hypothesis complexity stabilizes at a certain amount of epochs and the decision boundary density is sufficient all the simple pattern in the data that would be descriptive are covered up in a sense they are lost in the search procedure that only aims at memorizing the final complex and not descriptive classification hypothesis. It is equivalent to just memorizing the decision boundary itself which is sufficient to classify any instance from the instance space.

3.3. Generalization by Memorization

In this context, the question whether generalization by memorization is possible is explored by (Chatterjee 2018). That it is possible in some sense is shown experimentally by evaluating a randomly support-limited network of look-up tables that each perform a binary classification on the MNIST-Dataset. For the support k inputs are randomly selected. One conclusion of this work is, that neural networks might generalize by memorization as well.

First, it has to be clarified what memorization means in the suggested network of look-up tables. Each look-up table memorizes an output value for each possible input. No common operator is applied to the input data to calculate the output as opposed to a neuron where the same weight vector is applied to all inputs performing the same linear transformation and applying the same non-linear activation operator.

But due to the support limitation each look-up table memorizes the same output value for the set of all instances that have the k limited attribute values in common. In real data that can be generalized there should be subsets of instances of each class that can be differentiated by subsets of their attribute values. In the network of look-up tables subsets of fixed attribute dimension k are randomly formed in the first layer. These subsets might not be optimal in their relevance for classification. Other than being completely irrelevant, the subsets also could only have some irrelevant attribute values. The look-up tables of the succeeding layers can optimize that by learning a function that models the union of these subsets. If two different look-up tables of the first layer predict the same class for two of their respective subsets and a look-up table in the second layer that is only supported by these two look-up tables predicts the same class for either or both these two predictions then it effectively predicts this class for each instance that belongs to either of the two subsets, i. e. to their union. The look-up table of the second layer implements therefore a logical or-function effectively on classified subsets of instances.

Finally, there could be additional attribute values that make the memorized prediction more precise. In that case a logical and-function could be learned for a look-up table in the second layer that only predicts the class if the not so precise look-up table in the first layer predicts it and a look-up table with some of the same attribute inputs and the missing relevant attribute input also predicts this class. This is the intersection of the respective subsets represented by the two look-up tables in the first layer.

It follows that the network of look-up tables is effectively trained by optimizing look-up table vectors as parameters that determine logical operators, i.e. AND and OR operators and negated versions thereof, for the nodes in the classification network.

This is not different from rule learning. If the value vector of an instance is applied to a binary rule the rule ignores all attribute values that are not in the rule body and performs a logical operation usually a conjunction on the matched subset of the values of the instance.

3.4. Diversity in Ensemble Methods

The heuristic based knowledge that increased diversity in ensembles can increase the accuracy has recently been investigated more systematically for example by (Bian and Chen 2019). Using a diversity measure based on a bias-variance-covariance decomposition method adapted from regression to classification tasks, it was found that for a few specific ranges inreased diversity related to a reduction of the generalization error. This is a step towards a better understanding of whether and how diversity relates to the accuracy of the ensemble. This result indicates at least that the extent of diversity relates to a better accuracy when predicting unseen instances. This is by some researches denied on the basis of contradicting experimental results, see (Kuncheva and Whitaker 2003).

4. Network of Single Rules

In this chapter, a training algorithm for a classifier based on a deep network of single rules is proposed. The basic network structure is shown in Figure 4.1.

4.1. Data Transformation Structure

This classifier performs a classification as a network of single rules in the following manner. Single rules form the nodes of the network. Each rule transforms its input data into a class prediction for the overall target.

To create depth the network is composed of multiple layers of single rules.

For each rule of the first layer of the network the input is an instance representation I in the form of Equation 2.5.

Rules of succeeding layers are meta classifiers as defined for the ensemble method stacking as described in Section 2.4.4. Each rule of a succeeding layer (level-1 models) classifies a meta instance M_I, see Equation 2.17, that is generated from the classifications of m selected rules of its preceding layer (level-0 models) as illustrated in Figure 4.1.

This implies that stacking is used here layer-wise recursively each layer deepening the recursion. So, the rules in the last layer are the final meta classifiers because no further meta classifier is applied to their predictions.

4.2. Hypothesis Representation

4.2.1. Recursive Interpretation of Rule Set

As with any hypothesis represented as a set of rules it must be defined which of the rules of the set will be chosen to make the prediction of the rule set classifier. This choice falls naturally to a rule in the last layer as the final meta classifier. To achieve the singularity of a final meta classifier rule the last layer of the network is restricted to one rule.

The most common interpretation of a set of rules as a disjunction of the conjunctive features of each rule of the set however, must be replaced by a different interpretation.

Each rule in the first layer of the network models a conjunction of features. These rules of the first layer can in turn be predecessors of a rule in the second layer of the network.

