Learning Multi-Label Rules Using Bottom-Up Rule Induction

Lernen von Multi-Label Regeln mittels Bottom-Up Regel Induktion
Bachelor thesis by Pascal Dominik Hefter
Date of submission: January 8, 2020

1. Review: Prof. Dr. Johannes Fürnkranz
2. Review: M.Sc. Michael Rapp
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department
Knowledge Engineering Group

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Pascal Dominik Hefter, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 8. Januar 2020

Pascal Dominik Hefter

Abstract

In supervised learning, classification is the task of learning a model from labeled training data in order to be able to make predictions for yet unseen data. As opposed to binary classification, in multi-label classification, each example can be associated with multiple classes at the same time. For example, a song can often be assigned to multiple genres, such as *rock* and *ballad*. One approach to learn such classifiers is rule learning, which has the advantage of producing highly interpretable models. As there is not much research on separate-and-conquer rule induction using bottom-up search, especially for multi-label settings, and because there could be advantages in using example-driven strategies, such as learning label combinations that actually occur in the training data, in this work, a standard bottom-up separate-and-conquer rule learning algorithm is investigated. The proposed approach originates from binary classification, where the problem of finding an adequate generalization of rules has already been investigated. Different types of algorithms are considered and a detailed description of the proposed bottom-up rule learner is given. Since there are some different notations and definitions for the multi-label setting, a new way to define rule evaluation methods, together with a more convenient naming scheme, which is better suited for multi-label rule learning, is proposed. In addition to a comprehensive evaluation of the proposed algorithm and methods, an outlook is given on how to possibly deal in multi-label classification with the problems found for bottom-up binary classification.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

This introduction provides an overview of multi-label classification and rule learning in combination with the objectives of this thesis as well as the structure of the following chapters.

1.1. Motivation

In this work, we investigate a bottom-up approach for learning multi-label rules. This is an extension of the basic setting called binary classification, where a model is learned from training data in order to predict for a yet unseen example to exactly which one of the two possible classes it belongs. Another possible setting is called multi-class classification. This means, that there are not only two mutually exclusive classes, but there can be more classes and the goal is to predict for unseen examples to exactly which one of these classes the example belongs. This problem can be extended again by allowing every example to belong to more than just one class, which leads to the problem setting of this work, namely multi-label classification. A frequently used example of multi-label classification is the task of assigning a given newspaper article to one or several topics. As another example, a song might belong to the genre *rock* as well as to the genre *ballad* [Tsoumakas and Katakis, 2007].

The above described method of assigning multiple labels to an example is often transformed into multiple sub-problems using the so-called Problem Transformation Methods [Tsoumakas and Katakis, 2007]. However, there are several reasons not to transform the multi-label problem into sub-problems. For example, when using binary relevance, label dependencies can not be considered [Tsoumakas and Katakis, 2007]. Loza Mencía [2012] also proposed a separate-and-conquer rule learning algorithm for multi-label problems using top-down search. As the bottom-up variant of this algorithm might offer several advantages, such as learning label combinations that actually occur in the training data,

and since it has not been investigated so far, this work focuses on the bottom-up approach for solving the multi-label classification problem using rule induction. Based on previous work on binary classification, it is known that in bottom-up rule learning a generalization problem can occur [Fürnkranz, 2002], which is found to be present in multi-label rule learning as well. Therefore, this work aims at finding a reliable basic multi-label bottom-up rule learning algorithm, which allows further work to extend it, such that it can compete with the more frequently used top-down algorithms. Additionally, a new way to formulate rule evaluation metrics in the multi-label setting is given, which combines several existing definitions in order to deal with challenges that occur with the proposed bottom-up algorithm.

1.2. Organization of this Work

In this section, a short overview of the following chapters is given.

- **Chapter 2** gives a detailed introduction on (multi-label) classification, rule learning and rule induction, including notations, basic algorithms and examples. All further investigations in this work rely on the foundations given in this chapter.
- **Chapter 3** proposes and discusses three different concepts to face the multi-label rule learning setting and explains the choice of the concept for this work. The two remaining approaches are left for future work and have no further relevance in this work.
- **Chapter 4** presents the standard algorithm as well as the default parameters that are used in this work.
- **Chapter 5** discusses existing evaluation metrics for multi-label rule learning and proposes a new naming of those as well as a new definition of an evaluation metric in order to cope with arising challenges.
- **Chapter 6** analyses the evaluation results of the proposed bottom-up algorithm on real multi-label data sets in terms of consistency and coverage as well as the influence of the proposed changes to the evaluation metrics.
- **Chapter 7** gives an overview of possible extensions of the standard bottom-up algorithm in order to increase its performance. A description of how they work and when and why they could be further investigated is given.
- **Chapter 8** summarizes this work and gives an outlook on what could be further researched in the field of bottom-up multi-label rule learning including concrete approaches outlined throughout this work.

2. Foundations

The following sections provide a comprehensive overview of the fundamentals of multi-label rule learning. This includes formal definitions as well as common problem transformation problems and rule induction. Furthermore, the baseline of the separate-and-conquer algorithm proposed by Fürnkranz [1999], Loza Mencía and Janssen [2016] is outlined and common evaluation methods are explained.

2.1. Classification in Machine Learning

In general, classification in machine learning is the task of learning a model from a training data set. A model generalizes from the learned training instances in order to classify new examples. A training data set consists of instances, which are attribute-value pairs and have one or more associated classes. In this work, only attributes that are either numeric or nominal are considered. A nominal attribute has a discrete value such as *sunny* or *windy* and those attributes cannot be ordered. In contrast, numeric attribute values are continuous, e.g. *70.5* as a temperature value and they can be ordered.

The multi-label classification problem setting is an extension of binary and multi-class classification problems. In binary classification, there is only a single class and every example gets classified as either belonging to the single class or not belonging to the single class. This setting can be extended to multi-class classification, where more than one class can exist and each example can only be assigned to exactly one of these classes.

2.1.1. Multi-Label Classification

The difference between multi-class and multi-label classification is that in multi-label classification, each example can be assigned to any number of classes simultaneously. The

classes are often also referred to as labels. A commonly used example is the assignment of newspaper articles to certain genres. In a multi-label setting with n different classes or labels, a label vector can have label combinations of the form $y = (y_1, ..., y_n) \in \{0, 1\}^n$ [Loza Mencía, 2012, Loza Mencía and Janssen, 2016].

2.1.2. Problem Transformation Methods

There are different ways to solve a multi-label classification problem. In this section, some well-known problem transformation methods are discussed, as they are proposed by Tsoumakas and Katakis [2007]. These transform a complex multi-label classification task into many binary classification problems, which are easily solvable by binary classifiers. To get a result for the original multi-label task, the single-class predictions must be converted to multi-label predictions, which is further described in the following.

- **Binary Relevance:** This method uses n binary classifiers to solve a multi-label problem with n labels. Each binary classifier predicts the presence of one of the multi-label problem labels. As there is one classifier per label, the outcome for the multi-label problem is determined by transforming the single predictions into a label vector, which predicts all labels for the given instance. This approach is equivalent to solving the original multi-label classification problem using only single-label head rules, meaning that the head of a rule may only contain one label.
- **Pairwise Decomposition:** When this method is used, a binary classifier is learned for every possible pair of labels, which leads to $\frac{n(n-1)}{2}$ subtasks. In order to predict the labels for the original problem, each prediction for each label by any binary classifier is accumulated. The result can be seen as a voting for each label and whether or not a label should be predicted can then be determined by sorting the votes and separating using a threshold.
- **Label Powerset:** Using this approach, a multi-label problem with n different labels is decomposed into 2^n multi-class problems, where each multi-class problem represents one possible label combination of the original n label multi-label problem. For solving these meta classification tasks, either a common multi-class classifier can be used or the above explained binary relevance strategy can be applied. The resulting class of the meta classifier is then re-transformed into the label combination of the multi-label problem and describes the predicted labels.

2.2. Inductive Rule Learning

This section describes the composition of a rule as proposed by Loza Mencía and Janssen [2016]. A rule consists of a body and a head in the form $head \leftarrow body$. The body is a conjunction of conditions, where each condition defines an attribute-value test. The head contains the predictions for the labels, which can either be 1 (label is relevant) or 0 (label is irrelevant). A rule covers an instance, if the conjunction of the body of the rule is $true$ for the instance's attribute values. This is also referred to as the rule firing for a given instance. The predicted labels for this instance are then set according to the head of the rule.

2.3. Search Strategies

The following different approaches describe the direction in which the rules are learned, where a rule A is more general than rule B if and only if rule A covers all instances that are covered by rule B. Accordingly, rule B is more specific than rule A if and only if rule A is more general than rule B [Fürnkranz, 1999].

2.3.1. Top-Down Search

This general-to-specific search is used in most cases of separate-and-conquer algorithms. Using the top-down approach, a new rule is initialised with no conditions, which is the most general rule and covers every example. This rule then gets refined by successively adding conditions to the body of the rule, which is a specialization of the rule resulting in a fewer amount of covered examples. This procedure is repeated until a stopping criterion intercepts and the rule is added to the rule set.

2.3.2. Bottom-Up Search

This is the search strategy used in this work. Opposite to the top-down approach, in bottom-up rule learning, a new rule usually gets created from a randomly sampled training instance which is not yet covered by the model rule set. The resulting rule is very

specific, as it mostly only covers the example that it was created from. In order to generalize it and to cover more examples, in every iteration one condition is removed from the body of the rule, as long as a certain threshold is not reached. This process is repeated for every newly created rule.

This approach has not been investigated on multi-label rules so far. However, as it is an example-driven strategy, which repeatedly starts with an instance from the training set, the label combinations predicted by the rules are always existing combinations, at least in the training set.

2.3.3. Bidirectional Search

A combination of top-down and bottom-up search leads to bidirectional search. In most cases, this search is used with a tendency to the top-down approach, where a generalization of a rule is only applied if the rule covers too few examples.

2.4. Separate-and-Conquer Rule Learning

Separate-and-Conquer describes a strategy in rule learning, which iteratively extracts rules from the training data whilst reducing the size of the training set by removing all examples, that are covered by the newly learned rule. A new rule is refined until a stopping criterion stops the refinement, in order to find the best possible rule.

Algorithm 1 SeparateAndConquer [Fürnkranz, 1999]

```
1: Theory = ∅
2: while GetPositiveExamples(TrainingSet) ≠ ∅ do
3:     BestRule = FindBestRule(TrainingSet)
4:     CoveredExamples = GetCoveredExamples(TrainingSet,BestRule)
5:     Theory = Theory ∪ BestRule
6:     TrainingSet = TrainingSet \ CoveredExamples
7: end while
8: return Decision List R
```

2.5. Multi-Label Rule Learning

In machine learning, rule learning is a method to learn a model which consists of a set of rules. These rules can then be used to predict a class for unseen examples. Rules form relatively simple models compared to other advanced machine learning methods. However, they are easily understandable also for humans and can be easily modified. [Loza Mencía and Janssen, 2016]

2.5.1. Types of Multi-Label Rules

In binary classification, the head consists of one single label, which can either be 0 or 1, indicating whether the covered example does or does not belong to the single class. In multi-label classification the number of labels in the head can vary between 1 and the number of existing labels in the training data set. Table 2.1 shows the different types of multi-label rules, where each c_i represents a condition and each $\hat{y}_i$ the predicted label of the rule. Label-dependent means, that the body of a rule can contain conditions on the labels, whereas this is not the case with label-independent rules [Loza Mencía and Janssen, 2016].

Table 2.1.: Different types of multi-label rules. [Loza Mencía and Janssen, 2016, Rapp, 2016]

Head	Body	Example Rule
sparse dense	label-independent	$\hat{y}_1, \hat{y}_2 \leftarrow c_1 \wedge c_2 \wedge c_3$ $\hat{y}_1, \neg\hat{y}_2, \neg\hat{y}_3 \leftarrow c_1 \wedge c_2 \wedge c_3$
sparse dense	partially label-dependent	$\hat{y}_3, \hat{y}_4 \leftarrow c_1 \wedge \neg\hat{y}_1 \wedge \hat{y}_2$ $\hat{y}_3, \neg\hat{y}_4, \neg\hat{y}_5 \leftarrow \neg c_1 \wedge \neg\hat{y}_1 \wedge \hat{y}_2$
sparse dense	fully label-dependent	$\hat{y}_3, \hat{y}_4 \leftarrow \neg\hat{y}_1 \wedge \hat{y}_2$ $\hat{y}_3, \neg\hat{y}_4, \neg\hat{y}_5 \leftarrow \hat{y}_1 \wedge \neg\hat{y}_2$

2.6. Rule Models

The rule set returned by the separate-and-conquer algorithm can either be a sorted or an unsorted list of rules.

