Discovery of Multilabel Rules by Relaxed Pruning

Entdeckung von Multilabel Regeln mit relaxiertem Pruning
Bachelor-Thesis von Yannik Klein aus Stockstadt am Main
Tag der Einreichung:

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dr. Eneldo Loza Mencía

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Department of Computer Science
Knowledge Engineering Group

Discovery of Multilabel Rules by Relaxed Pruning
Entdeckung von Multilabel Regeln mit relaxiertem Pruning

Vorgelegte Bachelor-Thesis von Yannik Klein aus Stockstadt am Main

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dr. Eneldo Loza Mencía

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, Yannik Klein, die vorliegende Bachelor-Thesis entsprechend §22 Abs. 7 APB der TU Darmstadt 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 Falle 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.

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

Darmstadt, den 20. Oktober 2018

(Yannik Klein)

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

In machine learning, classification is the task of accurately assigning classes to yet unseen examples by deducting information from available training data. Contrary to binary and multiclass classification, where each example is only associated with a single class, each class can be associated with a subset of all classes in multilabel classification. Popular application areas include tagging of texts with keywords, biology and multimedia. For instance, the identification of scenes or objects in images – each image may contain several scenes or objects. A distinguishing feature of multilabel classification are the possible dependencies between labels. It is a common assumption that the predictive performance of algorithms may greatly benefit from exploiting such dependencies. For instance, the presence of a label *Genre-Best-Rock* implies the presence of label *Genre-Rock*. By revealing such dependencies the understanding of the data set is improved. Hence, rules of the form *head* ← *body* are suitable for making label dependencies explicit. While rules do not achieve the same performance as other approaches, they can be easily interpreted by humans. Typically, common approaches to multilabel classification learn only rules that make a single label prediction. However, dependencies can be lost and it can be argued that full expressiveness can only be achieved by learning rules that may predict more than one label at a time. Rapp [2016] proposes an algorithm that is capable of directly learning such *multilabel rules*. The idea of the algorithm is to search the best performing head for a given body. For doing so, all possible label combinations have to be evaluated in theory, which grow exponentially, however. Hence, Rapp [2016] proves anti-monotonicity and decomposability of multilabel evaluation metrics and exploits them in the search in order to greatly increase efficiency. However, typically very few multilabel rules are learned using this approach as a result of the strict definition of best performing head. Thus, this work relaxes the pruning constraints of anti-montonicity and decomposability by tolerating a certain decline in the heuristic value for rules that contain more labels in the head. The theorized hope was to induce a more predictive model and reveal more label dependencies. The primary contribution of this work is the definition of relaxing the pruning by lifting the label search space using *relaxation lift functions* and adjusting the algorithm to suit the changes. For instance, by lifting the performance of heads with more labels in the head, the original algorithm for pruning cannot be utilized anymore. Therefore, we present an alternative way of pruning in this work that is based on anti-monotonicity and decomposability. Additionally, we refine the algorithm by introducing other changes as well. We also extensively study the effects of the made changes in the form of a sensitivity analysis, which aims at qualifying the effects of the extent of the lift and the different relaxation lift functions, and in the evaluation, which compares the learned models and their predictive performance to the approach by Rapp [2016]. In most cases, the learned models seem to be an improvement – in terms of expressiveness and predictive performance. Furthermore, we identify good settings for the extent of the lift. Additionally, relaxing the pruning typically results, contrary to initial expectation, in a shorter run time, despite more label combinations needed to be evaluated. The reason being that fewer rules are required to cover the training data if each rule predicts more labels. Nevertheless, it is unclear whether or not more label dependencies are learned by relaxed pruning. Hence, we also introduce an alternative algorithm that utilizes relaxed pruning. The idea is to initially assume all labels to be irrelevant and let the algorithm correct its previous predictions, including wrong predictions made by the algorithm. While only briefly evaluated, the approach seems to be promising and especially induces more label dependencies. However, the algorithm still requires work as it does not employ a stopping criterion yet for instance, which is left for future work.

Zusammenfassung

In maschinellem Lernen, Klassifikation beschäftigt sich mit der Aufgabe bisher ungesehenen Beispielen mit Hilfe von verfügbaren Trainingsdaten akkurat eine Klasse zuzuordnen. Im Gegensatz zur binären und Multiklassen Klassifikation, bei denen ein Beispiel lediglich mit einer Klasse in Verbindung gebracht werden kann, kann ein Beispiel bei Multilabel Klassifikation mit einer Teilmenge aller Klassen assoziiert werden. Beliebte Anwendungsgebiete sind unter anderem das Markieren von Texten mit Schlagwörtern, Biologie und Multimedia. Ein Beispiel ist die Identifikation von Szenen oder Objekten in Bildern – jedes Bild könnte mehrere Szenen oder Objekte enthalten. Ein Alleinstellungsmerkmal von Multilabel Klassifikation sind die möglichen Abhängigkeiten zwischen Labels. Es ist eine weit verbreitete Annahme, dass die Vorhersagegenauigkeit der Algorithmen stark verbessert werden kann, indem solche Abhängigkeiten ausgenutzt werden. Zum Beispiel impliziert das Vorhandensein eines Labels *Genre-Best-Rock* das Vorhandensein eines Labels *Genre-Rock*. Durch die Aufdeckung solcher Abhängigkeiten wird zudem das Verständnis des Datensatzes verbessert. Demnach eignen sich Regeln der Form *Vorhersage* ← *Bedingung* gut, um die Abhängigkeiten zwischen Labels explizit zu machen. Zwar erreichen Regeln nicht die gleiche Vorhersagegenauigkeit wie andere Herangehensweisen, jedoch können diese leicht von Menschen interpretiert werden. Gängige Herangehensweisen an das Problem lernen typischerweise nur Regeln, welche lediglich eine Label-Vorhersage treffen. Dadurch gehen aber oft die Abhängigkeiten verloren und es kann argumentiert werden, dass volle Aussagekraft nur von Regeln erreicht werden kann, welche auch mehrere Labels vorhersagen können. Rapp [2016] schlägt einen Algorithmus vor, der in der Lage ist, solche *Multilabel Regeln* direkt zu lernen. Die Idee des Algorithmus ist es, die beste Vorhersage in Form einer Label-Kombination für eine gegebene Bedingung zu suchen. In der Theorie müssen dazu alle Kombinationen getestet werden, welche jedoch exponentiell wachsen. Deswegen beweist Rapp [2016] die Anti-Monotonität und Zerlegbarkeit von Multilabel Evaluationsmetriken und nutzt diese aus, um die Suche stark zu beschleunigen. Leider werden durch dieses Vorgehen aufgrund der strikten Definition der besten Vorhersage typischerweise nur wenige Multilabel Regeln gelernt. Daher relaxiert diese Arbeit die Beschränkungen durch das Pruning, indem Regeln mit einem schlechteren Wert toleriert werden, falls diese mehr Labels vorhersagen. Das erhoffte Ergebnis war eine Regelmenge mit einer besseren Vorhersagegenauigkeit und mehr Abhängigkeiten zwischen Labels zu lernen. Demnach liegt der primäre Beitrag dieser Arbeit in der Definition der Relaxierung des Prunings, indem längere Vohersagen im Label-Suchraum mittels *Relaxierungsfunktionen* bevorzugt werden und die daraus folgende Anpassung des Algorithmus. Beispielsweise können die Label-Kombinationen nicht auf gleiche Weise gepruned werden, weswegen ein alternatives Vorgehen präsentiert wird, welches auf Anti-Monotonität und Zerlegbarkeit basiert. Zudem werden weitere Änderungen am Algorithms durchgeführt. Die Effekte der Änderungen werden gründlich in Form einer Sensitivitätsanalyse und einer Evaluation studiert. In den meisten Fällen sind die gelernten Regelmengen eine Verbesserung – in Bezug auf Aussagekraft und Vorhersagegenauigkeit. Zusätzlich werden gute Einstellungen für die Relaxierungsfunktionen identifiziert. Weiterhin resultiert das relaxierte Pruning, im Gegensatz zu den Erwartungen, meistens in einer kürzeren Laufzeit trotz der Evaluation von mehr Label-Kombinationen. Der Grund dafür ist, dass durch mehr Vorhersagen pro Regel weniger Regeln benötigt werden, um die Trainingsdaten abzudecken. Trotzdem werden nicht unbedingt mehr Abhängigkeiten zwischen Labels gelernt. Deswegen stellen wir einen weiteren Algorithmus vor, welcher relaxiertes Pruning verwendet. Die Idee dabei ist, am Anfang alle Labels als irrelevant anzunehmen und den Algorithmus die Vorhersagen, inklusive seiner eigenen falschen Vorhersagen, verbessern zu lassen. Obwohl dieses Vorgehen nur kurz evaluiert wird, scheint es vielversprechend zu sein. Insbesondere werden mehr Abhängigkeiten zwischen Labels gelernt. Trotz alledem muss an dem Verfahren weiterhin gearbeitet werden. Beispielsweise wird noch kein Abbruchkriterium verwendet, was jedoch zukünftigen Arbeiten überlassen wird.