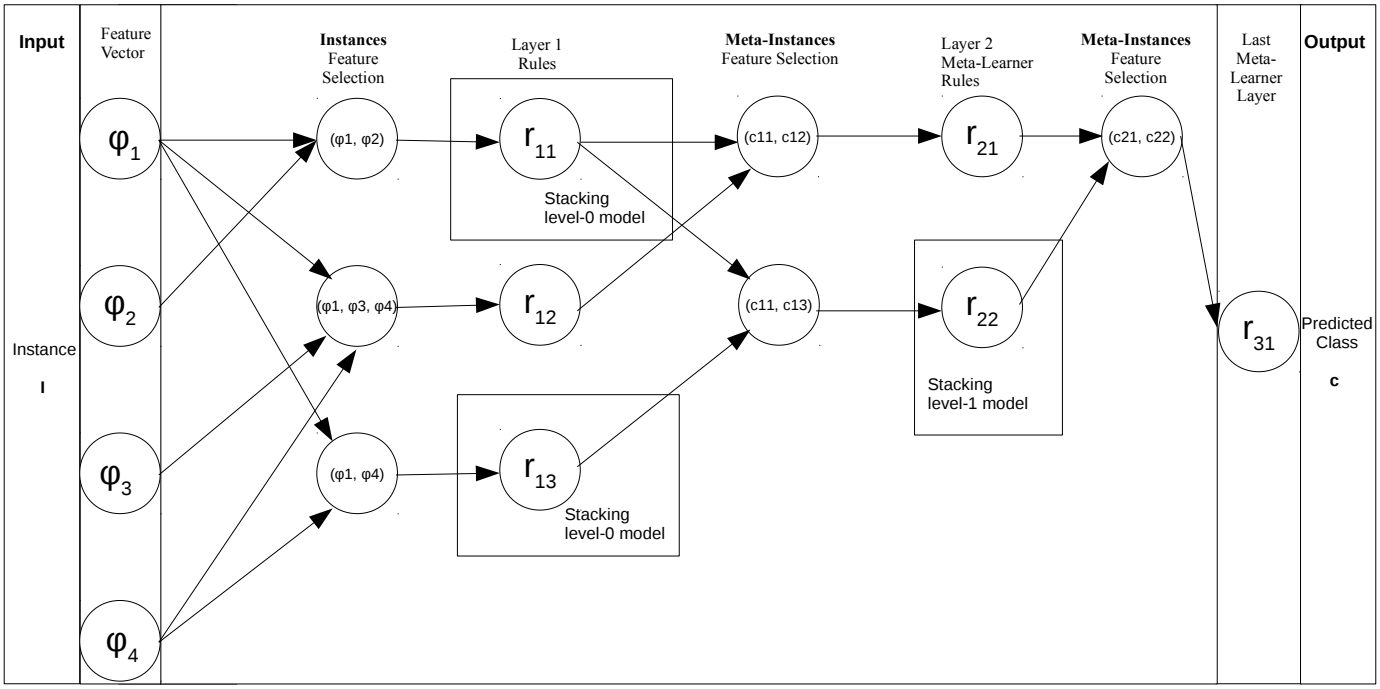

Figure 4.1.: A Deep Rule-Based Network with three Rule Layers Exposing Rules as Level-0 and Level-1 Model of Stacking

Such a second layer rule models the disjunction of its predecessor rules from the first layer. I. e. each rule in the second layer models a rule set classifier that represents the hypothesis in DNF. In this case the respective disjunctive rule is just an unusual notation of a rule set.

The third layer consists of conjunctive rules again. Each of those rules represent a hypothesis as a conjunction of DNF's. Table 4.1 shows example hypotheses for networks with 2 and 4 layers.

Set of Example Features = $\{\phi_1 = x < 8, \phi_2 = x > 2, \neg\phi_2 = x \leq 2, \phi_3 = y > 5\}$		
Number of Layers	Set of Rules	Hypothesis Representation
2	$\{r_{11} = \phi_1 \wedge \phi_2, r_{12} = \neg\phi_2 \wedge \phi_3, r_{13} = \phi_3, r_{21} = \hat{c}_{11} \vee \hat{c}_{12}\}$	$r_{21} = \phi_1 \wedge \phi_2 \vee \neg\phi_2 \wedge \phi_3$
4	$\{r_{11} = \phi_1 \wedge \phi_2, r_{12} = \neg\phi_2 \wedge \phi_3, r_{13} = \phi_3, r_{21} = \hat{c}_{11} \vee \hat{c}_{12}, r_{22} = \hat{c}_{13}, r_{32} = \hat{c}_{21}, r_{34} = \hat{c}_{22}, r_{41} = \hat{c}_{32} \vee \hat{c}_{34}\}$	$r_{41} = (\phi_1 \wedge \phi_2 \vee \neg\phi_2 \wedge \phi_3) \vee \phi_3$

Table 4.1.: Hypothesis Examples of Rule Network

For succeeding layers the rule type continues to alternate between conjunctive and disjunctive rules.

4.2.2. Rule Characteristics of Model

The rules that are induced for the network of rules hypothesis can play different roles in the hypothesis. Only a subset of the induced rules is used for the hypothesis that the entire network represents. If the prediction of a rule in one layer is not used in a condition by any rule in the following layer then this rule is not used at all. The subset of used rules is further divided into copy rules, feature rules and birth rules.

Copy rules can occur anywhere but the first layer. They have only one condition, i. e. created from the prediction of only one rule in the preceding layer. They predict the same class as this preceding rule under the same condition if the condition is interpreted by recursive substitution by the conditions of the preceding layers until the conditions are substituted by the original features created for the rules in the first layer.

Feature rules can also occur anywhere but the first layer. They have more than one condition and are therefore not only copies of a rule in the preceding layer but a rule with new features, features created as the conjunction or disjunction of previously created features.

Finally, it is possible that a rule is induced in some layer that does not use any prediction of the rules in the preceding layer. It has an empty rule body, i. e. no condition. See Figure 4.2.

4.3. Hypothesis Search

During the search the hypothesis is represented by a growing set of rules. All rules of the set are induced one by one. After one layer is complete the search proceeds by inducing the first rule of the succeeding layer. Once the single rule of the last layer is induced the hypothesis is found.

The search begins with the induction of a first single rule for the first layer of the network. As a rule of the first layer it is trained with training examples E as defined in Equation 2.4

The induction of this rule is performed by the procedure $FindBestRule$ as shown in Algorithm 1. As an evaluation function $F - Measure$ is used, see Equation 2.15.

The rule is initialized as most general and is specialized with each refinement by adding conjunctive features as conditions. The precision is improved avoiding some loss of recall. The trade-off between precision and recall is defined by the $F - Measure$ parameter β as explained in Section 2.15.

Since each rule in the second layer is induced as a meta classifier for the rules of the first layer it is required that the rules of the first layer are diverse classifiers.

Single rules as base classifiers can be diversified by optimizing them for different types of instances, i. e. subsets of the instance space.

4.3.1. Separate and Conquer Algorithm

One way to create diverse rules is to use different subsets of the training instances for the induction of each rule. The SeCo Algorithm as shown in Algorithm 2 produces a set of rules where each subsequent rule has a potentially lower recall relative to all training instances but a potentially higher precision for the instances it was trained on and a potentially lower precision relative to all instances. So within the trade off between diversity and performance the SeCo Algorithm produces rules that are more diverse but less performant if

applied to any possible instance. Typically, good performance is achieved by applying the rules only if an instance was not covered by a previous rule in a decision list. I. e. if the learned rule set would be used as a decision list, then each instance would be classified by the rule that is most likely most qualified to make a prediction for that kind of instance.

The step of instance separation can be based on different covering strategies.

TP and FP count as covered

Usually, a rule is said to cover all instances that the rule predicts to be positive. This includes TP and FP. So $uncovered$ in Algorihm 2 can be implemented according to this definition.

Only TP count as covered

One way to use more instances for the training of subsequent rules is to only treat TP as covered.

4.3.2. Weighted Covering

Another way to allow a higher recall for subsequent rules is to define the coverage by weight. Weighted Covering can be used to induce rules that are diverse and each have enough recall and are likely to be still meaningful if applied to instances of the complete instance space (Fürnkranz, Gamberger, and Lavrac 2012).

Weighted covering as an alternative strategy shifts the focus from diversity towards better performance of single rules if applied to any subset of the instance space, see (**Witten2017**). This is because succeding rules are still trained on training instances that have been covered by preceding rules but their impact on the rule induction is reduced by a weight factor. This factor is chosen to evolve according to the sequence $(1, 1/2, 1/3, 1/4, , ,)$ as defined by Equation 4.1.

$$NextWeight = \frac{1}{\frac{1}{PreviousWeight} + 1} \qquad \text{(Weight of Next Instance for Weighted Covering)} \tag{4.1}$$

In either case, the instance space is divided into subsets by determining subsets for which previously induced rules did not perform too well.