2.6.1. Decision Lists

A sorted list of rules is called a decision list. Three variants of decision lists are considered as described by Loza Mencía and Janssen [2016]. Every decision list is ordered in the same way as the rules are learned, i.e. the first discovered rule is also the first one in the decision list.

- **Binary Decision List:** The rules are checked in order of the decision list whether they cover the given instance. If that is the case, the firing rule predicts the class of the instance and no further rules from the decision list are checked, as there are only two mutual exclusive outcomes in binary classification.
- **Multi-Class Decision List:** In multi-class classification, the decision list works just as in binary classification, except that the rule which fires first predicts one class out of all possible classes. No further predictions are made afterwards for this instance.
- **Multi-Label Decision List:** As a rule in multi-label classification might only predict some of the true labels as positive for a given instance, in this case an extension of the classic decision list is proposed by Loza Mencía and Janssen [2016]. Hereby, the classification for an instance does not stop after the first rule covers it, but it is continued with the following rules in the decision list such that a new firing rule can add positive labels to the previous prediction for the instance. In order to prevent that in the end, all labels are predicted positive for the given instance, stopping rules are added to the model, which terminate the classification for the instance.

2.6.2. Disjunctive Normal Form

The unsorted case equals a Disjunctive Normal Form (DNF), which means that there is no order for the rules, instead for every rule it is checked whether it fires for the given example and if so, all of its predicted labels are set for the given instance.

2.7. Evaluation of Multi-Label Predictions

This chapter explains the methods and metrics that are usually used to evaluate the quality of the predictions of classifiers [Loza Mencía, 2012]. However, in Chapter 5 an altered

definition of some evaluation metrics will be introduced, as the given multi-label problem needs a more refined evaluation.

2.7.1. Confusion Matrix

The confusion matrix as shown in Table 2.2 contains information about the predictions of a rule. It consists of the entries defined as follows:

- **True Positive (TP):** The label is relevant and is predicted to be relevant.
- **False Positive (FP):** The label is irrelevant, but was predicted to be relevant.
- **False Negative (FN):** The label is relevant, but was predicted to be irrelevant.
- **True Negative (TN):** The label is irrelevant and is predicted to be irrelevant.

Table 2.2.: Confusion Matrix [Loza Mencía, 2012]

C	predicted	not predicted
relevant	TP	FN
irrelevant	FP	TN

2.7.2. Aggregation and Averaging

As there is a confusion matrix for every rule and for every instance, those matrices somehow have to be combined in order to get a single score which gives an estimation on how good a rule or model is. There are two main distinctions in combining these confusion matrices: micro- and macro-averaging, which differ in the order of aggregation and application of the evaluation function and are described in the following [Loza Mencía, 2012].

- **(Label and Example-Based) Micro-Averaging:** In this case, all computed matrices, for examples as well as for labels, are added up and the resulting matrix is then evaluated to a single score by the applied evaluation function.
- **Example-Based (Macro-)Averaging:** For each example, the confusion matrix is constructed by putting the labels together. The resulting example-wise matrices are then evaluated and averaged using the arithmetic mean.

- **Label-Based (Macro-)Averaging:** For each label, the confusion matrix is computed by aggregating the entries for the label. Then, each matrix is evaluated by the evaluation function and the resulting values are averaged using the arithmetic mean to get a final score.
- **(Label and Example-Based) Macro-Averaging:** Here, all atomic confusion matrices are computed for the labels as well for the examples. Then, to each of them the evaluation function is applied. The scores are then averaged using the arithmetic mean, where either first the example-wise values and then the label-wise values are averaged or the other way round.

As described by Loza Mencía [2012], choosing an adequate strategy is crucial as labels and examples have different influences for different strategies. For example, instances with many associated labels have a high influence on the result in micro-averaging, whereas in example-based averaging all examples are weighted equally. On the other hand, micro-averaged results are stronger influenced by labels which are associated with many examples and in order to consider all labels equally, label-based averaging should be used.

2.7.3. Selected Evaluation Functions

Evaluation functions are needed to evaluate the quality of rules. They are often referred to as heuristics [Janssen, 2012] and in this section some well-known heuristics used in this work are described.

- **Precision:** This heuristic computes the percentage of correctly predicted labels over all predicted labels.

$$\delta_{prec} = \frac{TP}{TP + FP} \tag{2.1}$$

- **Recall:** This heuristic measures the coverage, as it computes the percentage of correctly predicted labels among all relevant labels.

$$\delta_{rec} = \frac{TP}{TP + FN} \tag{2.2}$$

- **F-measure:** F-measure is a combination of Precision and Recall, which trades off Precision and Recall with the parameter $\beta \in [0, \infty)$. It is also called the harmonic mean if $\beta = 1$.

$$\delta_F = \frac{(\beta^2 + 1) * \delta_{prec} * \delta_{rec}}{\beta^2 * \delta_{prec} + \delta_{rec}} \tag{2.3}$$

- **Subset Accuracy:** Subset Accuracy can only be calculated example-based, as it is the percentage of perfectly predicted label vectors, where m is the number of examples, y_i the true label vector and $\hat{y}_i$ the predicted label vector. Additionally, $[y_i = \hat{y}_i]$ returns 1 if $y_i = \hat{y}_i$ is $true$ and 0 otherwise [Loza Mencía and Janssen, 2016].

$$\delta_{sub-acc} = \frac{1}{m} \sum_{i=1}^{m} [y_i = \hat{y}_i] \tag{2.4}$$

2.8. Full Prediction vs. Partial Prediction

For the evaluation of a rule, two types of predictions are distinguished as proposed by Rapp et al. [2018]:

- **Full Prediction:** Using full predictions, all existing labels of the data set are taken into account, regardless of the head of the rule. Any label that is not contained in the head is thus considered to be predicted irrelevant, i.e. $\hat{y}_i = 0$.
- **Partial Prediction:** Partial prediction only considers the labels that are contained in the head of the rule.

2.8.1. Toy Example

This distinction becomes relevant, when a rule predicts some positive labels, but does no prediction for some other labels. The following toy example illustrates the different results already in a very simple setting (cf. Table 2.3). In the example, the rule predicts only the label $\hat{y}_2 = 1$ but does no prediction for label y_1. Furthermore, the instance covered by the given rule has the true labels $y_1 = 0$ and $y_2 = 1$.

Considering partial prediction, both Precision and Recall are 1, because only the prediction on y_2 is evaluated and the label is correctly predicted positive. Using full prediction, $\hat{y}_1$ is evaluated as $\hat{y}_1 = 0$ and therefore is a FN, which leads to a decrease in Recall.

Table 2.3.: Evaluation Toy Example

head of rule	$\hat{y}_2 = 1$
covered instance	$y_1 = 1, y_2 = 1$
partial	$\delta_{prec} = \frac{1}{1+0} = 1 \quad \delta_{rec} = \frac{1}{1+0} = 1$
full	$\delta_{prec} = \frac{1}{1+0} = 1 \quad \delta_{rec} = \frac{1}{1+1} = \frac{1}{2}$

2.9. Related Work

There is no research so far on bottom-up multi-label classification with rule induction. However, there are two main approaches that cover a part of the investigated setting of this work, both using rule induction.

Fürnkranz [2002] investigated a bottom-up separate-and-conquer algorithm for binary classification using rule induction. The paper shows that there is a pathology in bottom-up rule learning for separate-and-conquer algorithms, because a rule can often not be generalized enough to cover the nearest instance using only one generalization step. That is, because in many cases the remaining instances in the training set differ in at least two attributes from the instance the rule was created from. As in one generalization step, only one of the rule conditions is removed or refined, it is not possible to increase the coverage of the rule in such a case. However, Fürnkranz [2002] proposed some approaches on how to overcome this problem, which are also mentioned in Chapter 7, because this problem also applies to bottom-up multi-label classification with rule induction and therefore should be considered in future work in order to achieve results that can compete with state-of-the-art classifiers.

Additionally, there is research on top-down multi-label classification [Loza Mencía and Janssen, 2016, Rapp, 2016]. Loza Mencía and Janssen [2016] proposed a separate-and-conquer algorithm for multi-label classification, which can learn dependencies between labels, but only uses rules that predict the presence of one label. This approach was extended to learning multi-head rules by Rapp [2016]. However, all of these algorithms use top-down search, which leaves the bottom-up separate-and-conquer algorithm with rule induction for this work.

3. Choice of the Standard Algorithm

In the following, three conceptional approaches to the given multi-label problem using bottom-up search are discussed. However, this is not a complete list and there might be other good approaches. In this work, a rule always has the head according to the randomly chosen training instance that it got created from (cf. Subsection 2.3.2). This head is not changed throughout the whole rule refinement process. Additionally, it is assumed, that in the algorithm used in this work, negative labels are not explicitly predicted and are therefore only relevant when using full predictions.

3.1. Choice for this Work

In the remainder of this work, only the first approach is used for the standard algorithm. The other two approaches are left for future work, as the second approach builds upon the first one and the random rule induction does not use a separate-and-conquer algorithm.

3.2. Multi-Class Separate-and-Conquer Algorithm

The simplest scenario is to treat the label combination of the randomly chosen instance, according to which a new rule is created, as one possible class. This results in several different classes depending on the label combinations of the randomly chosen instances. Whenever a training instance is covered by the current rule, it is removed from the test set. Accordingly, an unseen test instance is classified by the resulting decision list which is ordered in the order that the rules are learned. Therefore, only the first firing rule defines the label combination for the given instance. As the prediction of the rule only contains the labels that are given by the according randomly sampled instance, in this scenario full predictions should be used. This also takes into account the yet unpredicted labels

for the instance, which can afterwards not be predicted, because only the first firing rule is considered.

3.3. Multi-Label Separate-and-Conquer Algorithm

There are two differences between this approach and the previously described. The first is that an instance, which is covered by the current rule in the training phase is only removed from the training set, if all of its labels are covered. Otherwise, the instance remains in the training set until also the remaining labels are covered by some following rules. Accordingly, the prediction for a new instance is not limited to the head of the first firing rule in the decision list. Instead all firing rules in the decision list add their labels to the prediction for the given instance. In order to evaluate rules in this scenario, partial predictions should be used, since only the labels predicted by the current rule are relevant.

3.4. Random Rule Induction

Another approach is to learn the rules without the separate-and-conquer strategy and instead use random rule induction, where each training instance is chosen once to be generalized using the whole training set without removing any covered instances. In this case, either a decision list sorted decreasingly by the heuristic value of the rules can be used or a DNF, which is reasonable since it evaluates in exactly the same way, as the rules were created in the training phase. However, it might lead to an overuse of labels if no stopping criterion is used.

4. Bottom-Up Separate-and-Conquer Algorithm

In this chapter, the standard bottom-up algorithm is described. All further investigations in this work are based on this algorithm.

4.1. Standard Bottom-Up Implementation

The standard bottom-up algorithm (cf. Algorithm 2) is adapted from the simple separate-and-conquer algorithm from Fürnkranz [1999], Loza Mencía and Janssen [2016]. The initial rule set is empty. As long as there is a certain percentage of uncovered training instances, new rules are created iteratively. A rule is refined by the Algorithm 3 in order to optimize its quality and is then added to the theory. All of its covered instances are removed from the training set. The learned model is given in the form of a decision list, i.e. for making a prediction the learned rules are traversed in the order of how they were learned and after the first rule fires for the given instance, no further rules are taken into account.

As specific-to-general search is used, Algorithm 3 initially starts with a rule that is created from a randomly sampled instance (Algorithm 4). In order to find the best generalization of this rule, all possible generalizations that result from removing one condition are computed using Algorithm 6 and then the best of these rule candidates is chosen using the rule comparison as described in Subsection 4.1.5.

Algorithm 2 BottomUp SeparateAndConquer

```
1: Theory = ∅
2: while GetUncoveredExamples(TrainingSet) ≥ remainingInstancesPercentage ∗ |TrainingSet| do
3:     r_best = FindBestRule(TrainingSet)
4:     CoveredExamples = GetCoveredExamples(TrainingSet,BestRule)
5:     Theory = Theory ∪ r_best
6:     TrainingSet = TrainingSet \ CoveredExamples
7: end while
8: return Decision List R   ▷ Theory in order of the first learned rule to the last learned rule
```

Algorithm 3 FindBestRule, adapted from Loza Mencía and Janssen [2016], Rapp [2016]

```
1: r_best = CreateNewRule(TrainingSet)
2: improved=TRUE
3: while improved do
4:     r_refined = RefineRule(r_best,TrainingSet)
5:     if Evaluate(r_refined) ≥ Evaluate(r_best) then        ▷ compare heuristic values
6:         r_best = r_refined
7:     else
8:         improved = FALSE
9:     end if
10: end while
11: return r_best
```

4.1.1. Rule Instantiation

Algorithm 4 shows how to create a new rule from a randomly chosen instance. The head of the rule is created according to the labels of the instance. For the body, each attribute-value pair of the chosen instance is turned into one or more conditions. As there are two types of conditions, nominal and numeric, two cases have to be considered:

- **nominal attribute:** In this case, the condition of the rule simply checks whether the value is equal to the one of the instance.
- **numeric attribute:** Algorithm 5 describes how to proceed in case of a numeric attribute. In order to find a good split point between the sorted existing values in the training data set, the interval values are chosen as follows: $v_i' = \frac{v_i + v_{i+1}}{2}$.