1 Introduction

In this initial chapter, we give a short introduction to multilabel classification and the objectives of this work. Furthermore, the structure is outlined and a content overview of each chapter is given.

1.1 Revealing Label Dependencies by Learning Multilabel Rules

In machine learning, classification is the task of accurately assigning classes to yet unseen examples by deducting information from available training data. The simplest form aims at distinguishing only positive and negative examples of a single class, called *binary classification*. Let us assume we collect information about patients at a hospital. A binary classification problem may then be to decide whether or not the given patient has a certain disease, chickenpox for instance. We only decide whether he has the disease (positive example) or not (negative example). However, more than one disease exists. As a result, we would like to determine the disease our patient is infected with. Therefore, we determine the disease (out of several possible diseases) our patient is infected with. This type of setting is termed *multiclass classification*. Nevertheless, our patients can only have one disease at a time if using multiclass classification. Realistically, a patient may have several diseases at a time. Therefore, we should be able to assign several diseases to a patient as well. In cases such as this we talk about *multilabel classification*, which aims at assigning a subset of all available classes (diseases) to an example (patient).

While many well performing algorithms and tools for binary and multiclass classification exist, multilabel classification is not as well researched. Due to the many conventional algorithms in existence, the multilabel classification problem is usually transformed into multiple sub-problems, which can then be solved more easily. These approaches are called *problem transformation methods* [Tsoumakas and Katakis, 2007]. However, due to the transformation some problems arise, which are covered by Loza Mencía [2012] in more detail. In his work, Loza Mencía [2012] identifies the number of classes, also referred to as *labels* in the multilabel classification setting, as the main challenge (for pairwise learning). To outline this, just imagine an article in an online encyclopedia being tagged with keywords. In this case, every possible existing keyword is a label that the article could possibly be tagged with. Furthermore, learning algorithms may greatly benefit from revealing and exploiting dependencies between labels [Loza Mencía and Janssen, 2016]. These *label dependencies* are unique to the multilabel classification problem as they are not present in multiclass classification or binary classification. In our medical example this would imply dependencies between certain diseases. If one is present, another cannot be present simultaneously. Another disease might be more likely if another disease was previously identified or two diseases frequently co-occur. In an attempt to exploit label dependencies, Loza Mencía and Janssen [2016] have proposed an algorithm using a separate-and-conquer rule learning approach. Using this concept as a basis, Rapp [2016] has introduced a refinement of this algorithm, which learns multilabel rules directly by finding the best performing head for a given body. Moreover, Rapp et al. [2018] have proposed exploiting anti-monotonicity of multilabel evaluation measures for pruning the search space and therefore finding the best head more efficiently. However, typically not many multilabel rules are being induced using this approach, unless using precision as a base learner, as pointed out by Rapp [2016]. Hence, this work aims at relaxing the anti-monotonic and decomposable pruning constraints by introducing a bias towards multilabel rules. The intention of this approach is to exploit label dependencies more effectively and learn better models by inducing more multilabel instead of single-label rules. Due to the relaxed pruning, efficiency must also be considered.

1.2 Organization of the Work

In this section, the structure of the work at hand is outlined. Therefore, we briefly describe the contents and objectives of each chapter:

- **Chapter 2:** This chapter introduces the fundamentals of multilabel classification and inductive rule learning. At first, the multilabel classification problem is described formally. Following up, prob-

lem transformation methods are presented as a way of solving a multilabel classification problem. Furthermore, the importance and different types of dependencies between labels in the multilabel classification setting are outlined. Afterwards, separate-and-conquer rule learning, different types of multilabel rules and multilabel decision lists are covered. For evaluating the learned rules, we present several multilabel evaluation metrics. Finally, some related work is compared to our approach. Summing up, this chapter lays the foundations that are required for the following chapters. It also introduces the mathematical notations used throughout the entire work.

- **Chapter 3:** The presented algorithm requires to explore the label search space. As a result, chapter 3 presents two characteristics suggested by Rapp et al. [2018] that can be exploited in order to speed up the search through the label space – anti-monotonicity and decomposability. We further list which of the multilabel evaluation metrics presented in chapter 2 employ those characteristics.

- **Chapter 4:** This chapter explains the rule learning algorithm proposed by Rapp [2016], which this works aims at refining, in detail. We also briefly mention the original algorithm by Loza Mencía and Janssen [2016], which Rapp [2016] builds upon. The chapter explains the rule learning process, how the label search space can be explored while exploiting anti-monotonicity and decomposability as well as different approaches to measuring the performance of multilabel rules.

- **Chapter 5:** After presenting the algorithm that this work is based on in chapter 4, we explain the need for using a relaxed pruning approach in chapter 5. The basic idea is to allow the induction of multilabel rules if the quality of the rule does not decline significantly. Furthermore, we suggest a way of incorporating this relaxed pruning approach into the previously presented algorithm. Additionally, three different functions that define the extent of the allowed decline in the heuristic value are suggested and theoretically compared.

- **Chapter 6:** As a result of relaxing the pruning, anti-monotonicity and decomposability, which were introduced in chapter 3, cannot be used in the same way for speeding up the search through the label space. In chapter 6, we give counter-examples of why this is not possible and implement an alternative way of pruning the label search space that relies on the previously introduced characteristics of multilabel evaluation metrics. Furthermore, other changes to the algorithm are presented.

- **Chapter 7:** In this chapter we study the effects of relaxing the pruning. For doing so, we perform the rule learning process with different parameter settings on a data set and thoroughly compare the obtained results. The sensitivity analysis aims at getting more insight on how the parameters of the algorithm influence the performance and model characteristics of the adjusted approach.

- **Chapter 8:** As the relaxed pruning approach does not entirely satisfy the intentions of the adjustments, we suggest and alternative approach called prepending. In this chapter, the motivation of the approach is explained and the required changes to the algorithm are outlined.

- **Chapter 9:** In order to evaluate the performance and the model characteristics of the relaxed pruning approach, we execute the adjusted algorithm on seven data sets and extensively compare the results to the original algorithm by Rapp [2016]. The comparison also includes taking a look at the impact the relaxed pruning has on the induced rules. Additionally, good settings for the functions presented in chapter 5 are identified. Moreover, we show the functionality of prepending by briefly evaluating it on several data sets. We also regard the learned rules and evaluate the characteristics of prepending.