4.3.3. Random Attribute Subset Selection

Another way to create diverse rules is to divide the instance space into subsets by attribute selection. If a rule induction is prevented from using certain attributes to distingusih instances then all instances that can be classified independent from such attribute values form the subset for which the rule performs well. For the subset of the instance space that consists of the instances for whose classification the attribute value is relevant the rule performs worse.

In contrast to the separation approaches in the case of attribute selection it cannot be determined in advance which parts of the instance space need closer attention because even after evaluating the performance of a rule it cannot be determined to which subset of the instance space this performance should be attributed.

4.3.4. Bagging

The number of diverse rules per layer can be increased by the application of bagging as described in Section 2.4.4. Since the separate and conquer algorithm and the weighted covering algorithm are terminated at some point these algorithms can be repeated with the next bootstrap sample and this way produce more diverse rules for that layer. Furthermore, the degree of diversity of rules in one lyer can also be increased by applying different boot strap samples to each single rule of that layer.

4.3.5. Beyond the First Layer

The rules that are induced beyond the first layer as meta classifiers are effectively copies or logical combinations, referred to as feature rules. Birth rules should be avoided in the hypothesis search by improving the quality of copy and feature rules. As copies and combinations of the rules in the first layer and because these rules serve now as base classifiers for the succeding layer the described methods to generate diversity can be applied as well.

Additionally, the ratio between copy rules and feature rules might have an optimal balance. Copy rules preserve good results from previous layers but feature rules allow for exploration to eventually find better results by new combinations. For this reason a minimum number of two conditions for rules beyond the first layer can be enforced with a certain probability.

Figure 4.2.: Network of Rules - Used Rules are either Feature Rules (Red), Copy Rules (White) or Birth Rules (Green) - Red Lines Indicate Input of Feature Rules and Blue Lines are Input of First Layer Rules and Copy Rules

5. Evaluation

In this chapter different search strategies for a hypothesis that is represented as a network of rules as described in Chapter 4.3 are compared in terms of predictive performance with respect to small numeric and nominal binary datasets. The impact of increasing depth of the network is also considered. In addition, the rules are compared in terms of their descriptiveness of the learned hypothesis.

5.1. Experimental Setup

5.1.1. SeCo Framework

In order to evaluate different variations of the network of rules algorithm a modified version of the "SeCo-framework", see (Janssen and Zopf 2012), is employed. This framework implements the separate and conquer algorithm, see Algorithm 2, in a generic form to allow the use for many different variations of this algorithm. Since the network of rules algorithm is not a type of separate and conquer algorithm, but only incorporates it, the framework could not be used as designed. But the framework's generic nature for rule learning in general encouraged the integration of its functionality into the network of rules algorithm.

5.1.2. Basic Network Configuration

Class Selection

In order to ensure the availabilty of predictions of both classes to the rule induction procedure for a rule in a network layer that is not the first layer rules for both classes are induced. This is accomplished by alternating the predicted class starting with the minority class. The alternation rate depends on the type of dependency between rules in one layer. If there is no dependency then the predicted class is alternated rule by rule creating an equal amount of rules for each class. If the rules are created according to the SeCo or weighted covering algorithm the predicted class is not changed until the SeCo or weighted covering algorithm terminates and is repeated by a different bootstrap sample as described in Section 4.3.4.

Network Topology

The network has the highest number of rules, i. e. conjunctive rules, in the first layer to make as many features available to the network as possible. For the following layers the number of rules decreases by $3/4$, but the second layer is restricted to a maxiumum of 40 rules. Finally, as described in Section 4.2, there is only one rule in the last layer.

5.1.3. Hyperparameters

The subset of different strategies that are evaluated in this work can be specified by the cartesian product of a set of hyperparameters that determine the induction strategy for each single rule in the network. To search this strategy space for promissing strategy candidates some variations a tested. The variable parameters or hyperparameters that specify these variations are chosen randomly. This strategy is more efficient than to keep some parameters fixed and to only variate one variable parameter at a time (Bergstra and Bengio 2012).

5.1.4. Data Sets

As mentioned before, only binary classification tasks are considered. So the network is evaluated by training and testing it with datasets that have a class attribute with two values, a synthetic dataset referred to as 'synthetic' from (Rapp, Mencía, and Fürnkranz 2019) and two datasets from the UCI repository, see (Dua and Graff 2017), 'ionossphere' and 'vote'.

The dataset 'synthetic' shown in Figure 5.1 has two numerical attributes referred to as x and y and the two class values 'positive' and 'negative'. The values of the attributes x and y are real numbers in the domain [0, 10] and 1000 examples have been created for this dataset, 200 for class 'positive' and 800 for class 'negative'. This set is created twice using the same statistical distribution and the same ratio for the two classes. One of them is used for the training of a network of rules and the other one for testing it.

The 'ionossphere' dataset has 34 numerical attributes and the two class values 'good' and 'bad'. It consists of 351 examples for which the two class values are split approximately equally. Finally, the dataset 'vote' has 16 nominal attributes that can only take two different values 'yes' and 'no', the two class values are 'democrat' and 'republican'. There are 435 examples. Both data sets are sampled into one training and one test set using stratification so that the ratio between the two classes is preserved. The size of the training set and the test set is approximately equal.

5.2. Experimental Results

5.2.1. Accuracy by Statistical Hyperparameter Selection

The experimental results of the accuracy for the statistical hyperparameter selection are shown in Appendix A. For the three datasets the accuracies are mapped to a accuracy class that reflects the performance relative to the default rule for the predicted class. The network can perform well for many variations of the parameters.

5.2.2. Improvement by Depth

If the SeCo algorithm or weighted covering is applied to the rules in the first layer then this leads often to a hypothesis with high predictive performance that can hardly be increased by additional layers of rules, see Figures 5.2 - 5.4.