Algorithm 4 CreateNewRule

```
1:  r = ∅ ← ∅
2:  Instance = SelectRandomInstance(TrainingSet)
3:  for each (Attribute,Value) in Instance do
4:      if Attribute is nominal then
5:          Condition = (Attribute = Value)
6:      else
7:          Condition = CreateNumericCondition(Attribute,Value)
8:      end if
9:      r.body = r.body ∪ Condition
10: end for
11: for each Label in Instance do
12:     r.head = r.head ∪ Label
13: end for
14: return r
```

Algorithm 5 CreateNumericCondition, adapted from Rapp [2016]

1: SortedValues = SortAttributeValues(TrainingSet,Attribute)
2: Intervals = GetIntervalValues(SortedValues) ▷ calculate using $\frac{v_i+v_{i+1}}{2}$
3: NearestHighValue = FindNearestHigherValue(Intervals,Value)
4: NearestLowValue = FindNearestLowerValue(Intervals,Value)
5: NumericCondition = (Attribute $>$ NearestLowValue $\wedge$ Attribute $\leq$ NearestHighValue)
▷ alternatively use $\geq$ and $<$
6: **return** NumericCondition

4.1.2. Refinement of a Rule

In order to refine a rule using bottom-up search, a refinement is always executed in form of a generalization. Since the rule in most cases consists of a conjunction of more than one condition, there are many different possible generalizations of this rule, i.e. one generalization for each condition, which has to be considered in one iteration of a refinement. Thus, as shown in Algorithm 6, in every refinement step, all of these possible generalizations are computed and their quality is assessed in terms of a heuristic value as further described in Subsection 4.1.4. Then, they are compared according to their heuristic values and the refined rule with the better result is kept. If all refinements lead to an aggravation, the original rule is not further refined and is added to the rule set [Fürnkranz, 1999].

Algorithm 6 RefineRule

1: $r_{refined} = r$
2: **for each** Condition in $r_{refined}$ **do**
3: $r_{refined}$ = GeneralizeCondition($r_{refined}$,Condition)
4: **if** Evaluate($r_{refined}$) $\geq$ Evaluate(r_{best}) **then**
5: $r_{best} = r_{refined}$
6: **end if**
7: **end for**
8: **return** r_{best}

4.1.3. Generalization of Conditions

As there are two different types of attributes – nominal and numeric –, the way of generalizing a condition depends on its type (Algorithm 7). In the nominal case, simply removing the whole condition allows any value for the given attribute instead of the previously specified. In case of a numeric attribute, the condition may use the operators $<$, $>$, $\leq$ and $\geq$. The proceedings for both scenarios only differ in such way that a generalization of a condition using $<$ or $\leq$ has to define a *higher* value and for $>$ or $\geq$ a *lower* value has to be found (Algorithm 5). In order to improve the performance or to cope with problems where a generalization of a numeric condition does not cover more instances than initially, the generalization can be used with bigger step size or random steps can be taken, as it is described in Section 7.1. However, the proposed standard algorithm in this work always uses one-step generalization if not stated otherwise.

Algorithm 7 GeneralizeCondition

1: $r_{refined} = r_{refined} \setminus$ Condition
2: **if** Condition.Attribute $=$ numeric **then**
3: $\quad r_{refined} \cup$ CreateNumericCondition(Condition.Attribute,Condition.Value) ▷ only use the according higher/lower condition of the two returned conditions
4: **end if**
5: **return** $r_{refined}$

4.1.4. Evaluation of a Rule

In order to measure the quality of a rule, it has to be evaluated in terms of a heuristic value. The different methods for this evaluation are described in Section 2.7. As described in Section 3.2, for the standard bottom-up algorithm, the evaluation with full predictions is preferable. No restrictions are imposed on how to aggregate and average the predictions obtained for individual labels and examples. The rules are evaluated on the training instances that are not yet covered by other rules using F-measure, because this heuristic allows a trade-off between consistency and coverage using the β parameter.

4.1.5. Rule Comparison

One crucial decision is how to compare the currently best rule and the refined rule. In Rapp [2016], a refined rule will become the new best rule if and only if its heuristic value is higher than the currently best rules, i.e. because when using top-down search, the currently best rule is more general and therefore preferable in most cases where their heuristic values are equal. However, in the bottom-up implementation it is more promising to generalize a rule whenever the generalization does not aggravate the heuristic value [Fürnkranz, 2002]. That is, because in many cases the generalized rule does not cover more instances after one generalization. But two instances often differ in at least two attribute values, which makes it necessary to generalize such a rule at least two times to actually increase its coverage. The comparison of two rules does not only concern the heuristic value, but a series of rule component comparisons used for tie breaking, which are listed in the following. Whenever a component comparison is equal for both rules, the next comparison is considered.

1 **Heuristic Value:** The rule with the higher heuristic value is chosen. Using specific-to-general search, the above described problem often occurs. Therefore in many cases, the heuristic value of the original and all its one-step generalizations have the same heuristic value. In this case the following comparisons have to be applied.

2 **Rule Length:** In rule learning, shorter rules are in most cases considered more valuable. However, it is not desirable to have rules that cover too many instances. This is prevented by not considering rules with a worse heuristic value. As we start with a very specific rule, created from a training instance, a shortening of the rule is predominantly useful. Therefore, the rule with fewer conditions is chosen.

3 **Number of Generalizations:** Given that both of the previous comparisons lead to no decision, the amount of generalization steps performed on each of the rules are compared. This is relevant, because numeric conditions can be refined without removing the condition. For the same reason as in Item **2**, the rule with more generalization steps is chosen.

4 In the rare case, where the rules are not identical but still equal in all above described comparisons, the rule is chosen by random choice.

4.1.6. Parameters

In the following, an overview of the parameters used by the standard algorithm is given. If not stated otherwise, this configuration is used in the remainder of this work.

- **Heuristic: F-Measure:** This heuristic is commonly used in rule learning as it trades off Precision and Recall [Janssen, 2012]. Precision aims at covering as few negative examples respectively labels as possible by minimizing the amount of False Positives. This goes well with top-down search strategies, as the search starts with the most general rule covering all examples and therefore mainly consistency needs to be improved. On the other hand, Recall aims at maximizing the amount of covered positive examples by minimizing False Negatives. Therefore, Recall goes well with specific-to-general search, as it starts with a very small coverage and perfect consistency. Though using a bottom-up search strategy, the aim of the proposed algorithm is not to generalize until there is a single empty rule covering all instances, namely a Default-Rule, but to find a good trade-off between coverage and consistency. Thus, F-measure fits well, since the influence of Precision and Recall can be tuned with the parameter β.
- **Remaining Instances Percentage:** This parameter gives the percentage of the training set that has not to be used for training, as there might be some outliers that are not covered by any rule. In many cases, the remaining instances can be classified by a default rule with very few conditions. Values ≤ 0.1 should be considered and in this work, 0.05 is used.
- **Averaging Strategy:** As all of the averaging strategies described in Subsection 2.7.2 have their advantages, the choice of the specific averaging strategy depends on the experiment. However, if not stated otherwise, micro-averaging is used in this work.
- **Evaluation Strategy:** As already described in Section 3.2, full predictions are chosen for the standard algorithm.

5. Rule Evaluation for Multi-Label Rules

In order to obtain multi-label rules with a good quality in terms of traditional evaluation, but which also work for the multi-label problem, some modifications have to be made to the classical notation and semantic of the confusion matrix and therefore also the bipartition evaluation metrics. Thus, some changes to the metrics used by Loza Mencía [2012] are useful for this multi-label bottom-up algorithm, which can be seen in the evaluation in Chapter 6.

5.1. Classic Bipartition Evaluation Metrics

We recall the bipartition evaluation metrics and the confusion matrix in Table 5.1. Referring to classic binary classification, the entries have the following meaning:

- **True Positive (TP):** The instance belongs to the positive class and was correctly predicted as positive.
- **False Positive (FP):** The instance does not belong to the positive class, but was mistakenly predicted as positive.
- **False Negative (FN):** The instance belongs to the positive class, but was mistakenly predicted as negative.
- **True Negative (TN):** The instance does not belong to the positive class and was correctly predicted as negative.

Table 5.1.: Confusion Matrix [Loza Mencía, 2012]

C	predicted	not predicted
relevant	TP	FN
irrelevant	FP	TN

5.2. Multi-Class Bipartition Evaluation Metrics

As proposed by Loza Mencía [2012], for multi-label classification, the bipartition evaluation metrics and the confusion matrix must take into account each label and instance. This leads to the following definition (Table 5.2, Table 5.3):

- **TP:** The label is relevant and is correctly predicted to be relevant.
- **FP:** The label is irrelevant, but was mistakenly predicted to be relevant.
- **FN:** The label is relevant, but was mistakenly predicted to be irrelevant.
- **TN:** The label is irrelevant and is correctly predicted to be irrelevant.

Table 5.2.: Confusion Matrix

y_i	predicted	not predicted
relevant	TP	FN
irrelevant	FP	TN

Table 5.3.: Classic Bipartition Evaluation Metrics

	y	$\hat{y}$	
covered	0	0	TN
	0	1	FP
	1	0	FN
	1	1	TP
uncovered	0	0	TN
	0	1	TN
	1	0	FN
	1	1	FN

5.3. Adapted Multi-Label Bipartition Evaluation Metrics

In binary classification, if a rule does not cover an instance, we have a TN if the instance does not belong to the single class and a FN if the instance does belong to the single

class. That is, because a rule not covering an instance is equivalent to the rule predicting that instance as negative. Adequately, a rule covering an instance is equivalent to the rule predicting the instance as positive, which is either correct (TP) or incorrect (FP). However, in multi-label classification, a rule can cover an instance and predict for every label if it is positive or negative. Therefore a new case distinction has to be considered.

Rapp et al. [2018] used a slightly different bipartition evaluation metric for multi-label rules, where correct predictions for covered instances are counted as TP for positive labels as well as for negative labels (Table 5.4). The column y represents the true label of an instance – whether it is relevant (1) or irrelevant (0) – and the column $\hat{y}$ represents the predicted label of the rule classifying the instance.

Table 5.4.: Adapted Bipartition Evaluation Metrics

	y	$\hat{y}$	
covered	0	0	TP
	0	1	FP
	1	0	FP
	1	1	TP
uncovered	0	0	TN
	0	1	TN
	1	0	FN
	1	1	FN

This leads to the following definition:

- **TP:** The instance is covered by the rule and the label is predicted according to its true label.
- **FP:** The instance is covered by the rule, but the label is predicted the opposite of the true label.
- **FN:** The instance is not covered by the rule, but the true label is positive, therefore a positive label is missed, regardless of the prediction.
- **TN:** The instance is not covered by the rule and the true label is negative, therefore the prediction does not matter.

5.3.1. Problems with Existing Definitions

The problem with this definition occurs especially when using specific-to-general search as in this work. Neither the classic nor the adapted metric take into account the prediction for labels of uncovered instances. As the heuristic F-measure is used, Precision and Recall are relevant for the rule evaluation. In the case of Precision, no problem occurs, since it only takes into account the TP and FP, which only occur on covered instances and is not responsible for a generalization. However, for Recall, the False Negatives, which aggravate the Recall score, occur for uncovered instances. In the classic metrics, whenever a positive label is not covered, the Recall goes down. But in order to rate the Recall in such a case in a multi-label setting, the prediction also becomes relevant. As the aim of bottom-up search is to be able to cover this instance if possible, not covering a positive label, that would be correctly predicted positive if it was covered, needs to be punished with a False Negative in Recall. This way, generalizing the rule to cover this instance will increase Recall, which is the desired behavior. Accordingly, a correctly predicted negative label for an *uncovered* instance should aggravate the Recall score such that covering the instance will increase Recall. This leads to the following redefinition for uncovered instances.

5.4. Prediction-Dependent Multi-Label Bipartition Evaluation Metrics

According to the above explanation, Table 5.5 shows the proposed prediction-dependent evaluation metric. The changes for *uncovered* instances are as follows:

- **y=0 and $\hat{y}$=1:** This is now a True Negative, because the true label is negative, but would be predicted positive, if it was covered. Thus, covering this label with this rule is not desirable.
- **y=1 and $\hat{y}$=0:** Accordingly, in this case, a positive label is missed, but it should not be covered by this rule, as it would still be missed if mistakenly predicted negative.
- **y=1 and $\hat{y}$=1:** If this label was covered, it would be a True Positive. In order to achieve generalization, it is counted as False Negative, which leads to a worse recall score if the instance is not covered.
- **y=0 and $\hat{y}$=0:** Again, this is the same case just for a negative label, but with the same explanation for it to be a False Negative.