- **Chapter 10:** This final chapter summarizes the contributions of the work at hand and the results obtained in previous chapters. Furthermore, suggestions for possible future enhancements of the presented algorithms are given.

2 Fundamentals of Multilabel Classification and Inductive Rule Learning

In this chapter, the basics of machine learning as well as multilabel classification and inductive rule learning are explained. This includes the formal problem statement, a short overview over problem transformations methods, as well as dependencies between labels. Furthermore, separate-and-conquer algorithms are presented.

2.1 Multilabel Classification

Classification is the task of accurately assigning classes to yet unseen *examples*. Examples are often also referred to as *instances*. In this work both terms are used synonymously. Each example is represented by a finite number of *attributes*

$$\mathbb{A} = \{a_1, \ldots, a_k\},\ k = |\mathbb{A}|.$$

Attributes, which are also referred to as *features*, can have different values, depending on whether they are *nominal* or *numeric*. If describing a car for instance, the brand may be a nominal attribute and the age may be a numeric attribute. An example is then defined as

$$X = (x_1, \ldots, x_k) \in \mathbb{X}$$

where each value $x_i, i = 1, \ldots, k$ is a valid value of attribute a_i and $\mathbb{X}$ the set of all possible examples, i.e. every possible combination of attribute values. In *multilabel classification* (MLC) each example can be associated with a subset of all classes. In this setting, these classes are typically referred to as *labels*. In contrast, in *multiclass classification* each example can only be associated with exactly one out of several classes, whereas in *binary classification* only positive and negative examples of a single class can be distinguished. The multiclass classification problem can be regarded as a multilabel classification problem with the additional constraint of the presence of labels being mutually exclusive and is therefore a more specific problem statement [Loza Mencía and Janssen, 2016]. Labels are represented by a label vector $(y_1, \ldots, y_n) \in \{0, 1\}^n$, where

$$\mathbb{L} = \{\lambda_1, \ldots, \lambda_n\},\ n = |\mathbb{L}|$$

is a finite set of classes or labels. Each label attribute y_i then represents whether or not label λ_i is *irrelevant* (0) or *relevant* (1). Furthermore, we distinguish between two types of sets. The *training set* T, $m = |T|$ is used to learn a *model* for predicting the labels of yet unseen examples. In order to enable learning, each example X is annotated with its true label vector $\hat{Y} = (\hat{y}_1, \ldots, \hat{y}_n) \in \{0, 1\}^n$. In this work, we distinguish between the true label vector $\hat{Y}$ and the current label vector Y of an example. Hence, the i-th entry in the training set looks as follows:

$$T_i = (X_i, \hat{Y}_i) \in \mathbb{X} \times \{0, 1\}^n$$

The objective is to learn a function $f : \mathbb{X} \to \{0, 1\}^n$ that maps instances X to a label vector Y. Ideally, the predicted label vector equals the true (unknown) label vector of the example. For learning a predictive model, the learner must generalize the training set. For preventing a too close fit to the training set, an *overfitting avoidance bias* [Fürnkranz, 1999] is used. The quality of f is then measured on the *test set* by applying f to each instance in the test set and comparing the results to the true label vectors. This is done by calculating the value of a *loss function*. Different loss function are presented in Section 2.3.3.

2.1.1 Problem Transformation Methods

The main approach to MLC is to transform the problem into one or more single-label or regression problems. The transformed problems can then be solved using the wide variety of conventional algorithms. Approaches such as these are called *problem transformation methods* [Tsoumakas and Katakis, 2007].

Binary Relevance is arguably the most intuitive solution [Zhang et al., 2018]. The multilabel problem is decomposed into n independent binary classification problems, where n is the number of labels. Then, for each label λ_i a binary classifier c_i for learning the relevancy of λ_i is trained. In order to classify an unseen example, predictions of all classifiers $c_1, \ldots, c_n$ are combined. If a binary classifier c_i predicts a label λ_i to be relevant for a given example, the corresponding entry y_i in the label vector is set to 1 – otherwise it is set to 0. Hence implying the prediction of a full label vector.

The prominent advantage of binary relevance is its conceptual simplicity and its linear complexity in regard to the number of labels n. In addition, binary relevance is not restricted to a particular learning technique and is therefore quite flexible. Moreover, binary relevance optimizes the macro-averaged multilabel evaluations metrics, i.e. performance is optimized not for all labels in combination but for each label independently. [Zhang et al., 2018]

As each label is learned independently, however, dependencies between labels are lost. Due to this information loss, binary relevance is prone to predicting too many or too few labels, or label combinations that would never co-occur in practice [Read et al., 2011]. As it is a consensus assumption that multilabel classification may greatly benefit from exploiting correlations between labels [Zhang et al., 2018; Loza Mencía and Janssen, 2016], many correlation-enabling extensions have been proposed [Zhang et al., 2018].

Classifier Chains are such an extension. The number of classifiers are the same as for binary relevance (one for each label). However, classifiers are now linked along a chain $c_1, \ldots, c_n$ and built in ascending order, i.e. c_1 at first, followed by c_2 to c_n. During training, the feature space of classifier c_i is extended by the predictions of all previous links. In other words, each example X of classifier i is augmented by $i-1$ additional features, namely the predictions of the previous classifiers for X, i.e. $y_1, \ldots, y_{i-1}$ for X. In conclusion, label information is passed on between classifiers. Therefore enabling the exploitation of label correlations.

While classifier chains retain the advantage of low memory and run time complexity of binary relevance, they cannot be parallelized due to the dependency on preceding classifiers. Furthermore, the question arises of how to order the chain, i.e. which labels to learn first. This problem can be tackled by using a heuristic for selecting a chain order or by using an ensemble framework with random chain orders (ECCs) [Read et al., 2011; Zhang et al., 2018].

Nonetheless, the fixed order of the way the labels are learned in the classifiers makes it impossible to learn dependencies in the contrary direction [Loza Mencía and Janssen, 2016]. In their work, Loza Mencía and Janssen [2016] state they believe that for many data sets there is no possible sequence which can capture all present dependencies.

Label Powersets are another method for dealing with multilabel classification. Each different appearing label combination in the power set of the set of (relevant) labels is regarded as a separate atomic class [Loza Mencía, 2012]. Hence, at most $2^n - 1$ possibilities exist, neglecting the empty set. Conventional multiclass algorithms or further decomposition approaches can then be used for classification. However, this leads to an exponentially large number of classes and few examples per class [Tsoumakas and Katakis, 2007]. Thus, poor efficiency. In addition, only label relations and combinations which were seen in the training data can be predicted [Loza Mencía and Janssen, 2013].

2.1.2 Label Dependencies

Label dependencies are special to multilabel classification. In binary and multiclass classification the only observable probabilistic dependence is between the input attributes and the output classes [Loza Mencía and Janssen, 2016], whereas in multilabel classification labels might co-occur or be mutually exclusive for instance.

Let us assume we have weather information about a day and want to decide whether or not to play a certain game today. Then, three possible labels are *play*, *playmaybe* and *dontplay*. If we can conclude that $dontplay = 1$ and we have additionally previously revealed a label dependency $play = 0$, $playmaybe = 0 \leftarrow dontplay = 1$, we can further assign these labels.

This type of dependency enables building a more predictive rather than a descriptive model that is less susceptible to be influenced by missing values or outliers in the attribute values [Loza Mencía and Janssen, 2016]. It is a consensus assumption that multilabel classification algorithms may greatly benefit from exploiting such correlations and dependencies [Loza Mencía and Janssen, 2016; Zhang et al., 2018]. We distinguish two types of label dependencies. The previously shown example dependency is what we call an *unconditional* or *global* dependency [Dembczyński et al., 2012]. It does not depend on the attributes of a specific instance but rather solely on label conditions. Dependencies that are indeed dependent on attributes of specific instances are called *conditional* or *local* dependencies as they are only revealed in subspaces of the input space [Dembczyński et al., 2012; Loza Mencía and Janssen, 2016]. They model label dependencies that do only occur in certain situations, depending on the instance at hand. A simple example for a local dependency is $play = 1 \leftarrow playmaybe = 1 \wedge \text{temperature} \leq 25$. This label dependency only comes into play if the attribute temperature of the given instance is below or equal 25.