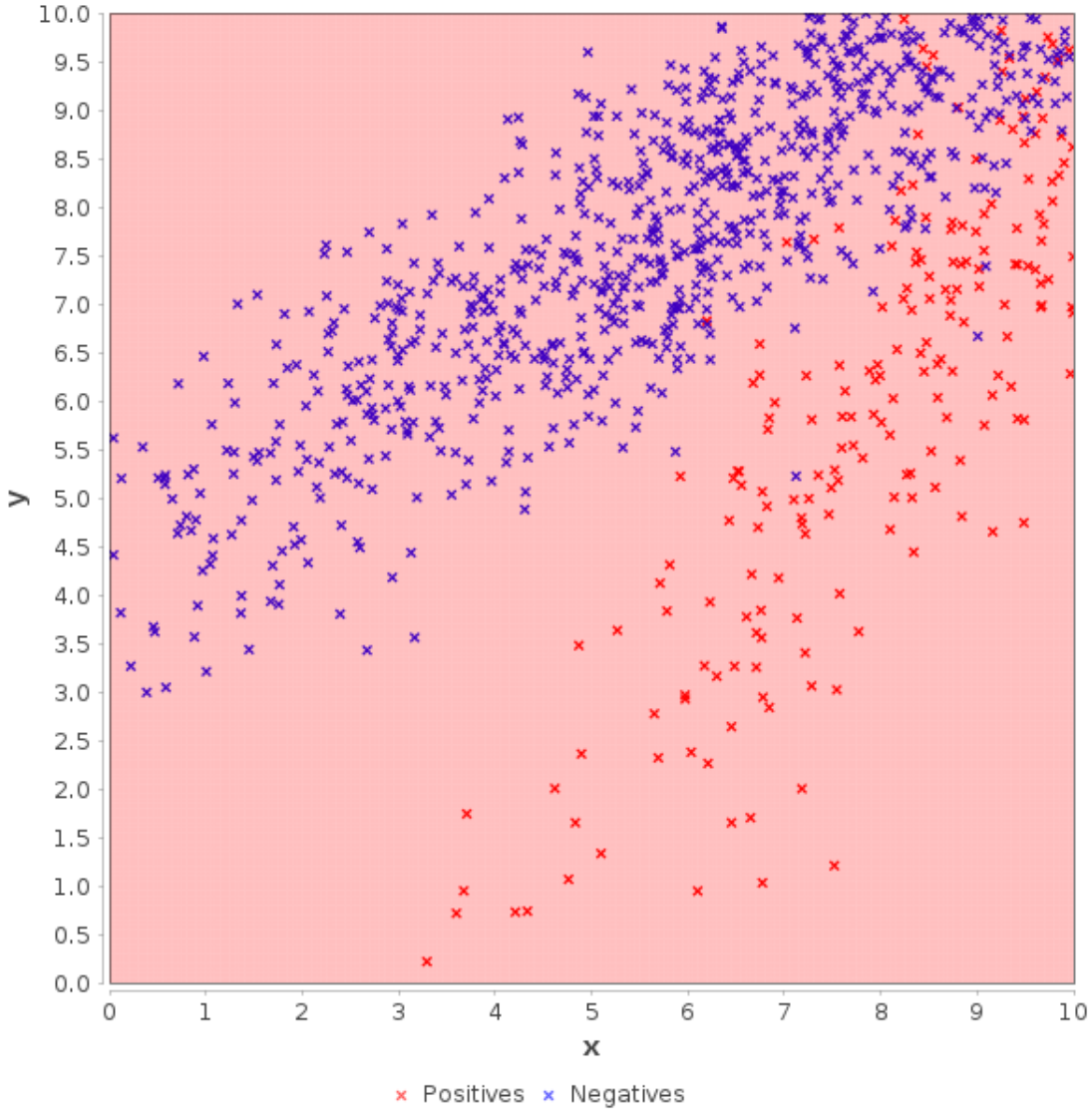

Figure 5.1.: Training Examples - 'Synthetic'

5.2.3. Rule Characteristics of Model

For many configurations more feature rules than copy rules are induced as shown in Figures 5.4 - 5.6. In configurations where the probability to enforce at least two conditions is high the accuracy is often still good as can be seen in the tables in the Appendix.