Table 5.5.: Prediction-Dependent Bipartition Evaluation Metrics

	y	$\hat{y}$	
covered	0	0	TP
	0	1	FP
	1	0	FP
	1	1	TP
uncovered	0	0	FN
	0	1	TN
	1	0	TN
	1	1	FN

5.5. Naming of the Prediction-Dependent Metric

For the multi-label setting, we have to consider three dimensions in order to fully describe a prediction of a rule for the label of a given example, which are the following:

- **covered/uncovered (C/U):** whether the rule covers the instance or not
- **relevant/irrelevant (R/I):** whether the label is relevant (1) or irrelevant (0)
- **predicted positive/predicted negative (P/N):** whether the rule predicts the label as positive (1) or negative (0). A label which is not predicted is considered as predicted negative.

This results in the terminology given in Table 5.6.

Table 5.6.: Definition of Naming

	y	$\hat{y}$	
covered	0	0	C_{IN}
	0	1	C_{IP}
	1	0	C_{RN}
	1	1	C_{RP}
uncovered	0	0	U_{IN}
	0	1	U_{IP}
	1	0	U_{RN}
	1	1	U_{RP}

5.6. Definitions of Recall

We recall the classic definition of the Recall heuristic:

$$\delta_{rec} = \frac{TP}{TP + FN} \tag{5.1}$$

It consists of the following three main components, which are marked by the parentheses in Equation 5.2, 5.3 and 5.4 and still have the same semantic.

- **TP in the numerator:** Is tried to be maximized in order to correctly cover as many relevant labels as possible
- **TP in the denominator:** Is equal to the numerator
- **FN in the denominator:** Is tried to be minimized in order to mistakenly miss as few relevant labels as possible

The new definition of the confusion matrix results in different definitions of Recall for the above described scenarios.

- **Classic Recall (cf. Table 5.3):**

$$\delta_{rec}^{classic} = \frac{C_{RP}}{(C_{RP}) + (C_{RN} + U_{RN} + U_{RP})} \tag{5.2}$$

- **Adapted Recall (cf. Table 5.4):**

$$\delta_{rec}^{adapted} = \frac{C_{RP} + C_{IN}}{(C_{RP} + C_{IN}) + (U_{RN} + U_{RP})} \tag{5.3}$$

- **Prediction-Dependent Recall (cf. Table 5.5):**

$$\delta_{rec}^{prediction-dependent} = \frac{C_{RP} + C_{IN}}{(C_{RP} + C_{IN}) + (U_{IN} + U_{RP})} \tag{5.4}$$

They can be renamed similarly by simply exchanging the names according to Table 5.6. Accordingly, the following equation shows the definition of prediction-dependent Precision with the new naming:

$$\delta_{prec} = \frac{TP}{TP + FP} \Rightarrow \delta_{prec}^{prediction-dependent} = \frac{C_{IN} + C_{RP}}{(C_{IN} + C_{RP}) + (C_{IP} + C_{RN})} \tag{5.5}$$

In the remainder of this work, F-measure is used with prediction-dependent Precision combined with one version of Recall, i.e. $\delta_{rec}^{classic}$, $\delta_{rec}^{adapted}$ or $\delta_{rec}^{prediction-dependent}$.

6. Evaluation

In this chapter, we evaluate the quality of the rules learned by the proposed bottom-up algorithm. As described in Chapter 5, we have three different versions of Recall – classic, adapted and prediction-dependent –, which can be used to guide the refinement of a rule as well as for the evaluation of a rule or model. Thus, in the first section of this chapter, we learn rules using the prediction-dependent measure, the classic measure and the adapted measure. In order to compare the generalization behavior of these three variants, we evaluate the rules with the prediction-dependent metric. In the second section, we compare the models learned with the different evaluation methods according to their classic model evaluation in order to see how the proposed bottom-up algorithm performs in the classic classification setting. However, we do not expect the bottom-up algorithm to be able to compete with the state-of-the-art Ripper algorithm using the binary relevance transformation method, because it employs several optimization techniques that our approach lacks [Cohen, 1995]. Instead, we consider Ripper's performance as an upper bound to ease the interpretation of the results. As an implementation of Ripper, we use JRip provided by the Weka framework[1]. The implementation of the bottom-up separate-and-conquer algorithm as well as the experiments use the SeCo-MLC framework[2]. As the prediction-dependent Precision has successfully been used in Rapp et al. [2018], and because Recall is crucial to the generalization behavior in the bottom-up algorithm, prediction-dependent Precision is used for calculating the F-measure in all of the experiments. Only for Recall, the different evaluation metrics are used. In order to get comparable results, the order of choosing random instances in one graph is defined by the same seed. All of the experiments are executed on three data sets from Mulan[3] in order to compare results for sparse and dense as well as for nominal and numeric data sets (cf. Table 6.1). In case of the data set Emotions, the bottom-up algorithm did not finish with $\beta = 0$ due to memory resource restrictions. However, this does not further

[1] https://sourceforge.net/projects/weka/
[2] https://github.com/keelm/SeCo-MLC
[3] http://mulan.sourceforge.net/datasets-mlc.html

affect the evaluation, since Recall is not taken into account for F-measure in this case and especially increasing values for β are interesting.

Table 6.1.: Data sets used for evaluation [Tsoumakas et al., 2011]

name	instances	nominal	numeric	labels	cardinality	density	distinct
Emotions	593	0	72	6	1.869	0.311	27
Flags	194	9	10	7	3.392	0.485	54
Genbase	662	1186	0	27	1.252	0.046	32

6.1. Analysis of Rule Generalization

6.1.1. Prediction-Dependent Optimization

The following graphs (Figure 6.1, Figure 6.2 and Figure 6.3) show the results of using the F-measure as heuristic for generalizing the rules, where both Recall and Precision are used according to the prediction-dependent metric. The plots show the average heuristic value of the rules, where these are evaluated according to prediction-dependent Precision and Recall. Additionally, the average coverage of the rules, i.e. the number of examples they cover, is shown. It is expected to behave similar to the Recall of the rules. The rules are learned with different values for β. The parameter β trades off Precision and Recall. If $\beta < 1$, F-measure focuses on Precision. As β increases, more emphasis is put on Recall. As a result, we expect the average Recall of the rules to increase with larger values of β.

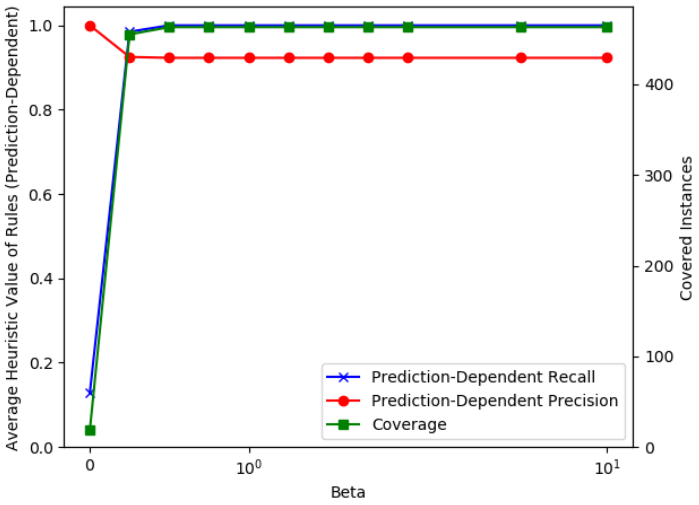

Figure 6.1.: Averaged heuristic values of the rules learned on the data set Genbase using prediction-dependent Recall for rule induction.

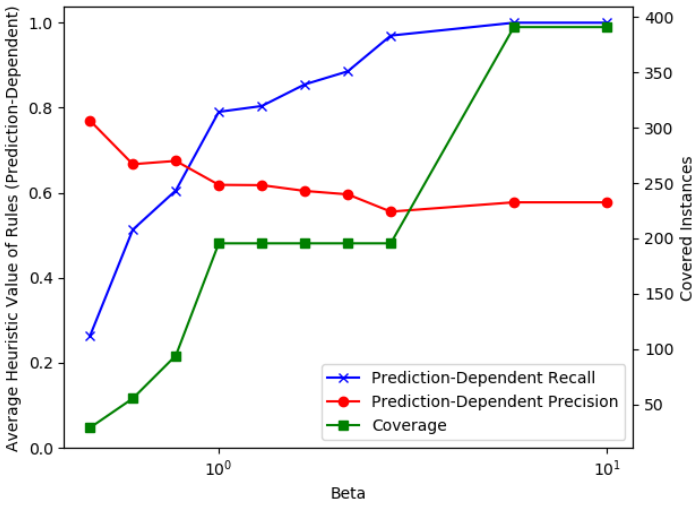

Figure 6.2.: Averaged heuristic values of the rules learned on the data set Emotions using prediction-dependent Recall for rule induction.

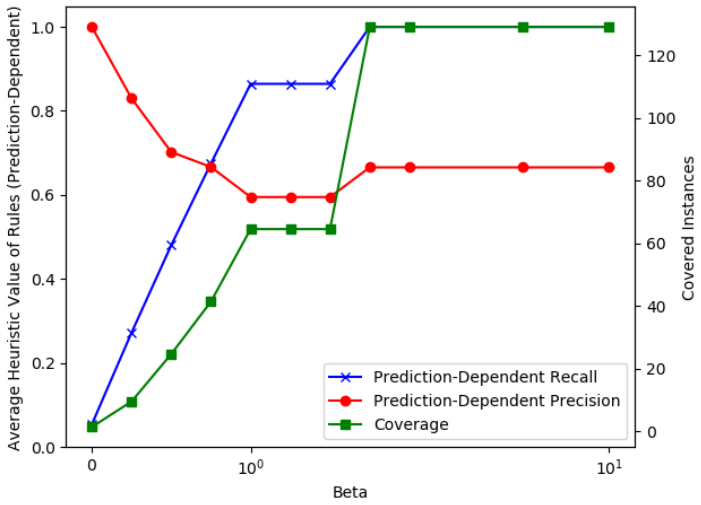

Figure 6.3.: Averaged heuristic values of the rules learned on the data set Flags using prediction-dependent Recall for rule induction.

On the data set Genbase, when using the prediction-dependent Recall for rule induction, the rules are generalized until the body is empty, almost independent from β (Figure 6.1). This means, that only one Default-Rule is learned, which predicts the labels of the randomly chosen instance for all instances. This might be due to the fact, that the Genbase data set only contains nominal attributes. As opposed to numeric attributes, nominal conditions can not be generalized as fine-granular, but can only be removed entirely. Thus, the removal of a nominal condition might lead to an enormous increase of coverage of the rule and therefore an increase of Recall. Additionally, the Genbase data set is very sparse, i.e. it has a low label cardinality, which means that lots of negative labels are predicted correctly (C_{IN}) with the Default-Rule, which leads to a high Recall when using the prediction-dependent metric.

However, in case of the data sets Emotions (Figure 6.2) and Flags (Figure 6.3), the curves for Recall, Precision and Coverage look exactly as expected, such that with an increasing value for β, Recall and coverage increase whilst Precision is decreasing for the average rule. This shows, that the bottom-up algorithm can generalize the single rules very well when using the prediction-dependent metric. In the following this behavior is compared to the classic and adapted metric.

6.1.2. Classic Optimization

Figure 6.4 shows how the average rule performs when it is learned with the classic metric, but evaluated on the prediction-dependent metric. In the classic case, the rules hardly get generalized at all. That is, because the classic Recall can only be increased by correctly covering more positive labels (C_{RP}), but not by covering negative labels correctly (C_{IN}). As the head of the rules does not change, and mistakenly predicting a negative label (C_{RN}) reduces Recall, the Recall of the rules does not increase with increasing β. The behavior is comparable for the data set Genbase (Figure A.2), whereas the rules for the Flags data set get more general with increasing β (Figure A.1). However, the maximal Recall is already reached with $\beta < 1$, which means that it is harder to find a good trade-off between Precision and Recall in this case.