2.2 Inductive Rule Learning

As this thesis is about a rule learning algorithm, a short overview over inductive rule learning is given. Rule learning is a classical machine learning field and is therefore well researched [Loza Mencía and Janssen, 2016]. A rule learning algorithm tries to solve the *concept learning* problem, i.e. to differentiate instances belonging to a certain concept from those that are not (e.g. birds from other animals) [Mitchell, 1997], using (multiple) rules. A rule r is of the form

$$head \leftarrow body$$

where the body consists of a number of attribute-value tests called *conditions* and a head that predicts a class affiliation if the body evaluates to true for a given example. In the case of binary and multiclass classification the head consists of a single class value. In multilabel classification, however, the head can consist of up to n label conditions. In this work, only conjunctive rules (in contrast to disjunctive rules for instance) are considered, i.e. the conditions in the body must all be met in order for the rule to fire. The imaginary $\wedge$ is therefore omitted and replaced by a comma. An example for a rule with two conditions in the body and two label conditions in the head is

$$\text{play} = 1, \text{playMaybe} = 1 \leftarrow \text{windy} = \text{false}, \text{temperature} \leq 25$$

Implying that if the value of attribute windy is false and the value of attribute temperature is less or equal 25 for a given instance, the two labels are considered to be relevant for this specific instance. Is this the case, i.e. the rules *fires* for an instance, the rule is said to *cover* the instance. While rules using first order logic exist [Janssen, 2012], we are only concerned with propositional rules in this work. Due to this restriction, conditions can be constructed using an attribute and comparing it to a certain value. For nominal attributes this comparison amounts to an equality or inequality check against a value of this

attribute present in the training data, whereas numerical attributes can be compared using the standard comparative operators. The value that is tested against for numerical attributes must not occur in the training data. In accordance to the work this thesis builds on, label conditions may be included in the body. [Janssen, 2012]

Following the argumentation of Janssen [2012], the big advantage of rules are their easy interpretability in comparison to more complex models. Due to their simplicity, however, they often lack behind in accuracy as they are restricted as for instance continuous target functions can only be approximated by discrete rules. For more information confer [Janssen, 2012]. As a result of their easy interpretability, rules are well suited to make dependencies between labels explicit and hence further the understanding of the data set at hand [Loza Mencía and Janssen, 2016].

2.2.1 Separate-and-Conquer Rule Learning

As it turns out, a single rule is rarely sufficient to describe a data set well [Janssen, 2012]. Therefore, rule learning algorithms induce a set of rules $\mathbb{R}$ using a *separate-and-conquer* (SeCo) or *covering* approach [Fürnkranz, 1999].

These approaches can usually be characterized by two main steps and differ in the way they learn individual rules. First, a single rule only covering positive examples is learned (the *conquer* part). Which rule is chosen is determined by maximizing some given measure for estimating the quality of a rule. Second, all examples that the newly learned rule covers are removed from the data set (the *separate* part) and the rule is added to the rule set. If there are any positive examples of the concept to learn left, the first step is repeated. Algorithm 1 depicts the basic structure that all SeCo algorithms share. [Fürnkranz, 1999; Janssen, 2012]

Algorithm 1 Basic structure of a separate-and-conquer rule learning algorithm for concept learning. [Fürnkranz, 1999; Janssen, 2012]

```
1:  procedure SeparateAndConquer(Examples)
2:      Theory = ∅
3:      while GetPositiveExamples(Examples) ≠ ∅ do
4:          BestRule = FindBestRule(Examples)                ▷ the conquer part
5:          CoveredExamples = GetCoveredExamples(Examples, BestRule)
6:          Examples = Examples \ CoveredExamples            ▷ the separate part
7:          Theory = Theory ∪ BestRule
8:      end while
9:      return Theory
10: end procedure
```

Repeating these steps until all positive examples are covered (*completeness*) and all negative examples remain uncovered (*consistency*) typically results in overfitting the data set. Overfitting is to be avoided as our goal is to find a general rule set that is capable of describing a domain rather than just a specific training set [Janssen, 2012]. Therefore, the two constraints can be relaxed so that some positive examples must not be covered and some negative examples may be covered. Methods such as stopping criteria or post-processing can be used to learn simpler and often more predictive theories. [Fürnkranz, 1999; Loza Mencía and Janssen, 2016]

The returned rule set can be either ordered (a *decision list*) or unordered. Given an example X and a decision list $\mathbb{D} = \langle r_1, r_2, \ldots \rangle$, each rule of the decision list is checked in order, i.e. starting with r_1 and continuing with r_2 and so on. The first rule that covers the given example is used to predict the class

value of X. Thereafter, all subsequent rules are skipped for this example. This is sufficient for binary and multiclass problems as each example may only be associated with a single class value. [Loza Mencía and Janssen, 2016]

It is important to note that the SeCo strategy only works for binary data sets. A simple way of being able to handle multiclass problems is considering each class separately. Then, a separate rule set is learned for each class and the predictions are combined. [Loza Mencía and Janssen, 2016]

2.2.2 Different Types of Multilabel Rules

Table 1 gives an overview of the different types of possible multilabel rules. While Loza Mencía and Janssen [2016] focus on learning *single-label head* rules, which have only a single label in the head, this work concentrates on learning *multilabel head rules*, which contain two or more label assignments in their head. In order to model co-occurrence dependencies, several single-label head rules are required. In contrast, multilabel head rules conveniently allow to model such dependencies between labels [Loza Mencía and Janssen, 2016].

Head		Body	Example Rule
single-label	positive negative	label-independent	$y_1 \leftarrow c_1, c_2, c_3$ $\overline{y_1} \leftarrow \overline{c_1}, c_2, c_3$
	positive negative	partially label-dependent	$y_3 \leftarrow c_1, \overline{y_1}, y_2$ $\overline{y_3} \leftarrow \overline{c_1}, \overline{y_1}, y_2$
	positive negative	fully label-dependent	$y_3 \leftarrow \overline{y_1}, y_2$ $\overline{y_3} \leftarrow y_1, \overline{y_2}$
multilabel	sparse dense	label-independent	$y_1, y_2 \leftarrow c_1, c_2, c_3$ $y_1, \overline{y_2}, \overline{y_3} \leftarrow c_1, c_2, c_3$
	sparse dense	partially label-dependent	$y_1, y_2 \leftarrow c_1, y_3$ $y_1, y_2, \overline{y_3} \leftarrow c_1, y_4$
	sparse dense	fully label-dependent	$y_1, \overline{y_2} \leftarrow y_3$ $y_1, y_3, \overline{y_4} \leftarrow y_2$

Table 1: Different types of single-label and multilabel rules. [Loza Mencía and Janssen, 2016, Table 1]

We further distinguish between *sparse* and *dense* multilabel head rules. A sparse multilabel head rule is a multilabel head rule that does not contain all labels in the head, i.e. the prediction for some labels are omitted and postponed for subsequent rules. This approach is often more beneficial than predicting a full label combination. Hence, a dense multilabel head rule is a multilabel head rule that contains all labels in the head, i.e. predicts the full label combination. [Loza Mencía and Janssen, 2016]