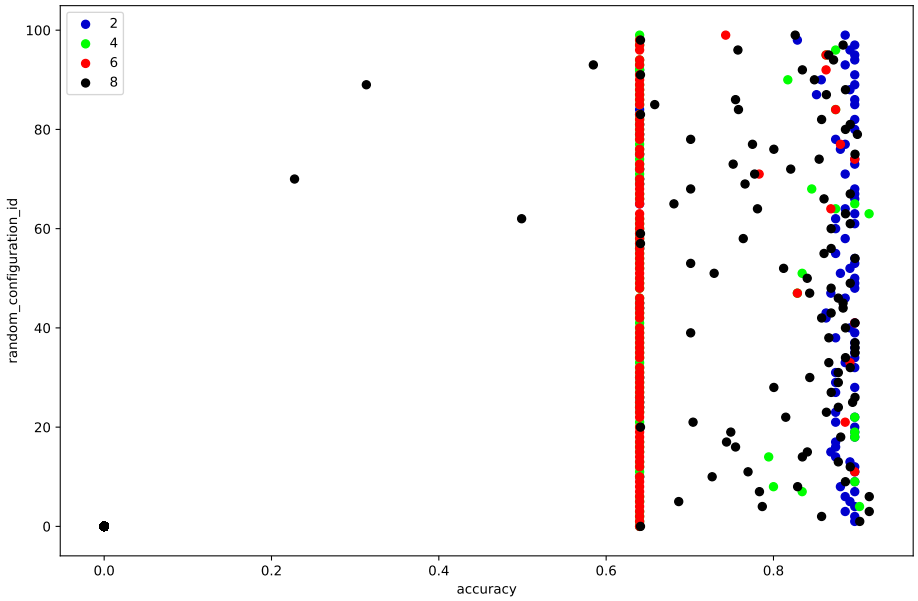

Figure 5.2.: Layerwise Accuracy - Ionossphere

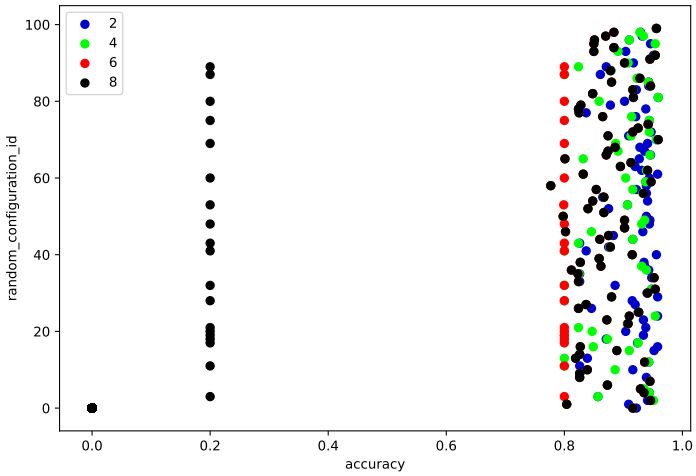

Figure 5.3.: Layerwise Accuracy - Synthetic

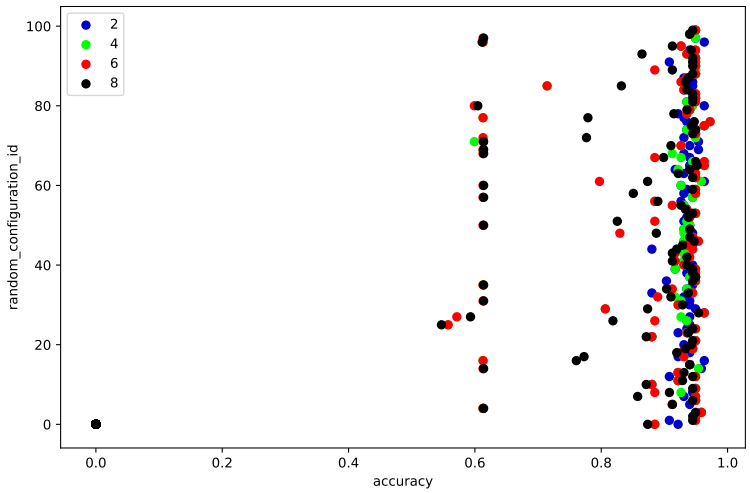

Figure 5.4.: Layerwise Accuracy - Vote

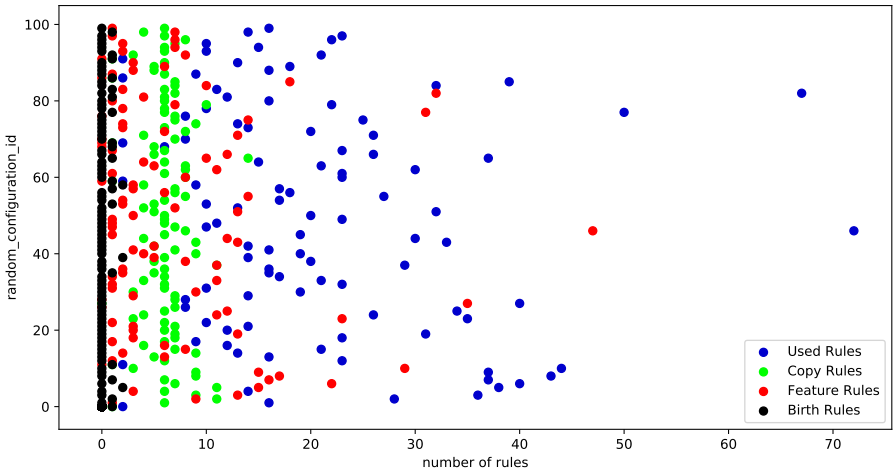

Figure 5.5.: Rule Characteristics of Model - Ionossphere

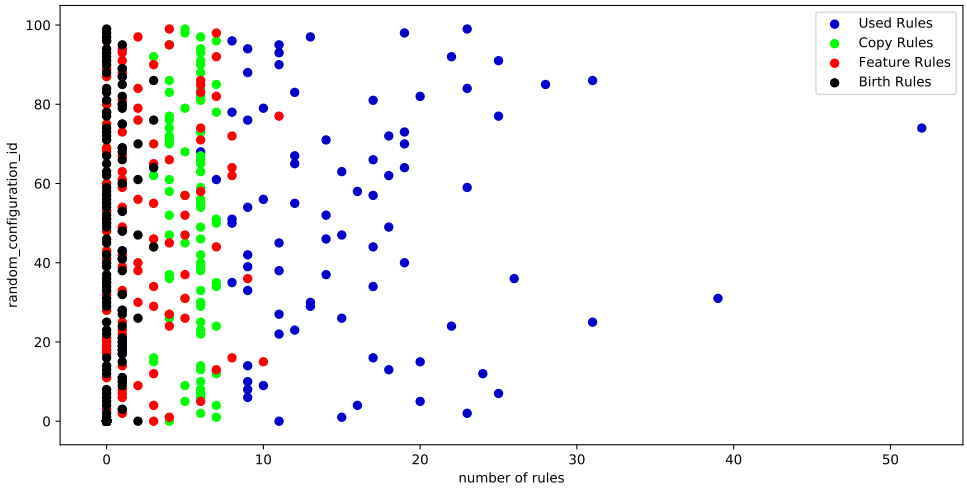

Figure 5.6.: Rule Characteristics of Model - Synthetic

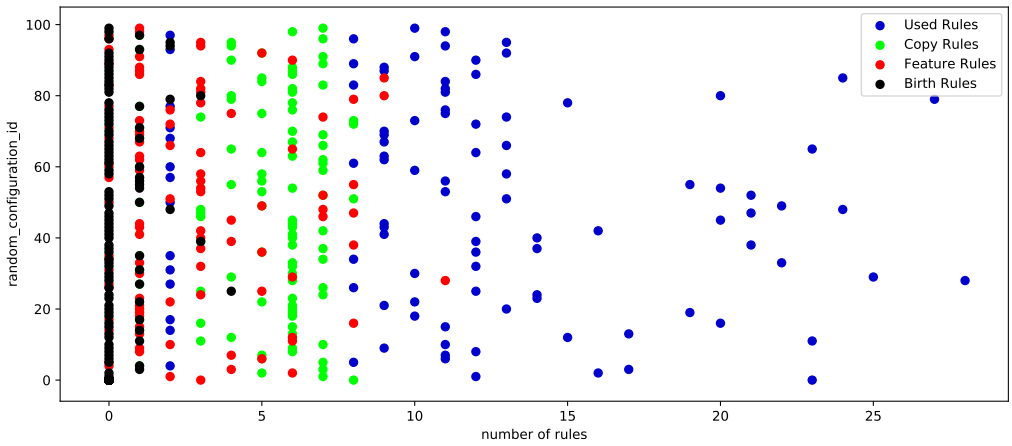

Figure 5.7.: Rule Characteristics of Model - Vote

6. Conclusion

The proposed network of rules can in many configurations make predictions comparable to other rule set models where rules are induced using the SeCo akgorithm or weighted covering. The first two layer operate comparable to those methods. An improvement of accuracy by the production of suitable intermediate features in deeper layers could not systematically be generated. It only happened coincidentally and does not exceed the accuracy of better configuration where this accuracy is already obtained by the first two layers.

For future work the algorithm that induces the rules of the network could be computationally optimized to allow experiments with greater number of nodes and layers to investigate the explorative impact of randoms subset selection to find more suitable feature rules in deeper layers.

Another way to force the network to explore more different feature rules could be to lower the accuracy of rules in the first layers. Finally, this strategy could be supplemented by the generation of relational features by inducing additional association rules that find pattern independent from the class that could prove relevant for the class in combination with other rules.

Bibliography

Arpit, Devansh et al. (2017). “A Closer Look at Memorization in Deep Networks”. In: *CoRR* abs/1706.05394. arXiv: 1706.05394. URL: http://arxiv.org/abs/1706.05394.

Bergstra, James and Yoshua Bengio (2012). “Random Search for Hyper-Parameter Optimization”. In: *J. Mach. Learn. Res.* 13, pp. 281–305. URL: http://dl.acm.org/citation.cfm?id=2188395.