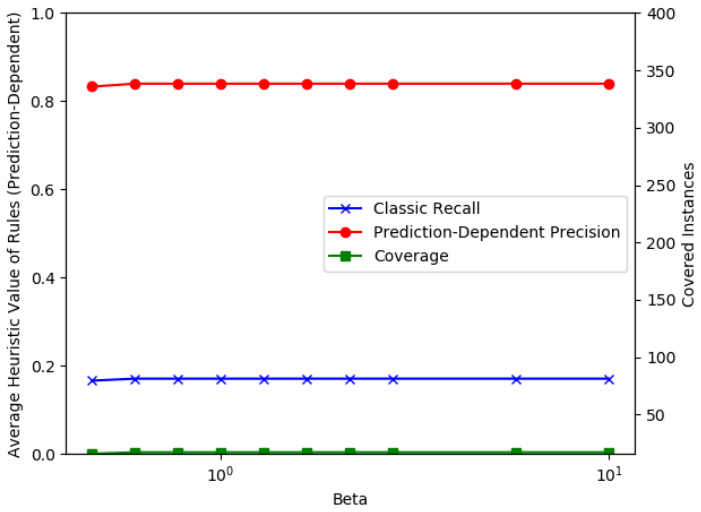

Figure 6.4.: Averaged heuristic values of the rules learned on the data set Emotions using classic Recall for rule induction.

6.1.3. Adapted Optimization

In the adapted case, all three data sets (Figure 6.5, Figure 6.6, Figure 6.7) behave comparable to the prediction-dependent case on the Emotions data set (cf. Figure 6.2). However, the steepness of the Recall curve is higher on the data sets Emotions and Flags, which

again shows, that the prediction-dependent refinement offers a more fine-granular regulation of the trade-off between Precision and Recall. This might be due to the fact, that with the prediction-dependent metric as opposed to the adapted metric, the predictions for uncovered but correctly negative predicted labels (U_{IN}) are taken into account for Recall. Thus, for the same β, the Recall value is lower than with the adapted metric. As for the data set Genbase, the rules are not generalized to a Default-Rule as in the prediction-dependent and classic optimization. Genbase is a sparse data set, which might indicate, that the prediction-dependent metric does have problems with a fine-granular generalization on such data sets. The reason might be, that a rule which covers many correctly predicted negative labels (C_{IN}), as it is likely to happen in a sparse data set, has an increased Recall, because all the correctly predicted negative labels increase its value, whilst mistakenly negative predicted labels (C_{RN}) do not aggravate Recall as they are counted as False Positives in this metric.

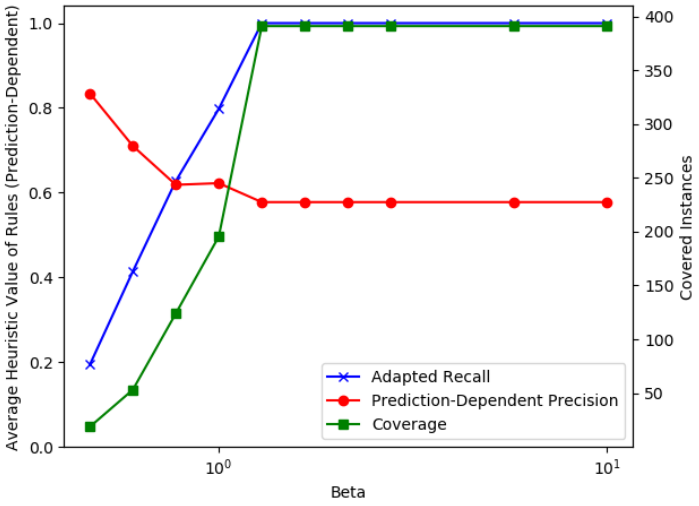

Figure 6.5.: Averaged heuristic values of the rules learned on the data set Emotions using adapted Recall for rule induction.

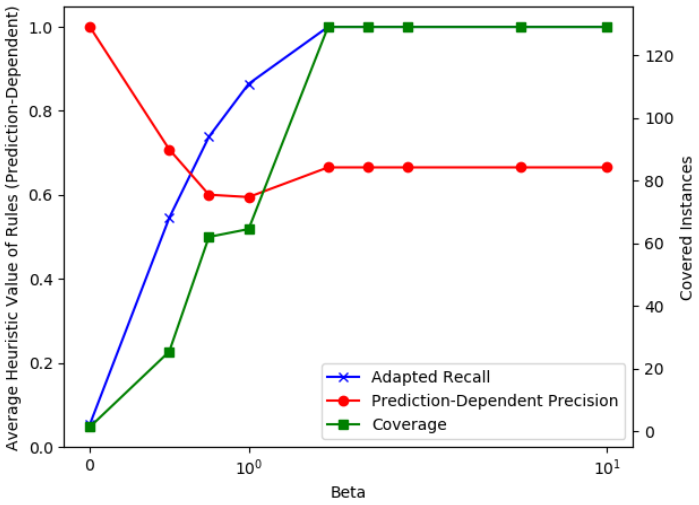

Figure 6.6.: Averaged heuristic values of the rules learned on the data set Flags using adapted Recall for rule induction.

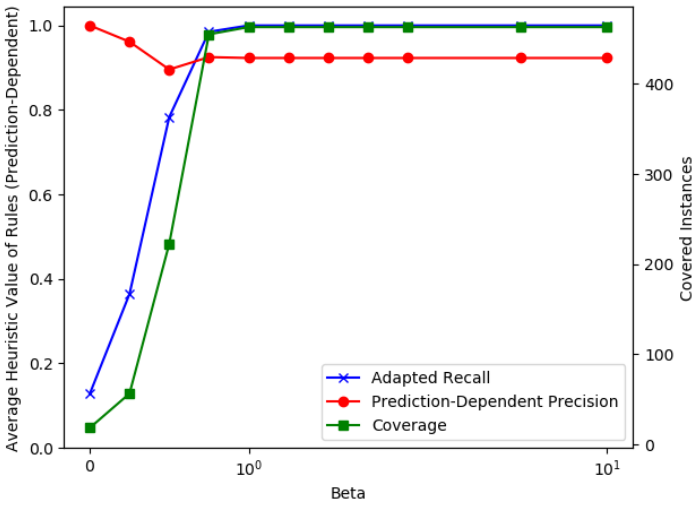

Figure 6.7.: Averaged heuristic values of the rules learned on the data set Genbase using adapted Recall for rule induction.

As a summary, we can see, that the average rules learned with the prediction-dependent method look very similar to what we want to achieve, at least when also evaluated with the prediction-dependent method. The trade-off between coverage and consistency can be tuned with the parameter β more fine-granluar than with the other metrics for single rules. However, in the end, the performance of the model with the classic metric is important, which is evaluated in the following chapter.

6.2. Model Comparison

In order to compare the bottom-up algorithm using the prediction-dependent evaluation method with a state-of-the-art multi-label classifier, we use binary relevance from the Java library Mulan[4], which then uses the JRip implementation in Weka of the Ripper algorithm [Cohen, 1995]. As both JRip and binary relevance are well-developed, the resulting model can be seen as an upper bound for the respective data set. Additionally, the Default-Rule created from the randomly sampled instance refers as a lower bound, as this is the simplest model that the bottom-up algorithm can learn. The performance of the bottom-up algorithm is not expected to be able to compete with JRip, as JRip uses pruning and further optimization of the model after training. Additionally, in this work, we focus on generalizing single rules, whereas for combining those to a good model, there are still plenty of optimizations for this bottom-up algorithm that are out of the scope of this work and remain for future work, especially the refinement of the head.

However, in the following sections, the different Recall metrics are used for calculating F-measure in order to refine the rules. But this time, not the average of the rules is shown, but the Recall, Precision and Subset Accuracy of the model, evaluated always with the classic metric on the test data set. While Recall and Precision are not expected to be able to compete with JRip in binary relevance, we look at Subset Accuracy, because the bottom-up algorithm is example-driven. That means, that a new rule is expected to have a high Subset Accuracy, since it predicts at least the correct label vector for the instance it was created from. This is not the case for the binary relevance Ripper algorithm, which learns rules separately for every label on its own and therefore does not necessarily predict existing combinations of labels.

6.2.1. Prediction-Dependent Optimization

Figure 6.8 shows the classic micro-averaged Recall of the model, which was learned with the prediction-dependent Recall in F-measure. As expected, JRip performs a lot better in terms of Recall. However, one reason therefor is, that the rules of the bottom-up algorithm are optimized using the prediction-dependent method and therefore are not optimized on what leads to good results in classic evaluation. That is, because correct predictions of negative labels for covered instances (C_{IN}) do not increase Recall in classic evaluation, but do with the prediction-dependent metric. Additionally, the classic evaluation counts

[4] http://mulan.sourceforge.net/

mistakenly negative predicted labels (C_{RN}) as False Negatives, which in the prediction-dependent evaluation is a False Positive. This might lead to the decrease of classic Recall with increasing β, because the FP from the prediction-dependent optimization are now counted as FN. On the data set Flags, the results are similar (Figure 6.11) and accordingly, on the data set Genbase, again almost instantly the Default-Rule is learned with the prediction-dependent method, which equals a very low Recall in classic evaluation (Figure 6.14).

The results for Subset Accuracy are not as expected (Figure 6.10, Figure 6.13 and Figure 6.16). Rather than having a high Subset Accuracy, which is the expectation when using an example-driven algorithm, the Ripper algorithm with binary relevance has a higher Subset Accuracy. As binary relevance only learns separated rules for each label, the Subset Accuracy of such learned models is not expected to be higher than the one of the bottom-up algorithm, which starts with label combinations that actually exist. One possible explanation is, that the rules are not generalized enough for small values of β, which can be seen for the data set Flags in Figure 6.11 where the Recall is low just as the Subset Accuracy in Figure 6.13. For higher values of β, the Subset Accuracy is the same as for the Default-Rule, because the learned model also just learns the Default-Rule. As the rules are not optimized on Subset Accuracy but on F-measure, the rules might be generalized to cover similar label combination, but predict them not exactly correct and therefore do not increase Subset Accuracy.

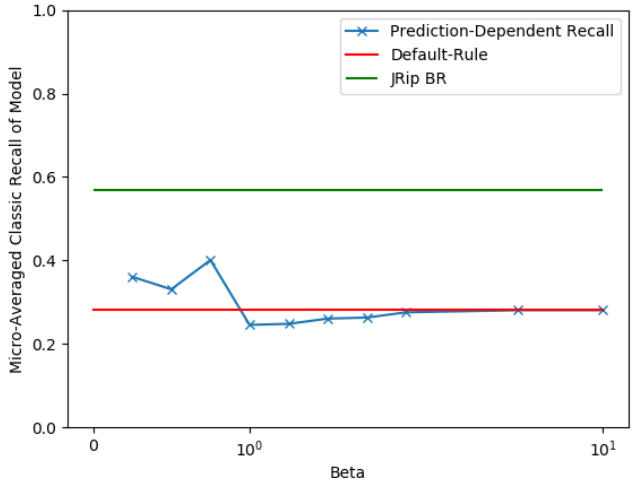

Figure 6.8.: Micro-averaged classic Recall of the model learned on the data set Emotions using prediction-dependent Recall for rule induction.

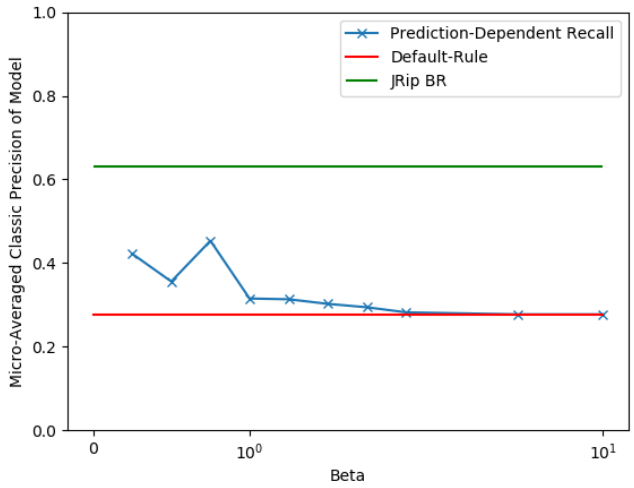

Figure 6.9.: Micro-averaged classic Precision of the model learned on the data set Emotions using prediction-dependent Recall for rule induction.

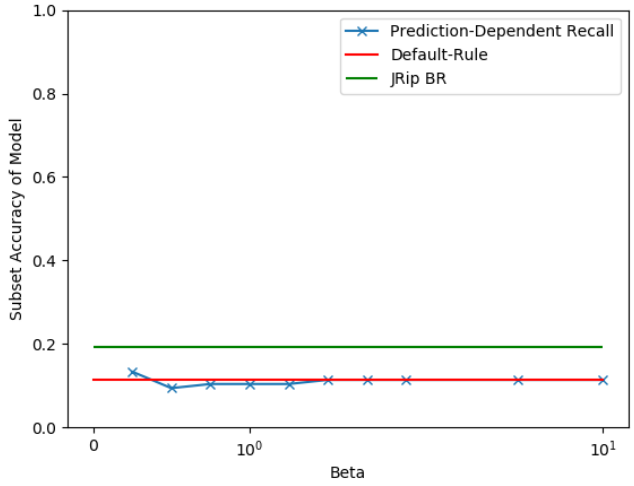

Figure 6.10.: Subset Accuracy of the model learned on the data set Emotions using prediction-dependent Recall for rule induction.