Typically, the conditions in the body are tests on attributes from the instance space (denoted as c_i). For reflecting label dependencies, however, labels need to be allowed in the body as well. A rule that contains labels in the body is called *label-dependent*. Similarly, a rule that does not contain any label conditions in the body is called *label-independent*. We furthermore distinguish between rules for which the body contains both normal and label conditions (*partially label-dependent*) as well as rules for which the body exclusively consists of label conditions (*fully label-dependent*). The different types of rules can be linked with different types of dependencies. A global dependency is best reflected by fully label-dependent rules (dependencies only exist between the labels). Local dependencies on the other hand are characterized by partially label-dependent rules as they depend on a specific instance through the normal condition. [Loza Mencía and Janssen, 2016]

Moreover, we can distinguish rules by the type of label condition they predict. Just like conditions can be constructed by selecting one of several values, labels can be set to two different types of predictions. For once, a label condition may predict the relevancy or presence ($y_i = 1$) of a label (*positive heads*). Of course, a label condition may also predict the irrelevancy or absence ($y_i = 0$) of a label (*negative heads*). In the remainder of this work, the presence and absence of a label is denoted as y_i and $\overline{y_i}$, respectively. Due to the asymmetrical nature of multilabel data sets, i.e. labels appear relatively infrequently, multilabel algorithms usually focus on predicting the presence of labels. Therefore, they are often restricted to learning only positive head rules. However, allowing negative heads may reveal additional dependencies between labels, e.g. dontplay $= 0 \leftarrow$ play $= 1$. [Loza Mencía and Janssen, 2016]

According to Loza Mencía and Janssen [2016], there are two major reasons for focusing on the induction of label-dependent rules. First, label-independent multilabel head rules can only model co-occurrence (or co-absence) dependencies. For instance, global dependencies are difficult to express using only such rules. Second, using label-independent multilabel rules may require to formulate separate rules for each distinct label combination present in the data set [Loza Mencía and Janssen, 2016]. As a result, the number of rules can become very large even for medium sized data sets. Full expressiveness is achieved by label-dependent multilabel rules [Loza Mencía and Janssen, 2016].

2.2.3 Multilabel Decision Lists

Regular decision lists have already been explained in Section 2.2.1. They are an ordered rule set in which the first rule that fires determines the class value of the given example. In order to carry over the concept of a decision list to the multilabel classification setting, some adjustments need to be made. Otherwise, the classification process would stop with the first rule that fires, resulting in only a single label being set for a single-label rule or multiple labels being set for multilabel rules (unless predicting full label combinations). Even for the latter this could be problematic as an example usually has more relevant labels than a rule has labels in the head. The solution are *multilabel decision lists*. Just like for normal decision lists, rules are ordered in a sequence $\mathbb{D} = \langle r_1, r_2, \ldots \rangle$. However, the classification process does not stop as soon as the first rule fires. The process continues with the subsequent rules unless all labels are already set for the instance or the rule that fires is explicitly marked as a *stopping rule*. In that case, the classification process is terminated.

According to Loza Mencía and Janssen [2016], this mechanism allows the flexibility of assigning several labels with a single decision list but also stop the classification if need be. Especially if only positive heads are allowed, stopping rules become important. They prevent a bias towards predicting too many labels as relevant. Note that once a label is set, it cannot be changed or be corrected later by subsequent rules. We will change this behaviour in Section 8. Loza Mencía and Janssen [2016] further observe that a multilabel decision list can be viewed as a generalization of a normal single-label decision list.

The application of a decision list $\mathbb{D}$ to a test example X is depicted in Algorithm 2. As described, each rule is checked in order whether or not it covers the test example. If so, the head is applied to the label vector of the example. In the case of a stopping rule or if all rules have been checked, the classification process is terminated and all labels that have not been set by a rule are assumed to be irrelevant.

The operation of Algorithm 2 is best illustrated with an example, which is given in Table 2. Again, the attributes of the data set describe the weather on a day and the labels describe whether or not to play a certain game, do not play or play maybe. Initially, the labels of the test instance are all missing values. The first rule in the decision list covers the example. Therefore, label *playmaybe* is set to be irrelevant for the test instance. The rule at position two of the decision list does not cover the test instance. Hence, no label is set. The rule at position three covers the test instance, resulting in the application of the head to the instance. Thus, *dontplay* is set to be relevant. The fourth rule does also cover the test instance.

Algorithm 2 Application of a multilabel decision list to a test example. [Loza Mencía and Janssen, 2016, Figure 7]

Require: Test example X, multilabel decision list $\mathbb{D}$
1: $Y = (?, \ldots, ?)$
2: **for each** rule r in decision list $\mathbb{D}$ **do** ▷ in the order of insertion
3: **if** r covers X **then**
4: apply head of r on Y if corresponding value in Y is unset
5: **if** r marked as stopping rule **or** Y is complete **then**
6: assume all remaining labels in Y are negative
7: **return** Y
8: **end if**
9: **end if**
10: assume all remaining labels in Y are negative
11: **return** Y
12: **end for**

However, the corresponding label *dontplay* is already set for this instance. Therefore, the prediction for this test instance is not updated. The rule at position five is a fully label-dependent rule. The rule models the label dependency between *play* and *dontplay*. As *dontplay* is currently set to be irrelevant, the rule does not cover the test instance and we continue with the rule at position six. The currently considered rule is a rule with no conditions and is therefore always applicable. Label *dontplay* is already set and therefore not updated, which would make no difference in this case. Furthermore, label *play* is set to be irrelevant due to the value still missing. Hence, the complete label set of the test instance was predicted. No labels need to be assumed to be irrelevant.

Position	Rule
1	playmaybe = 0 ← outlook = sunny
2	play = 1, dontplay = 0 ← outlook = overcast
3	dontplay = 1 ← humidity ≥ 82.5, humidity ≤ 95.5
4	dontplay = 0 ← temperature ≥ 66.5
5	play = 1 ← dontplay = 0
6	play = 0, dontplay = 1 ← ∅

outlook	temperature	humidity	windy	play	dontplay	playmaybe
sunny	85	85	FALSE	?	?	?

Table 2: Example decision list and example test instance with initially missing values for the three labels.

2.3 Multilabel Evaluation Metrics

Evaluating multilabel classification algorithms requires different evaluation measures than evaluating binary and multiclass classification algorithms [Tsoumakas et al., 2005]. Hence, several measures from the latter domains as well as information retrieval were adopted and adapted for measuring the quality of multilabel classification algorithms [Loza Mencía, 2012]. Depending on what kind of prediction an evaluation function assesses, we can distinguish between bipartition and ranking evaluation measures [Tsoumakas et al., 2005; Loza Mencía, 2012]. However, the latter are not considered in the work at hand. Due to there being no generally accepted procedure [Loza Mencía, 2012], selected bipartition evaluation measures as well as aggregation and averaging strategies are presented in this section.

2.3.1 Bipartition Evaluation Metrics

According to Koyejo et al. [2015], a classification *metric* is used to measure the utility of a classifier, as defined by the practitioner or end-user. As most bipartition evaluation measures are based on binary and multiclass classification measures, they can always be computed from a confusion matrix [Loza Mencía, 2012]. A *confusion matrix* is a 2×2 dimensional matrix where the columns are the predictions and the rows the actual class [Loza Mencía, 2012; Chawla, 2005]. For the multilabel setting, this amounts to whether or not they are relevant and whether a label was predicted to be relevant or irrelevant. Table 3 illustrates the structure of a confusion matrix.