Bian, Yijun and Huanhuan Chen (Oct. 30, 2019). “When does Diversity Help Generalization in Classification Ensembles?” In: arXiv: 1910.13631v1 [cs.LG].

Bishop, Christopher M. (2007). *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer. ISBN: 9780387310732. URL: http://www.worldcat.org/oclc/71008143.

Breiman, Leo (1996). “Bagging predictors”. In: 24, pp. 123–140. ISSN: 0885-6125. DOI: 10.1007/bf00058655.

Chatterjee, Satrajit (2018). “Learning and Memorization”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=S1XFzYyPM.

Dua, Dheeru and Casey Graff (2017). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Fürnkranz, Johannes (1999). “Separate-and-Conquer Rule Learning”. In: *Artif. Intell. Rev.* 13.1, pp. 3–54.

Fürnkranz, Johannes, Dragan Gamberger, and Nada Lavrac (2012). *Foundations of Rule Learning*. Cognitive Technologies. Springer. ISBN: 978-3-540-75196-0. DOI: 10.1007/978-3-540-75197-7.

Janssen, Frederik and Markus Zopf (2012). “The SeCo-Framework for Rule Learning”. In: *Proceedings of the German Workshop on Lernen, Wissen, Adaptivität - LWA2012*. Dortmund, Germany.

Kuncheva, Ludmila I. and Christopher J. Whitaker (2003). “Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy”. In: *Mach. Learn.* 51.2, pp. 181–207. DOI: 10.1023/A:1022859003006.

Mitchell, Tom M. (1982). “Generalization as Search”. In: *Artif. Intell.* 18.2, pp. 203–226. DOI: 10.1016/0004-3702(82)90040-6.

Murphy, Kevin P. (2012). *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press. ISBN: 0262018020.

Rapp, Michael, Eneldo Loza Mencía, and Johannes Fürnkranz (2019). *Simplifying Random Forests: On the Trade-off between Interpretability and Accuracy*. arXiv: 1911.04393 [cs.LG].

Ting, Kai Ming and Ian H. Witten (1997). “Stacking Bagged and Dagged Models”. In: *Proceedings of the Fourteenth International Conference on Machine Learning (ICML 1997), Nashville, Tennessee, USA, July 8-12, 1997*. Ed. by Douglas H. Fisher. Morgan Kaufmann, pp. 367–375.

Wolpert, David H. (1992). “Stacked generalization”. In: 5, pp. 241–259. ISSN: 0893-6080. DOI: 10.1016/s0893-6080(05)80023-1.

Zhang, Chiyuan et al. (2017). “Understanding deep learning requires rethinking generalization”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=Sy8gdB9xx.

Thesis Statement pursuant to §22 paragraph 7 and §23 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Daniel Jung, have written the submitted thesis independently pursuant to §22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form. I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once. In the submitted thesis the written copies and the electronic version for archiving are pursuant to §23 paragraph 7 of APB identical in content.

Darmstadt, June 30, 2020

_____

(Daniel Jung)

A. Appendix

Table A.1.: Accuracy for Dataset 'Synthetic' Part 1

Beta Con	Beta Dis	secoFirstLayer	secoAll	baggingFirstLayer	baggingAll	accuracy	accuracyClass
0.705	0.875	+	+	+	-	0.959	VeryGood
0.345	1.25	+	-	+	+	0.956	VeryGood
0.344	0.311	+	-	-	+	0.954	VeryGood
0.521	0.386	+	-	+	-	0.954	VeryGood
0.404	0.052	+	+	+	-	0.952	VeryGood
0.141	0.375	+	-	-	+	0.947	Good
0.194	0.945	+	+	-	+	0.946	Good
0.471	0.154	+	-	-	+	0.946	Good
0.056	0.087	+	+	+	+	0.945	Good
0.065	0.273	+	+	+	+	0.945	Good
0.242	0.47	+	-	-	-	0.942	Good
0.226	0.072	-	-	-	-	0.941	Good
0.678	0.333	-	-	+	-	0.941	Good
0.074	0.171	+	+	+	-	0.936	Good
0.372	0.162	-	-	-	+	0.935	Good
1.167	0.279	-	-	+	+	0.934	Good
0.848	1.062	-	-	+	-	0.929	Good
0.033	0.084	+	+	+	-	0.928	Good
0.032	0.038	+	+	+	+	0.926	Good
0.072	0.132	+	+	-	-	0.925	Good
0.223	0.068	+	+	+	-	0.917	Good
1.12	1.659	-	+	-	-	0.916	Good
0.765	0.052	-	-	-	+	0.916	Good
1.145	0.048	-	-	-	-	0.916	Good
0.403	0.04	+	-	+	-	0.915	Good
0.999	0.767	-	+	+	+	0.913	Good
0.804	0.16	+	+	+	-	0.91	Good
0.111	0.165	-	-	-	-	0.908	Good
0.148	0.183	+	+	+	-	0.902	Good
0.065	0.052	+	-	+	-	0.902	Good
0.445	0.053	-	-	+	-	0.902	Good
0.04	0.152	+	-	-	-	0.895	Bad
0.394	0.087	+	+	+	+	0.889	Bad
1.188	0.252	-	+	+	+	0.886	Bad
2.273	0.357	-	-	+	-	0.884	Bad
0.931	0.204	-	-	+	-	0.884	Bad
0.269	0.07	+	-	+	+	0.88	Bad
0.342	0.499	-	+	-	-	0.88	Bad
0.221	0.077	-	-	+	+	0.878	Bad
0.273	0.215	-	+	+	-	0.878	Bad
1.718	0.583	-	-	+	-	0.875	Bad
0.133	0.523	+	-	-	+	0.874	Bad
1.654	0.355	+	-	-	-	0.874	Bad
2.177	0.039	-	-	-	-	0.873	Bad
0.09	0.807	+	-	-	+	0.872	Bad