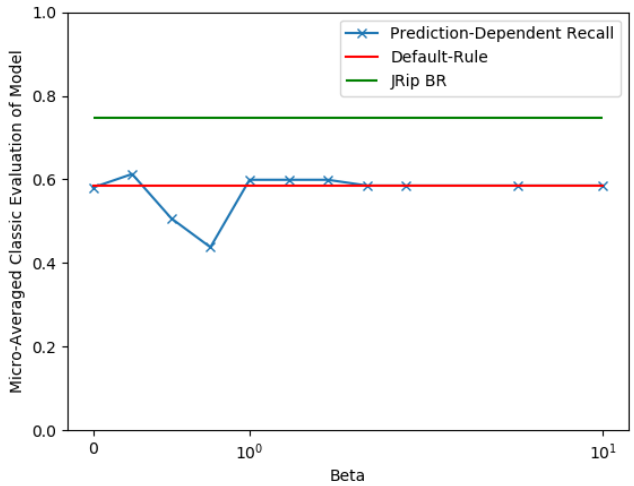

Figure 6.11.: Micro-averaged classic Recall of the model learned on the data set Flags using prediction-dependent Recall for rule induction.

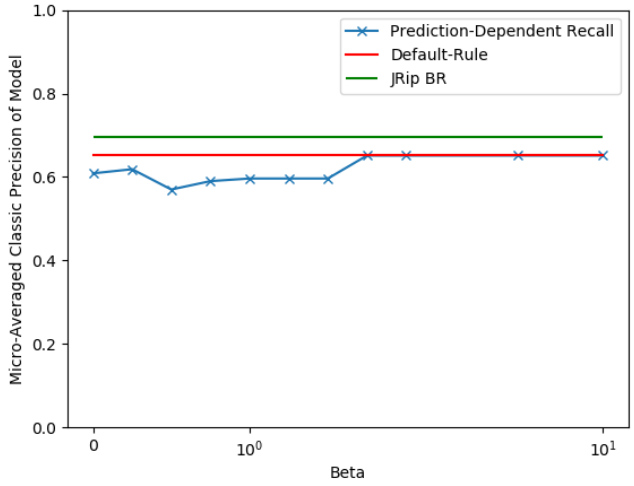

Figure 6.12.: Micro-averaged classic Precision of the model learned on the data set Flags using prediction-dependent Recall for rule induction.

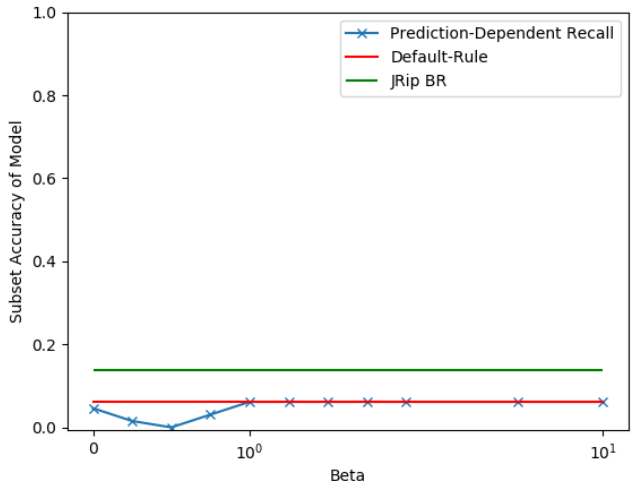

Figure 6.13.: Subset Accuracy of the model learned on the data set Flags using prediction-dependent Recall for rule induction.

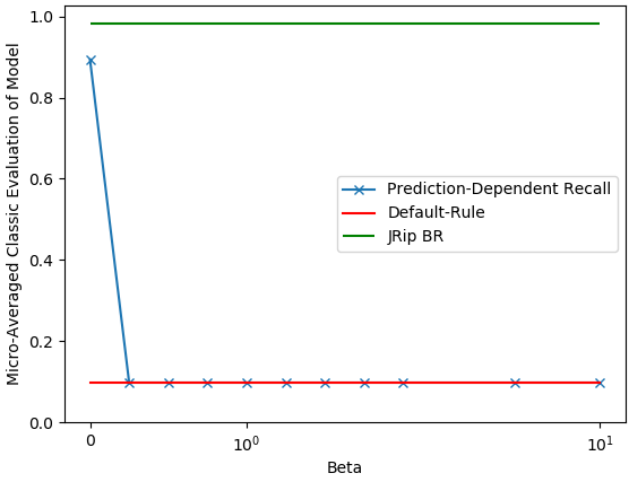

Figure 6.14.: Micro-averaged classic Recall of the model learned on the data set Genbase using prediction-dependent Recall for rule induction.

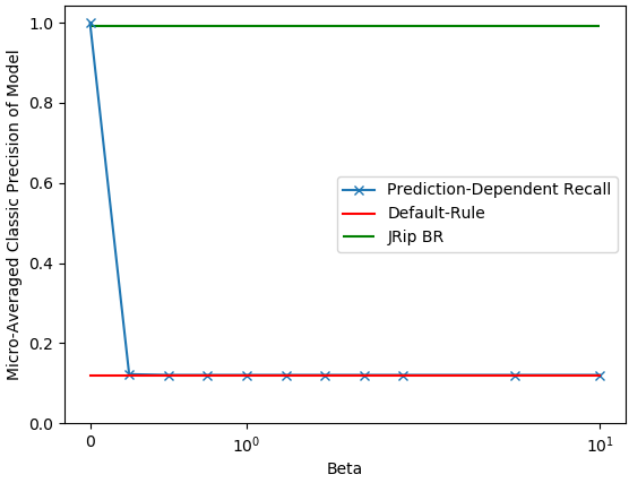

Figure 6.15.: Micro-averaged classic Precision of the model learned on the data set Genbase using prediction-dependent Recall for rule induction.

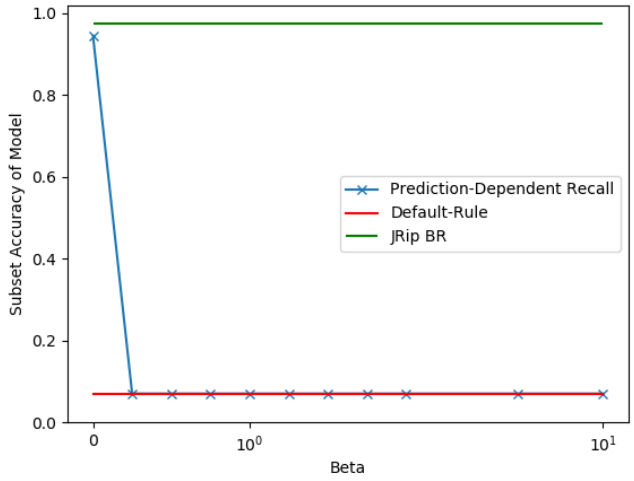

Figure 6.16.: Subset Accuracy of the model learned on the data set Genbase using prediction-dependent Recall for rule induction.

6.2.2. Adapted Optimization

The results for Recall and Precision of the models learned with F-measure using adapted Recall are mostly similar to those of the prediction-dependent optimization (Figure 6.17, Figure 6.20, Figure 6.23, Figure 6.18, Figure 6.21 and Figure 6.24). The reason might be, that the difference between the prediction-dependent and adapted metric concerning the behavior of Recall on uncovered instances only has a weak influence on how the algorithm performs on the classic evaluation method. Rather the changes applied to the behavior on covered instances is relevant in this case, because both the adapted and prediction-dependent method are not punished with lower Recall when mistakenly predicting a negative label for a covered instance (C_{RN}). Instead, the Precision decreases, which becomes less relevant with increasing β. Thus, the rules have high Recall with the adapted and prediction-dependent metric, but the classic evaluation counts the wrong negative predictions as False Negatives and therefore the Recall is lower.

Concerning Subset Accuracy, the adapted metric has the same problems as the prediction-dependent metric (Figure 6.19, Figure 6.22 and Figure 6.25). The Subset Accuracy of the Default-Rule is never beaten, because for $\beta < 1$, the rules are too specialized and

therefore do not cover all instances with the exact same label vector predicted by the rule. On the other hand, for $\beta > 1$ the model itself is reduced to only the Default-Rule and therefore has the same Subset Accuracy.

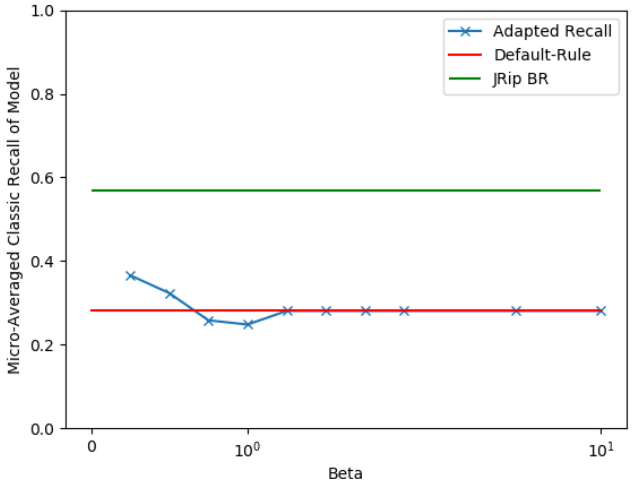

Figure 6.17.: Micro-averaged classic Recall of the model learned on the data set Emotions using adapted Recall for rule induction.

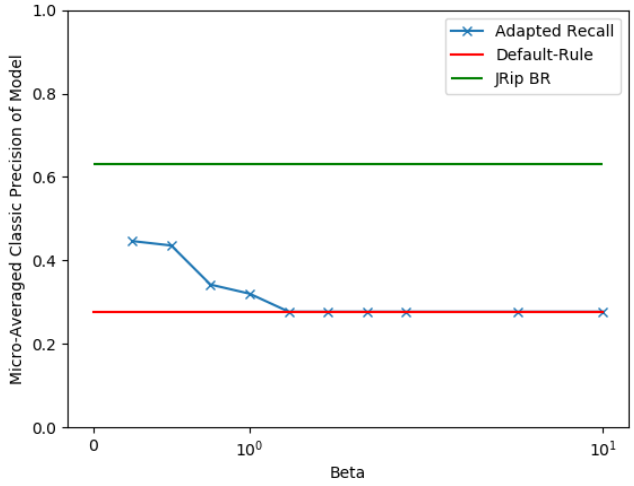

Figure 6.18.: Micro-averaged classic Precision of the model learned on the data set Emotions using adapted Recall for rule induction.

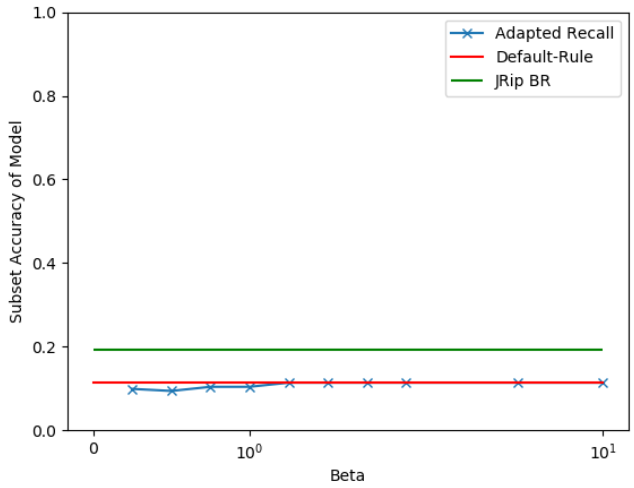

Figure 6.19.: Subset Accuracy of the model learned on the data set Emotions using adapted Recall for rule induction.

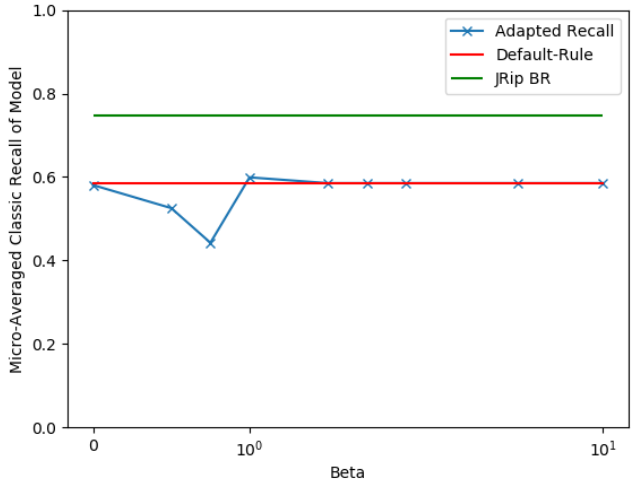

Figure 6.20.: Micro-averaged classic Recall of the model learned on the data set Flags using adapted Recall for rule induction.