C	predicted	not predicted
relevant	TP	FN
irrelevant	FP	TN

Table 3: The structure of a (multilabel) confusion matrix, where true positives are denoted as TP, false negatives as FN, false positives as FP and true negatives as TN. [Loza Mencía, 2012; Chawla, 2005]

The matrix entries are integers counting the true and false positives as well as the true and false negatives of a prediction $Y = (y_1, \ldots, y_n)$ in comparison to the given true label vector $\hat{Y} = (\hat{y_1}, \ldots, \hat{y_n})$. For a global confusion matrix, the entries are usually computed for each example and then added up. In order to easily define aggregation and averaging operators, a finer distinction is needed. In addition to a confusion matrix per example, we further construct a confusion matrix for each label within an example. The *atomic confusion matrix* of a label λ_i and an instance X_j is, in accordance to Loza Mencía [2012], denoted by C_j^i and further illustrated in Table 4. An atomic confusion matrix is of maximal possible distinction and consists of a single entry only. The entries of a confusion matrix are defined as follows:

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

Note that later in this work we are going to use an alternative definition of true positives and true negatives. If a rule covers an example and the prediction of the classifier matches the ground truth, the prediction is counted as a true positive, even if both the prediction and the ground truth of the label are irrelevant. [Chawla, 2005]

	λ_1	λ_2	λ_3	λ_4
$\mathbf{x}_1$	C_1^1	C_1^2	C_1^3	C_1^4
$\mathbf{x}_2$	C_2^1	C_2^2	C_2^3	C_2^4
$\mathbf{x}_3$	C_3^1	C_3^2	C_3^3	C_3^4

Table 4: Confusion matrices with maximal granularity, i.e. atomic confusion matrices.

Some measures additionally require the number of positive examples P or the number of negative examples N. We therefore define

$$\mathrm{P} = \mathrm{TP} + \mathrm{FN} \tag{1}$$

$$\mathrm{N} = \mathrm{TN} + \mathrm{FP} \tag{2}$$

in accordance to the notation used in Table 3. Note that both numbers can easily be computed from a confusion matrix and that the calculation of the total number of positive or negative examples requires a global confusion matrix.

2.3.2 Aggregation and Averaging

The most common approach to defining multilabel evaluation metrics is to average binary evaluation metrics [Koyejo et al., 2015]. Several possibilities for aggregating and hence obtaining a single comparable value on the test set exist. Two well-known averaging strategies are *micro-averaging* and *macro-averaging*. The main difference between the two lies in the order of aggregation and application of the evaluation function. While micro-averaging first adds up all results (aggregation), which are then transformed into a single value by the evaluation function (application), macro-averaging first applies the evaluation function on all available results (application), followed by the aggregation by computing the arithmetic mean of the computed values (aggregation). Let C_i, $i = 1, \ldots, n$ be a sequence of confusion matrices. While other addable and scalable results are eligible for aggregation operators, they are not considered in this work. We can now define the two aggregation operators

$$\sum_{i=1}^{n} C_i := C_1 \oplus \ldots \oplus C_n \tag{3}$$

$$\underset{i=1}{\overset{n}{\text{avg}}} C_i := \frac{1}{n} \sum_{i=1}^{n} C_i \tag{4}$$

where $\oplus$ denotes the cell-wise addition of confusion matrices and the averaging operator amounts to calculating the arithmetic mean over a number of values. Averaging strategies can then be distinguished by the order in which they apply the aggregation operators and the evaluation function δ. [Loza Mencía, 2012]

Let i iterate over all labels $\lambda_1, \ldots, \lambda_n$ and j over all instances $X_1, \ldots, X_m$ of the training set T and let $\circ$ denote the concatenation of operations, then the following four mathematically distinct combinations exist (cf. [Loza Mencía, 2012])[1]:

- **(Label and Example-Based) Micro-Averaging:** All atomic confusion matrices are added up to a single complete matrix. Then, the evaluation function δ is applied to the complete confusion matrix. Corresponds to averaging over both labels and examples [Koyejo et al., 2015].

$$\delta(\sum_i \sum_j C_j^i) = \delta(\sum_j \sum_i C_j^i)$$

- **Example-Based (Macro-)Averaging:** First, the atomic confusion matrices for each example (one for each label) are added up to a confusion matrix per example (hence example-based). Second, the evaluation function is applied to the confusion matrix per example. Hence, one value for each example is obtained and then finally averaged over all m examples. Corresponds to classification performance across examples [Koyejo et al., 2015]. Note that this averaging strategy values each example equally, i.e. each example is considered with the same weight [Loza Mencía, 2012].

$$\underset{j}{\text{avg}} \circ \delta(\sum_i C_j^i)$$

[1] Note that different averaging strategies can (but do not have to) have the same result for some evaluation measures [Tsoumakas et al., 2005].

- **Label-Based (Macro-)Averaging:** At first, the atomic confusion matrices for each label (one for each example) are added up to a confusion matrix per label (hence label-based). Then, the evaluation function is applied to the confusion matrix per label and averaged. Hence, one value for each label is obtained and then finally averaged over all n labels. Corresponds to the average classification performance across labels [Koyejo et al., 2015]. Note that this averaging strategy values each label equally, neglecting their frequency, i.e. a rare label is considered to be as important as a frequent one [Loza Mencía, 2012].

$$\underset{i}{\text{avg}} \circ \delta(\sum_{j} C_j^i)$$

- **(Label and Example-Based) Macro-Averaging:** First, the evaluation function is applied on each atomic confusion matrix. Second, the obtained values are averaged over both examples and labels, whereby the order of the averaging operations is irrelevant.

$$\underset{i}{\text{avg}} \circ \underset{j}{\text{avg}} \circ \delta(C_j^i) = \underset{j}{\text{avg}} \circ \underset{i}{\text{avg}} \circ \delta(C_j^i)$$

2.3.3 Selected Evaluation Functions

In this section, the multilabel evaluation metrics used in this work are described. Evaluation metrics are also called *heuristics* as they only offer an approximate solution, i.e. we select the rule that seems to currently be the best according to some measure but we do not have full knowledge for selecting the ideal rule [Janssen, 2012]. Therefore selecting a seemingly worse rule now might lead to better results further down the road. As Janssen [2012] explains, in order to learn a good and predictive rule set, each individual rule must optimize two criteria simultaneously. For once, *coverage*, the number of covered positive examples (in our case number of true positives), must be maximized. Secondly, *consistency*, the number of covered negative examples (in our case false positives), must be minimized. The following heuristics are used to estimate the quality of a rule or rule set:

- **Precision:** Computes the percentage of correctly predicted labels among the predicted labels. If no negative heads are allowed, this amounts to the percentage of relevant labels among the predicted labels. Precision is especially vulnerable to overfitting a data set as it tends to learn overly complex rules. To illustrate this, consider a rule that covers one example correctly (e.g. predicts the full label combination for an example correctly). For this rule $\delta_{prec} = 1$. In contrast, for a rule that covers 99 out of 100 covered examples correctly $\delta_{prec} = 0,99$. Hence, the former rule is chosen albeit the latter most likely being better and having better predictive qualities. Implying coverage becomes less important the more consistent a rule is. [Janssen, 2012]

$$\delta_{prec}(C) = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall:** Computes the percentage of relevant labels that are predicted. While precision optimizes consistency, recall tries to maximize coverage.

$$\delta_{rec}(C) = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{P}}$$

- **F-Measure:** The f-measure originates in information retrieval and solves the overfitting problem of precision by incorporating recall. Namely, it computes the harmonic mean between precision and recall, trading off the two values. The harmonic mean definition of the f-measure is usually referred to as the f_1-measure as (for this measure) the ideal ratio of recall against precision is one, hence both being equally important [Sasaki, 2007]. The general definition of the f-measure is

$$F_\beta = \delta_F(C) = \frac{(\beta^2+1)\cdot\delta_{prec}(C)\cdot\delta_{rec}(C)}{\beta^2\cdot\delta_{prec}(C)+\delta_{rec}(C)}$$

  where $\beta \geq 0$ is a parameter controlling the desired balance between precision and recall [Sasaki, 2007; Loza Mencía and Janssen, 2016]. For $\beta > 1$, the f-measure becomes more recall-oriented, whereas for $\beta < 1$ it becomes more precision-oriented [Sasaki, 2007]. Recall that in MLC it is often preferable to focus on correctly classifying relevant labels rather than irrelevant ones. This makes the f-measure a good choice as maximizing it results in trying to match all the relevant labels in the label matrix (depending on the balance defined by β) [Loza Mencía and Janssen, 2016]. Further note that the f-measure completely ignores true negatives.