Table A.2.: Accuracy for Dataset 'Synthetic' Part 2

rssAll	weightedFirstLayer	weightedAll	rssK	tpCovering	enforceTwoCond	accuracy	accuracyClass
-	-	-	-1	+	0.511	0.959	VeryGood
-	-	+	-1	+	0.11	0.956	VeryGood
-	-	+	-1	-	0.178	0.954	VeryGood
-	-	+	-1	+	0.716	0.954	VeryGood
+	-	-	8	-	0.077	0.952	VeryGood
-	-	+	-1	+	0.703	0.947	Good
-	-	-	-1	-	0.427	0.946	Good
-	-	+	-1	+	0.184	0.946	Good
-	-	-	-1	-	0.31	0.945	Good
-	-	-	-1	-	0.954	0.945	Good
-	-	+	-1	-	0.836	0.942	Good
-	+	+	-1	-	0.276	0.941	Good
-	+	+	-1	+	0.885	0.941	Good
-	-	-	-1	-	0.637	0.936	Good
-	+	+	-1	+	0.367	0.935	Good
-	+	+	-1	+	0.321	0.934	Good
+	+	+	8	+	0.051	0.929	Good
-	-	-	-1	+	0.794	0.928	Good
-	-	-	-1	-	0.68	0.926	Good
-	-	-	-1	+	0.413	0.925	Good
-	-	-	-1	-	0.258	0.917	Good
+	+	-	7	+	0.971	0.916	Good
+	+	+	8	+	0.806	0.916	Good
-	+	+	-1	+	0.748	0.916	Good
-	-	+	-1	+	0.081	0.915	Good
+	+	-	9	+	0.769	0.913	Good
-	-	-	-1	+	0.385	0.91	Good
-	+	+	-1	-	0.422	0.908	Good
+	-	-	6	+	0.707	0.902	Good
-	-	+	-1	+	0.161	0.902	Good
+	+	+	9	-	0.797	0.902	Good
+	-	+	8	+	0.443	0.895	Bad
+	-	-	8	-	0.93	0.889	Bad
+	+	-	1	-	0.907	0.886	Bad
-	+	+	-1	-	0.284	0.884	Bad
+	+	+	4	+	0.549	0.884	Bad
+	-	+	8	+	0.757	0.88	Bad
-	+	-	-1	-	0.396	0.88	Bad
-	+	+	-1	-	0.156	0.878	Bad
-	+	-	-1	+	0.034	0.878	Bad
-	+	+	-1	+	0.769	0.875	Bad
+	-	+	5	+	0.436	0.874	Bad
+	-	+	3	-	0.674	0.874	Bad
-	+	+	-1	-	0.175	0.873	Bad
+	-	+	5	+	0.502	0.872	Bad

Table A.3.: Accuracy for Dataset 'Ionossphere Part 1'

Beta Con	Beta Dis	secoFirstLayer	secoAll	baggingFirstLayer	baggingAll	accuracy	accuracyClass
0.506	0.164	-	-	+	+	0.914	VeryGood
0.621	0.156	-	+	+	+	0.914	VeryGood
0.15	0.14	-	-	-	+	0.903	Good
0.269	0.054	-	+	-	-	0.858	Good
0.072	0.037	+	-	+	+	0.829	Good
0.035	0.099	+	-	-	+	0.886	Good
0.171	0.711	+	+	-	-	0.892	Good
0.531	0.112	-	-	+	-	0.877	Good
0.081	1.068	-	+	+	-	0.835	Good
0.608	0.108	-	+	+	+	0.84	Good
0.229	0.157	+	-	-	-	0.88	Good
0.273	0.556	-	+	+	+	0.863	Good
0.771	0.054	+	-	-	-	0.878	Good
0.478	0.114	+	-	+	+	0.895	Good
2.335	0.045	+	-	+	-	0.897	Good
0.077	0.981	-	+	+	-	0.869	Good
0.09	0.495	-	+	+	-	0.877	Good
1.292	0.037	+	-	-	-	0.843	Good
0.325	0.286	-	-	-	-	0.877	Good
0.102	0.07	+	+	-	-	0.892	Good
0.034	0.047	+	-	+	-	0.866	Good
0.239	0.049	-	-	-	-	0.886	Good
0.069	0.036	-	+	-	-	0.897	Good
0.069	0.448	-	+	-	-	0.897	Good
0.503	0.211	-	+	-	-	0.897	Good
1.728	0.203	-	-	+	+	0.866	Good
0.992	1.159	-	+	-	+	0.886	Good
0.202	1.415	-	+	-	-	0.897	Good
0.802	0.292	-	-	+	-	0.858	Good
0.826	1.003	-	-	+	+	0.869	Good
0.848	0.631	+	-	+	+	0.883	Good
0.435	0.281	+	-	-	+	0.883	Good
0.629	0.048	-	+	+	+	0.877	Good
1.598	0.037	-	+	+	-	0.843	Good
0.16	0.692	-	-	-	-	0.869	Good
0.128	0.122	+	+	-	-	0.892	Good
0.098	0.06	-	-	-	+	0.84	Good
0.243	0.436	-	-	-	+	0.897	Good
0.28	0.852	-	+	+	-	0.86	Good
0.782	0.911	+	-	+	+	0.869	Good
0.032	0.326	-	+	+	-	0.869	Good
0.218	0.051	+	+	-	-	0.892	Good
0.122	0.59	+	-	-	-	0.886	Good
0.196	0.872	-	+	-	+	0.86	Good
0.075	0.635	+	+	-	-	0.892	Good
0.392	2.341	+	+	+	+	0.821	Good

Table A.4.: Accuracy for Dataset 'Ionossphere Part 2'

rssAll	weightedFirstLayer	weightedAll	rssK	tpCovering	enforceTwoCond	accuracy	accuracyClass
-	+	+	-1	-	0.276	0.914	VeryGood
-	+	-	-1	-	0.591	0.914	VeryGood
-	+	+	-1	+	0.097	0.903	Good
-	-	-	-1	-	0.332	0.9	Good
-	-	+	-1	+	0.104	0.897	Good
-	-	-	-1	+	0.727	0.897	Good
-	+	-	-1	+	0.146	0.897	Good
-	+	-	-1	-	0.269	0.897	Good
-	+	-	-1	-	0.033	0.897	Good
-	+	-	-1	+	0.247	0.897	Good
-	+	+	-1	-	0.362	0.897	Good
-	-	+	-1	+	0.661	0.895	Good
-	-	-	-1	-	0.608	0.892	Good
-	-	-	-1	-	0.188	0.892	Good
-	-	-	-1	-	0.642	0.892	Good
-	-	-	-1	-	0.505	0.892	Good
-	-	-	-1	+	0.565	0.892	Good
-	-	+	-1	+	0.747	0.892	Good
+	-	+	6	+	0.881	0.886	Good
-	+	+	-1	-	0.465	0.886	Good
+	+	-	9	+	0.082	0.886	Good
-	+	+	-1	-	0.054	0.886	Good
-	-	+	-1	+	0.664	0.886	Good
+	-	+	9	+	0.945	0.886	Good
-	-	+	-1	-	0.711	0.883	Good
-	-	+	-1	-	0.261	0.883	Good
-	-	+	-1	+	0.535	0.883	Good
+	-	+	9	+	0.262	0.88	Good
-	-	+	-1	-	0.802	0.878	Good
-	+	+	-1	-	0.578	0.877	Good
-	+	-	-1	+	0.018	0.877	Good
+	+	+	9	-	0.063	0.877	Good
-	+	-	-1	-	0.864	0.877	Good
-	-	+	-1	-	0.553	0.872	Good
-	+	-	-1	+	0.964	0.869	Good
-	-	+	-1	+	0.236	0.869	Good
+	+	+	5	+	0.385	0.869	Good
+	+	-	7	+	0.466	0.869	Good
+	+	+	4	-	0.511	0.869	Good
+	-	+	7	-	0.741	0.866	Good
-	+	+	-1	+	0.627	0.866	Good
+	+	+	7	-	0.335	0.866	Good
-	+	-	-1	+	0.795	0.863	Good
+	+	+	2	-	0.206	0.863	Good
-	+	-	-1	+	0.446	0.86	Good
+	+	-	7	+	0.994	0.86	Good