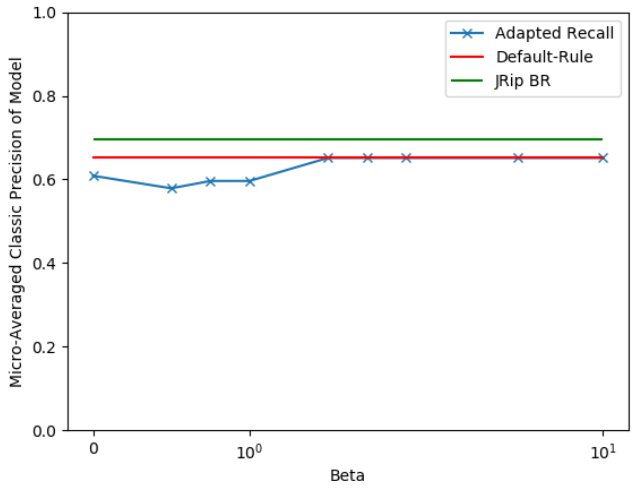

Figure 6.21.: Micro-averaged classic Precision of the model learned on the data set Flags using adapted Recall for rule induction.

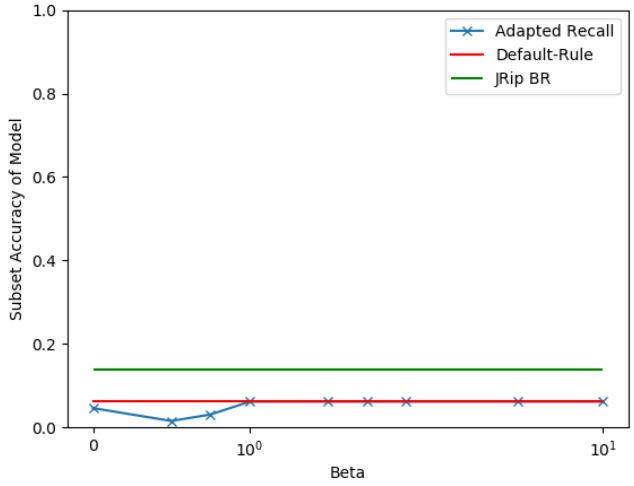

Figure 6.22.: Subset Accuracy of the model learned on the data set Flags using adapted Recall for rule induction.

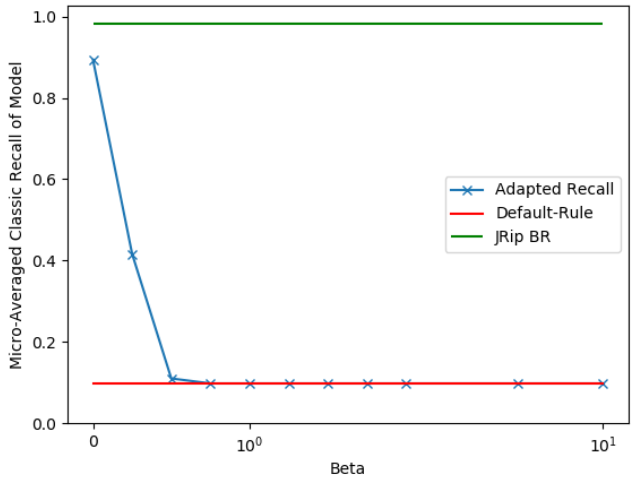

Figure 6.23.: Micro-averaged classic Recall of the model learned on the data set Genbase using adapted Recall for rule induction.

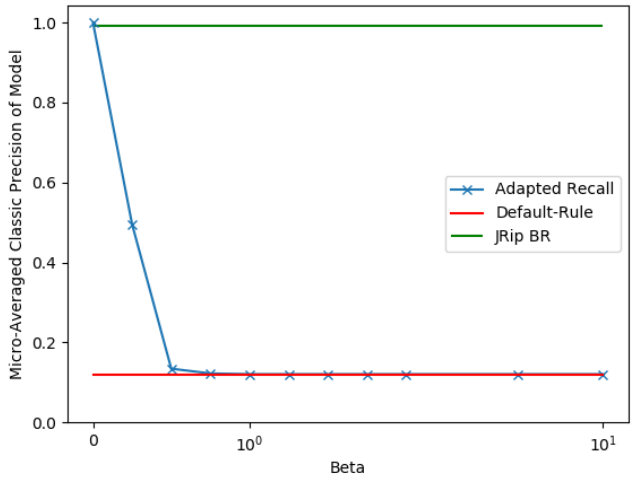

Figure 6.24.: Micro-averaged classic Precision of the model learned on the data set Genbase using adapted Recall for rule induction.

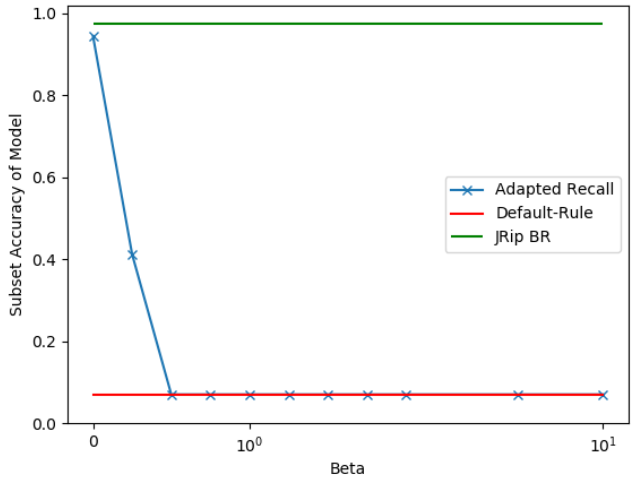

Figure 6.25.: Subset Accuracy of the model learned on the data set Genbase using adapted Recall for rule induction.

6.2.3. Classic Optimization

When using classic Recall for optimizing the rules, the results for Recall and Precision are better than with prediction-dependent and adapted Recall on the classic evaluation metric (Figure 6.26, Figure 6.29, Figure 6.32, Figure 6.27, Figure 6.30 and Figure 6.33). That is, because the model consists of rules that are optimized such that they have high heuristic values in the classic evaluation metric. However, the Subset Accuracy of the bottom-up algorithm again is way lower than the Subset Accuracy of JRip, which likely has the same cause as in Subsection 6.2.1 and Subsection 6.2.2 (cf. Figure 6.28, Figure 6.31 and Figure 6.34).

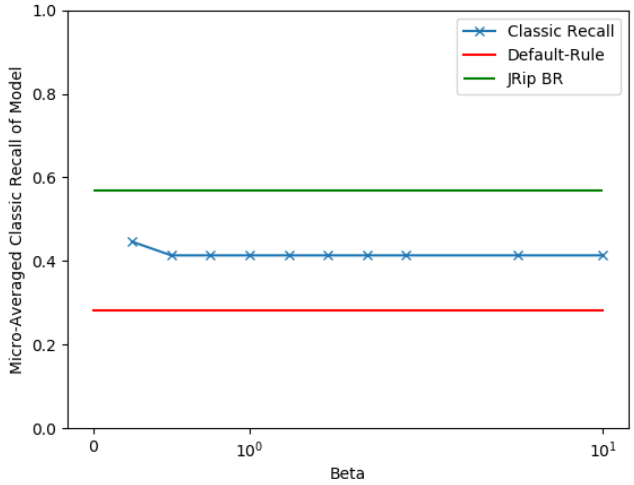

Figure 6.26.: Micro-averaged classic Recall of the model learned on the data set Emotions using classic Recall for rule induction.

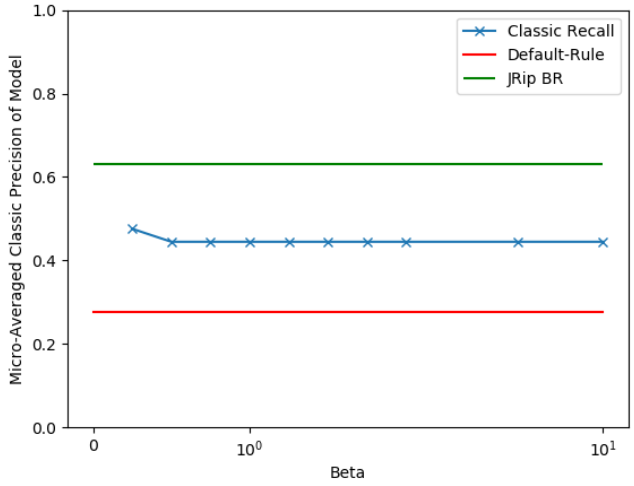

Figure 6.27.: Micro-averaged classic Precision of the model learned on the data set Emotions using classic Recall for rule induction.

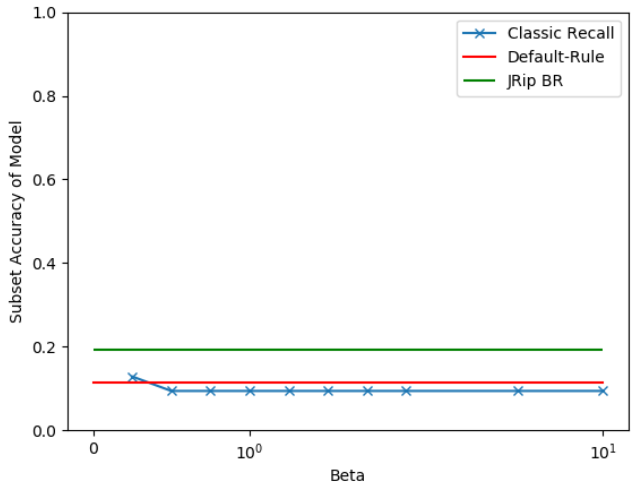

Figure 6.28.: Subset Accuracy of the model learned on the data set Emotions using classic Recall for rule induction.

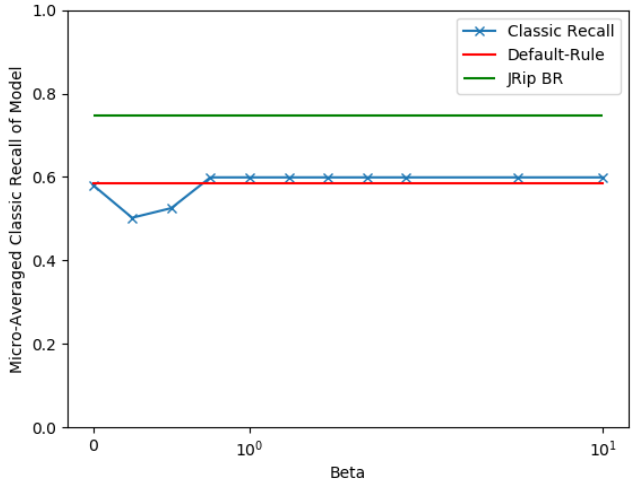

Figure 6.29.: Micro-averaged classic Recall of the model learned on the data set Flags using classic Recall for rule induction.

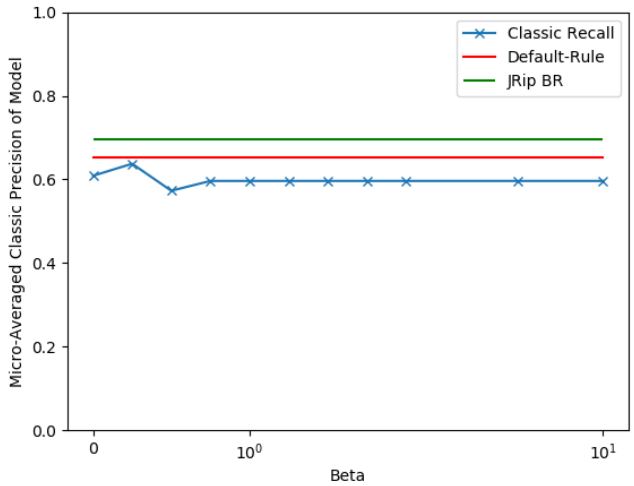

Figure 6.30.: Micro-averaged classic Precision of the model learned on the data set Flags using classic Recall for rule induction.

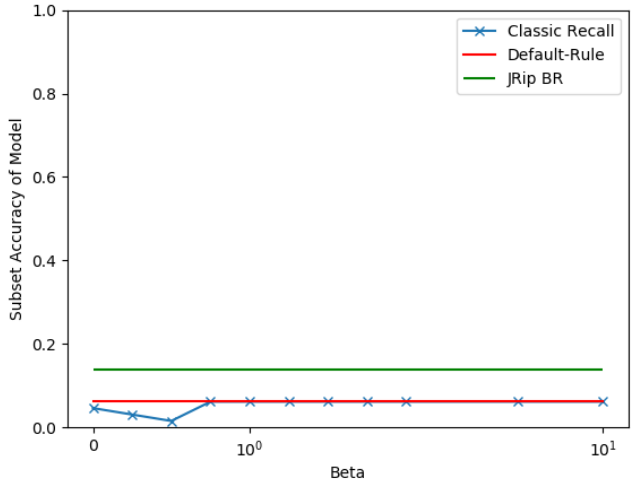

Figure 6.31.: Subset Accuracy of the model learned on the data set Flags using classic Recall for rule induction.