- **Hamming Accuracy:** Before defining hamming accuracy, we define hamming loss. The hamming loss evaluation function computes the percentage of misclassified labels. Misclassified labels are either relevant labels predicted to be irrelevant (false negatives) or the other way around (false positives). Therefore, the hamming loss is defined as

$$\delta_{hammloss}(C) = \frac{\mathrm{FN}+\mathrm{FP}}{\mathrm{P}+\mathrm{N}}$$

  Hamming loss is a popular loss function in literature. However, it has a bias towards favoring algorithms with high precision and low recall. For more information on this subject confer Loza Mencía [2012]. In order to better compare hamming loss values to other heuristic values, it is often expressed in terms of hamming accuracy. Hamming accuracy is simply defined as

$$\delta_{hamm}(C) = 1-\delta_{hammloss} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{P}+\mathrm{N}}$$

- **Subset Accuracy:** Subset accuracy is a very strict evaluation measure. It is related to the 0/1 loss [Friedman, 1997] and measures the percentage of perfectly classified examples, i.e. the predicted label vector $Y_i = (y_1,\ldots,y_n)$ and the true label vector $\hat{Y}_i = (\hat{y_1},\ldots,\hat{y_n})$ must match exactly for an example X_i in order to be counted positively. Hence, subset accuracy is defined as

$$\delta_{subset}(C) = \frac{1}{m}\sum_{i}^{m}[Y_i = \hat{Y}_i]$$

  where the $[x]$ operator is used to transform a boolean value into an integer. If x = true, then $[x] = 1$. Similarly, if x = false, then $[x] = 0$. Despite expressing the ideal objective for a classification algorithm (correctly predicting the label set of each example), subset accuracy becomes increasingly useless with a higher number of labels and problems that are hard to learn [Loza Mencía, 2012]. The reason for that is the fact that the heuristic value is (almost) zero in such cases. Differences in subset accuracy then become insignificant, although one algorithm being better than the other for other measures. Note that subset accuracy is example-based and is not eligible for micro-averaging for instance. [Loza Mencía, 2012; Zhu et al., 2005; Tsoumakas et al., 2005; Loza Mencía and Janssen, 2016]

2.4 Related Work

In Section 2.1.1, we have already seen different approaches for solving the multilabel classification problem. However, these approaches aimed at transforming the multilabel classification problem into a multiclass classification problem. In this section, we first cover two algorithms that aim at learning multilabel rules. Furthermore, an alternative way of construction a decision list is presented.

2.4.1 Algorithms Learning Multilabel Rules

The first algorithm that is capable of learning multilabel rules is called multiclass, multilabel association classification (MMAC) [Thabtah et al., 2006]. The approach is based on learning association classification rules, which are a special case of association rule mining. MMAC utilizes association classification rules, which are a special case of association rule mining. (Descriptive) association rules relate properties of the data in the body to other properties in the head [Loza Mencía et al., 2018]. Thus, more than one label may appear in the head of the rule. In order to find association rules, typically an exhaustive search is employed [Loza Mencía et al., 2018]. Implying that all rules satisfying a minimum support and a minimum confidence are learned. Similarly, for learning class-association rules (CARs) frequent items have to be identified at first. Afterwards, association classification methods learn a complete set of CARs for frequent items with a minimum confidence [Thabtah et al., 2006]. MMAC can be divided into three phases: rule generation, recursive learning and classification. Following the description of Thabtah et al. [2006], the algorithm starts with scanning the training data in order to generate a complete set of CARs. In the second phase, more rules that fit the thresholds are identified by regarding the remaining unclassified instances. The last and final step merges rules with the same body into multilabel rules, where the order of the labels in the head are ranked according to their frequency. This is a major difference to the approach presented in this work. Whereas MMAC merges single-label rules with the same body to multilabel rules, we directly learn the best performing head for a given body, which may result in a multilabel rule. Additionally, Thabtah et al. [2006] employ a different strategy for classification. While we use a multilabel decision list and typically make partial predictions, MMAC uses a method that relates to regular decision lists. Implying that the first rule that fires classifies the test instance. Hence, only a single rule classifies a test instances and thus exclusively predicts the labels of said instance. Furthermore, for MMAC and most other approaches based on association rules, it remains unclear what multilabel loss functions the discovered rules are trying to minimize [Loza Mencía et al., 2018]. Although MMAC uses an evaluation measure that is somewhat suitable for measuring the quality of the learned models for multilabel classification problems, they do not utilize those measure in the learning process. In contrast, the approach presented in this work is capable of employing different kinds of popular multilabel evaluation metrics both in the learning process and to evaluate the learned models. Loza Mencía et al. [2018] identify this as a key advantage of rule-based methods, i.e. that the learned rules can be flexibly adapted to suit different evaluation metrics.

Bosc et al. [2016] propose an approach that is based on an algorithm for subgroup discovery and that can find the top-k multilabel rules. While they use a quality measure to rate rules, the measure is based on subgroup discovery and is thus not related to popular multilabel evaluation metrics [Loza Mencía et al., 2018]. Furthermore, Bosc et al. [2016] aim at learning descriptive rules with the intention of gaining information on odor structures. Contrary, the algorithm presented in this work aims at learning predictive rules, exploiting label dependencies for improving the predictive quality and making them explicit. However, both approaches aim at exploring the exponentially growing label search space, for which Bosc et al. [2016] opt for a beam search. They start with the most general local subgroup and try to specialize it. The process continues until no improvements in the quality measure can be observed anymore. This approach has similarities to the finding of the best head in this work. Especially since they use anti-monotonicity of support in order to prune the search space. Nevertheless, the approach presented in this work employs a still quite different strategy with different objectives.

2.4.2 Prepending Inferred Rules

So far we have regarded two ways of learning a decision list. Regular decision list are explained in Section 2.2.1, whereas a generalization of this concept was introduced in Section 2.2.3 in the form of a multilabel decision list. Both types of decision lists are constructed by learning a new rule on the current training set and appending the learned rule at the end of the list. The first rule that fires then makes the prediction or parts of it. An alternative strategy for constructing decision lists is *prepending* [Webb, 1993; Newlands and Webb, 2004]. The general idea is to learn a default rule that predicts the majority class initially and then continue to learn rules. Contrary to the other decision lists, prepending inserts new rules at the beginning of the list instead of at the end. Hence, the rule that was learned last is the first rule in the decision list. This approach was hypothesized to develop smaller decision lists and improve the predictive quality by taking the default rule into account during learning [Webb, 1993; Newlands and Webb, 2004].

Initial experiments by Webb [1993] could not confirm these theories. Instead, the predictive accuracy of prepending suffered in comparison to appending the learned rules. The found reason for this observation is the way rules that cover only a few positive cases are handled [Webb, 1993; Newlands and Webb, 2004]. Due to such rules covering only a few positive examples, they are typically learned last and hence placed at the front of the decision list. In contrast, appending places such rules at the end of the decision list. However, these rules have low support and thus have a higher expected error rate. As a result of their placement at the front of the decision list by prepending, their likelihood of firing is maximized. Thus, they contribute significantly to the global error of the decision list. Summing up, the problem is that rules with too low of a support are placed before rules with the best support.