Table A.5.: Accuracy for Dataset 'Vote Part 1'

Beta Con	Beta Dis	secoFirstLayer	secoAll	baggingFirstLayer	baggingAll	accuracy	accuracyClass
1.763	0.774	-	-	-	-	0.945	VeryGood
1.234	0.3	-	+	+	-	0.945	VeryGood
2.008	0.039	+	+	-	-	0.949	VeryGood
0.077	0.057	-	-	-	+	0.913	VeryGood
0.605	0.701	-	-	+	+	0.945	VeryGood
0.173	0.202	+	-	-	+	0.908	VeryGood
2.478	2.237	-	-	-	+	0.945	VeryGood
0.051	1.442	+	+	+	-	0.929	VeryGood
1.59	0.305	-	-	-	-	0.945	VeryGood
0.059	0.536	+	-	+	+	0.931	VeryGood
0.047	0.312	-	+	-	+	0.94	VeryGood
0.077	1.808	-	-	-	-	0.92	VeryGood
0.142	0.698	+	-	+	-	0.933	VeryGood
0.135	0.191	+	-	+	-	0.943	VeryGood
1.179	0.775	-	-	+	-	0.945	VeryGood
0.031	0.337	+	-	+	-	0.938	VeryGood
0.339	0.53	-	-	-	-	0.945	VeryGood
2.935	1.029	+	-	-	+	0.954	VeryGood
0.249	0.655	+	-	-	+	0.929	VeryGood
0.137	0.245	+	-	+	+	0.91	VeryGood
0.173	0.094	+	+	-	-	0.938	VeryGood
0.326	0.18	-	+	+	-	0.903	VeryGood
1.119	0.203	-	-	-	-	0.945	VeryGood
0.563	0.079	-	+	-	+	0.949	VeryGood
0.544	0.723	-	-	-	-	0.945	VeryGood
1.964	0.406	+	+	+	-	0.945	VeryGood
0.062	0.961	+	-	-	+	0.936	VeryGood
0.063	0.036	-	+	+	-	0.913	VeryGood
0.303	0.153	+	+	+	-	0.936	VeryGood
0.054	0.032	-	+	+	-	0.913	VeryGood
0.069	0.098	+	-	+	-	0.92	VeryGood
0.104	0.188	+	+	+	-	0.929	VeryGood
0.057	0.931	-	-	+	+	0.947	VeryGood
0.344	0.072	+	-	-	+	0.94	VeryGood
0.129	0.328	+	+	+	-	0.94	VeryGood
0.44	0.77	+	-	+	-	0.938	VeryGood
1.048	0.208	-	-	+	+	0.945	VeryGood
0.302	0.41	+	+	+	+	0.933	VeryGood
0.076	0.084	-	+	+	+	0.926	VeryGood
0.816	0.091	-	+	-	-	0.945	VeryGood
0.487	0.126	-	-	+	-	0.945	VeryGood
0.291	0.343	-	+	+	+	0.922	VeryGood
1.607	1.589	+	-	+	+	0.94	VeryGood
0.733	0.732	+	-	-	+	0.952	VeryGood
1.261	0.055	+	+	-	+	0.949	VeryGood
0.101	1.267	-	-	-	-	0.91	VeryGood

Table A.6.: Accuracy for Dataset 'Vote Part 2'

rssAll	weightedFirstLayer	weightedAll	rssK	tpCovering	enforceTwoCond	accuracy	accuracyClass
+	+	+	5	+	0.339	0.945	VeryGood
+	+	-	4	-	0.65	0.945	VeryGood
-	-	-	-1	+	0.472	0.949	VeryGood
+	+	+	5	+	0.036	0.913	VeryGood
-	+	+	-1	-	0.708	0.945	VeryGood
+	-	+	7	+	0.59	0.908	VeryGood
+	+	+	6	-	0.029	0.945	VeryGood
-	-	-	-1	+	0.856	0.929	VeryGood
+	+	+	9	-	0.644	0.945	VeryGood
-	-	+	-1	+	0.915	0.931	VeryGood
-	+	-	-1	-	0.43	0.94	VeryGood
+	+	+	9	+	0.674	0.92	VeryGood
-	-	+	-1	-	0.259	0.933	VeryGood
-	-	+	-1	+	0.702	0.943	VeryGood
-	+	+	-1	-	0.241	0.945	VeryGood
-	-	+	-1	+	0.302	0.938	VeryGood
+	+	+	8	+	0.439	0.945	VeryGood
+	-	+	6	-	0.627	0.954	VeryGood
+	-	+	7	+	0.261	0.929	VeryGood
+	-	+	7	+	0.976	0.91	VeryGood
-	-	-	-1	-	0.023	0.938	VeryGood
-	+	-	-1	-	0.115	0.903	VeryGood
-	+	+	-1	+	0.724	0.945	VeryGood
-	+	-	-1	-	0.471	0.949	VeryGood
+	+	+	7	-	0.803	0.945	VeryGood
-	-	-	-1	+	0.873	0.945	VeryGood
+	-	+	9	+	0.978	0.936	VeryGood
-	+	-	-1	+	0.056	0.913	VeryGood
-	-	-	-1	+	0.336	0.936	VeryGood
-	+	-	-1	+	0.204	0.913	VeryGood
+	-	+	8	-	0.177	0.92	VeryGood
-	-	-	-1	-	0.789	0.929	VeryGood
-	+	+	-1	+	0.583	0.947	VeryGood
-	-	+	-1	+	0.98	0.94	VeryGood
-	-	-	-1	+	0.958	0.94	VeryGood
+	-	+	3	+	0.554	0.938	VeryGood
-	+	+	-1	+	0.465	0.945	VeryGood
-	-	-	-1	+	0.657	0.933	VeryGood
-	+	-	-1	+	0.9	0.926	VeryGood
+	+	-	8	-	0.037	0.945	VeryGood
-	+	+	-1	-	0.274	0.945	VeryGood
+	+	-	7	-	0.372	0.922	VeryGood
+	-	+	5	+	0.474	0.94	VeryGood
-	-	+	-1	-	0.183	0.952	VeryGood
-	-	-	-1	+	0.25	0.949	VeryGood
+	+	+	6	-	0.383	0.91	VeryGood