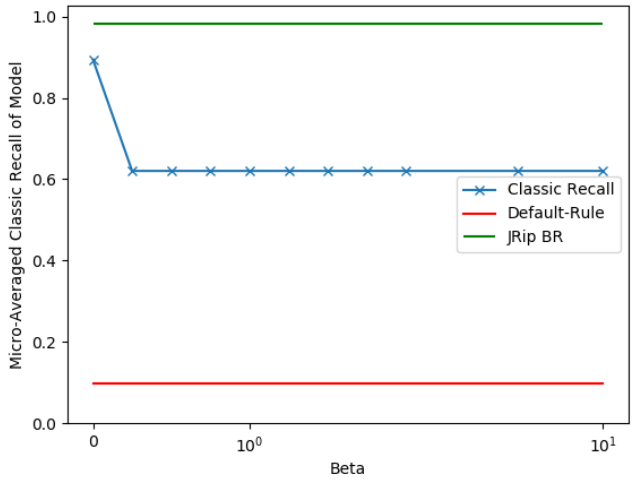

Figure 6.32.: Micro-averaged classic Recall of the model learned on the data set Genbase using classic Recall for rule induction.

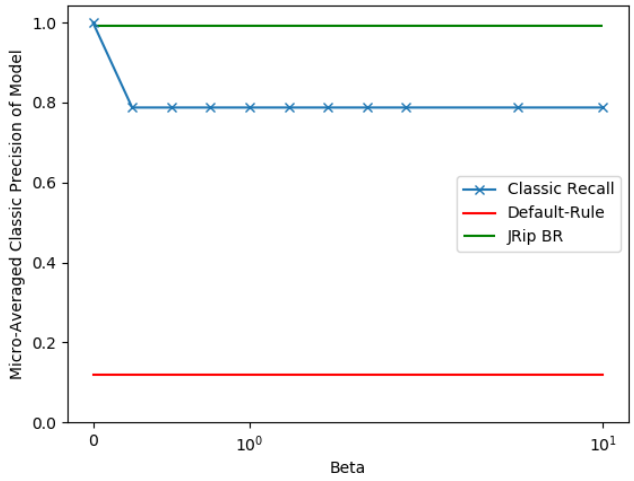

Figure 6.33.: Micro-averaged classic Precision of the model learned on the data set Genbase using classic Recall for rule induction.

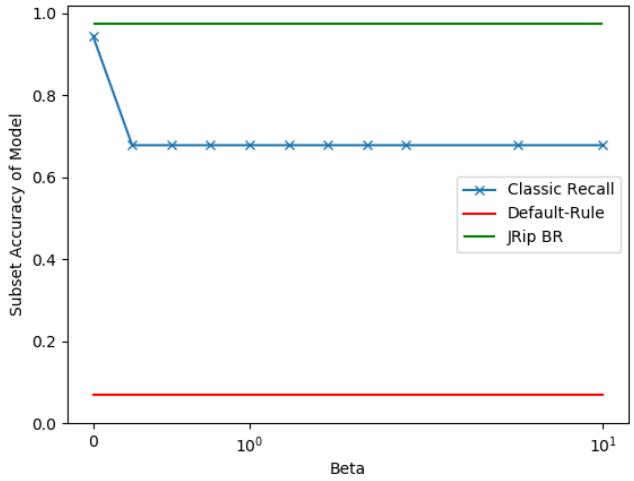

Figure 6.34.: Subset Accuracy of the model learned on the data set Genbase using classic Recall for rule induction.

6.3. Summary of Experimental Results

We evaluated a basic version of a bottom-up algorithm for multi-label classification with rule induction. The lack of performance of the models in classic evaluation can be partially explained with the mistakenly negative predictions for positive labels (C_{RN}) due to the fact that oftentimes the instances, which the heads are created from, contain many negative labels, but there is still plenty of future work to do in order to be able to compete with state-of-the-art classifiers. Some possible extensions are described in the following chapter, but also further investigation on when the algorithm performs better has to be done, such as nominal or numeric data sets. Additionally, investigations on how other state-of-the-art algorithms for multi-label classification [Loza Mencía and Janssen, 2016] perform with the proposed prediction-dependent evaluation metric, in order to get more evidence that the classic evaluation metric is not always reasonable in multi-label classification settings.

7. Extensions of the Bottom-Up Algorithm

In this chapter, a brief overview of some extensions that can be applied to the introduced standard bottom-up algorithm is given. These approaches can be considered in future work, as they might be relevant for enhancing the results of the algorithm but not for creating a fundamentally working bottom-up implementation.

7.1. Random Generalization of Numeric Attributes

In the following, a different approach of refining a condition is described. The generalization of a numeric condition can be carried out by defining a new interval, which includes not only the next value existing in the training data, but is defined by taking a random step size. Possible outcomes are the same as in a one step generalization, but also an interval covering all higher respectively lower values that occur in the training set for this attribute can be created. This variation possibly misses a more fine-granular generalization of the rule, but also leads to a speed-up of the generalization and has the advantages of random search, such as better exploration and avoidance of dead ends. In order to achieve a very strong generalization, even a complete removal of numeric conditions, according to the generalization of nominal conditions, can be considered.

7.2. Random Rule Generalization

The above mentioned changes can also be extended to randomly choosing only one of all possible generalizations of the rule without computing all other possibilities. For a rule with n conditions, this means that instead of n possible generalizations, only one random generalization has to be computed and compared to the original rule. Besides

the increase of performance, this simplification offers the advantages of random search as mentioned above.

7.3. Generalization Problems

As investigated by Fürnkranz [2002] for binary classification, a rule can not be generalized in one iteration to cover more instances, if its associated instance differs from all other training examples in more than one attribute. That is, because every candidate rule for the generalization only changes one condition and therefore does not cover the nearest neighbor or any other examples afterwards. This can be extended to a rule and the nearest neighbor example differing in n attributes need n generalization steps in order to cover more examples.

N-Step look-ahead is one of the approaches investigated by Fürnkranz [2002] in order to avoid this problem. Instead of generalizing only one condition in each iteration, n conditions can be refined. However, an increasing n comes with a quadratic increase of the number of possible generalizations and Fürnkranz [2002] showed, that a 2-step look-ahead only slightly improves the result. Therefore, a more promising approach was proposed, i.e. not to randomly choose a training instance for a new rule, but to search for an instance with neighbors that do not differ too much from it. This approach is also left for future work in the multi-label classification setting with bottom-up search.

8. Conclusion

8.1. Summary

In this work, we chose a separate-and-conquer algorithm using bottom-up search to learn multi-label rules from training data, which can then make predictions for new data. We specified the behavior of the algorithm for nominal and numeric attributes, defined how to create and refine rules and investigated different methods for evaluating rules. Finally, a new evaluation metric was proposed in order to fulfill the requirements of multi-label classification, in combination with a new naming for it. The evaluation of the standard bottom-up algorithm shows, that there is still a lot of future work to do in terms of multi-label rule learning using bottom-up search. State-of-the-art classifiers using e.g. problem transformation methods as J-Rip with binary relevance produce models with higher consistency and coverage at least in the classical evaluation setting. However, the aim of this work was not to create a state-of-the-art multi-label classifier, but rather to find a good standard algorithm that can use bottom-up search to generalize rules until a certain degree of coverage is reached, whilst maintaining consistency as much as possible. Additionally, we could show, that the single rules learned by the bottom-up algorithm show the expected behavior when using the proposed prediction-dependent evaluation metric.

8.2. Future Work

In order to evaluate the expressiveness of the proposed prediction-dependent evaluation metric, more experiments have to made where state-of-the-art multi-label classifiers are optimized using the prediction-dependent method. Further, some possible improvements to the bottom-up algorithm have been described in this work, which might increase its performance. That is, the two differing algorithms as described in Chapter 3, but also the

extensions as mentioned in Chapter 7. However, the most promising modification is to allow changes to the head of a rule, which is left for future work.

Bibliography

William W. Cohen. Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

Johannes Fürnkranz. Separate-and-Conquer Rule Learning. *Artificial Intelligence Review*, 13(1):3–54, 1999. ISSN 02692821. doi: 10.1023/A:1006524209794.

Johannes Fürnkranz. A pathology of bottom-up hill-climbing in inductive rule learning. In *International Conference on Algorithmic Learning Theory*, pages 263–277. Springer, 2002.

Frederik Janssen. *Heuristic rule learning*. PhD thesis, Darmstadt University of Technology, Germany, 2012. URL `http://tuprints.ulb.tu-darmstadt.de/3135/`.

Eneldo Loza Mencía. *Efficient Pairwise Multilabel Classification*. Dissertation, Technische Universität Darmstadt, Knowledge Engineering Group, July 2012. URL `http://tuprints.ulb.tu-darmstadt.de/3226/7/loza12diss.pdf`. submitted on 2012-06-10, defended on 2012-07-24.

Eneldo Loza Mencía and Frederik Janssen. Learning rules for multi-label classification: a stacking and a separate-and-conquer approach. *Machine Learning*, 105(1):77–126, 2016. ISSN 15730565. doi: 10.1007/s10994-016-5552-1.

M Rapp. A separate-and-conquer algorithm for learning multi-label head rules. Master's thesis, TU Darmstadt, Knowledge Engineering Group, 2016.

Michael Rapp, Eneldo Loza Mencía, and Johannes Fürnkranz. Exploiting anti-monotonicity of multi-label evaluation measures for inducing multi-label rules. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 29–42. Springer, 2018.

Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

Grigorios Tsoumakas, Eleftherios Spyromitros-Xioufis, Jozef Vilcek, and Ioannis Vlahavas. Mulan: A java library for multi-label learning. *J. Mach. Learn. Res.*, 12:2411–2414, July 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2021078.

A. Appendix

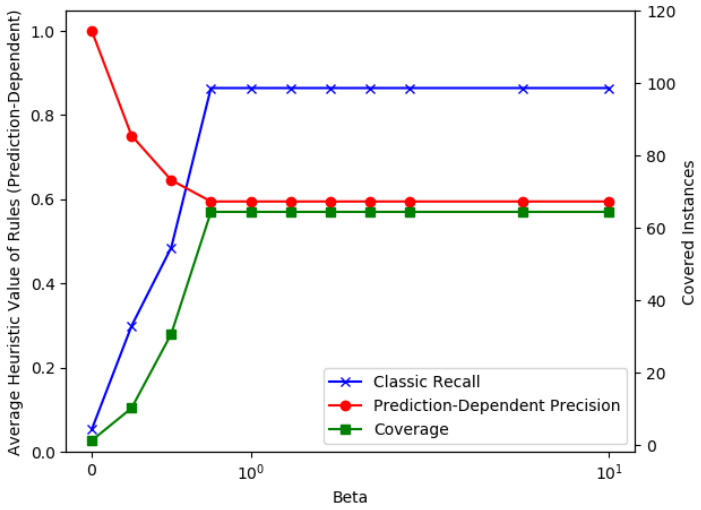

Figure A.1.: Averaged heuristic values of the rules learned on the data set Flags using classic Recall for rule induction.

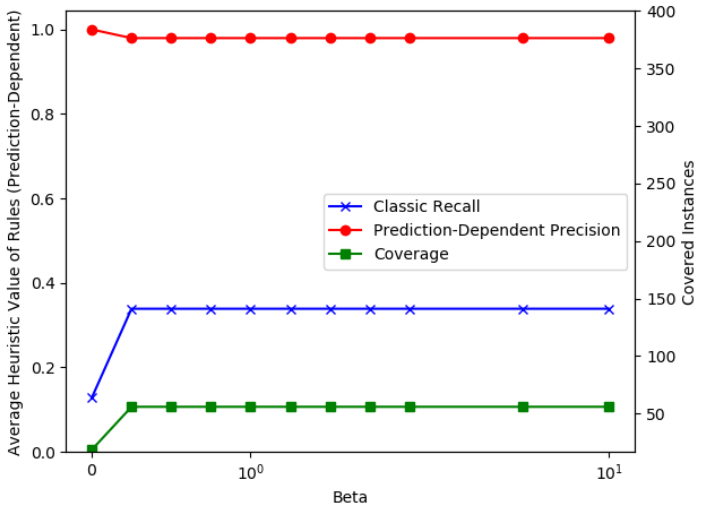

Figure A.2.: Averaged heuristic values of the rules learned on the data set Genbase using classic Recall for rule induction.