Both by Webb [1993] and Newlands and Webb [2004] adjustments for solving this problem have been proposed. For instance, Webb [1993] requires rules to have a certain minimum support in order to get learned. Applying these adjustments has helped to improve the performance of prepending and has resulted in more compact decision lists. However, they still lack behind in predictive performance.

In this work, we propose a similar approach to prepending in Section 8. Hence, we refer to our approach as prepending as well. For our approach, we do also take advantage of prepending a default rule initially. The difference is that afterwards we still append newly learned rules at the end of the list. Thus, we consider the default rule in the learning process. Furthermore, as we learn rules in a different manner and append them to the end of the decision list, we do not run into similar problems regarding rules that only cover a few positive examples. Additionally, we set a condition that limits the quality of the rule. While not exactly the same as requiring a minimum support, it can be thought of as a related concept. For instance, Webb [1993] cuts of rules with a too low support and requires rules to have a minimum number of covered positives.

3 Anti-Monotonicity of Multilabel Evaluation Metrics

This work focuses on finding the best head for a given rule body. For doing so, we systematically search through the label space $\mathbb{L}$ (cf. Figure 1). However, due to its exponential growth and a possibly large number of labels this becomes infeasible [Loza Mencía, 2012]. Especially when considering that this action must be performed for every possible candidate, i.e. every possible body of length one, and following refinements. In the master thesis by Rapp [2016] and subsequent publication thereof Rapp et al. [2018], the problem is tackled by transferring anti-monotonicity to the MLC setting. Anti-montonicity is a well known property in association rule mining and is used to limit the number of necessary evaluations while still guaranteeing the best solution [Ng et al., 1998]. Using anti-monotonicity, the label space can be pruned by leaving out unpromising label combinations without missing the head with the best performance. In this section, the anti-monotonicity and decomposability characteristics of multilabel evaluation measures according to Rapp et al. [2018] are introduced. Formally, we need to find the best performing multilabel head Y as defined by

$$h_{max} = \max_{Y} \delta(r) = \max_{Y} \delta(Y \leftarrow B)$$

where h is an evaluation function (also called heuristics) and a body B [Rapp et al., 2018]. This entire section is based on Rapp et al. [2018] and Rapp [2016]. Hence, we omit explicit references.

3.1 Anti-Monotonicity

Anti-monotonicity is the first property which can be exploited for pruning the search through the label space. Intuitively speaking, once the performance decreases by adding an additional label to the head, the maximum performance cannot be be reached anymore by further label additions to the same head. Hence, all supersets of the label combination in the head must not be evaluated anymore, i.e. can be pruned without missing out on the best performing head. In order to formally define the anti-monotonicity characteristic, let $Y_p \leftarrow B$ and $Y_s \leftarrow B$ denote two multilabel rules with the same body B (as the body is fixed for one search through the label space) and heads Y_p and Y_s. Furthermore, let $Y_p \subset Y_s$, i.e. Y_s consists of all labels present in Y_p as well as at least one additional label. Then, a multilabel evaluation function δ is *anti-monotonic* if

$$Y_p \subset Y_s \wedge \delta(Y_s \leftarrow B) < \delta(Y_p \leftarrow B) \Longrightarrow \delta(Y_a \leftarrow B) < h_{max}, \forall Y_a : Y_s \subset Y_a$$

and δ must not be *monotonous*, considering the averaging and evaluation strategy. The reason being that when using a monotonous function, adding an additional label to a multilabel head never causes the measured performance to decrease on a data set T. Formally, monotonicity is defined as

$$\delta(Y_a \leftarrow B, T) \geq \delta(Y_p \leftarrow B, T), \forall Y_a : Y_s \subset Y$$

Omitting the condition of not being montonous can lead to rules with full label combinations in the head. Hence, being equivalent to an exhaustive search of the label space in terms of computational complexity.

Figure 1 illustrates how the label space can be pruned by exploiting anti-monotonicity. Naturally, all single-label heads need to be evaluated. If the heuristic value increases, we cannot prune as the heuristic value might still increase by adding more labels. Confer the edge of $\{y_1\}$ to $\{y_1, y_2\}$) for instance. If the heuristic value declines (indicated by a red arrow), however, thanks to anti-monotonicity we can be sure that the heuristic values cannot increase anymore by adding more label attributes to the head. An example for this is the label combination $\{y_2, y_3\}$. By adding y_3 to $\{y_2\}$ the heuristic value declines. Therefore, anti-montonicity can be applied and all supersets of the head can be pruned.

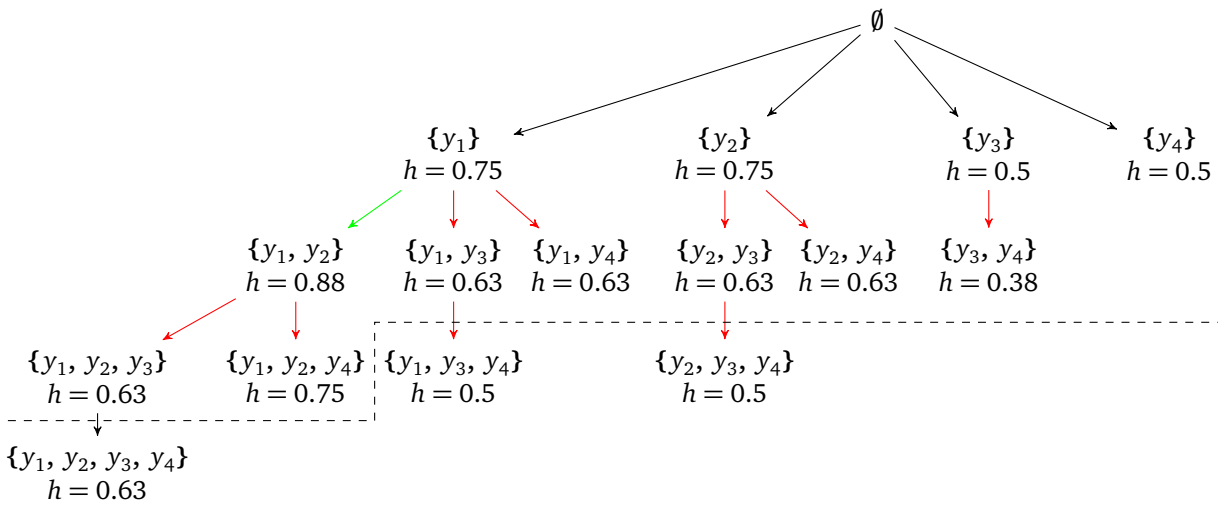

Figure 1: Searching through the label space and pruning according to anti-montonicity. All heads below the dashed line do not need to be evaluated. Red arrows indicate a drop in the heuristic value, whereas green arrows indicate an improvement. [Rapp, 2016, Presentation]

3.2 Decomposability

Decomposability is the second and stronger property. Using decomposability, no deep searches through the label space must be performed. Instead, it comes at a linear cost as multilabel heads can easily be constructed by first evaluating each label independently and then iteratively combining the performance measures. Intuitively speaking, decomposability states that if combining several labels to a new multilabel head, the performance of the new multilabel head drops (in comparison to the performance of the best single label) if one single label performance is worse than the others, i.e. not all performances are equal. Think of it as one or more labels dragging the performance of the combined multilabel head down. Therefore, after evaluating all single label heads, we only need to combine the heads with the maximum performance out of all single label heads (ignoring the labels dragging us down) to find the best performing head. Keep this property in mind as we exploit it in Section 6. Using the same notation as for the definition of anti-monotonicity, a multilabel evaluation function δ is *decomposable* if

- the multilabel head rule $Y \leftarrow B$ contains a label attribute $y \in Y$ for which the corresponding single label single-label head rule $y_i \leftarrow B$ does not reach h_{max}, the multilabel head rule cannot reach that performance either (and vice versa):

$$\exists i \left(y_i \in Y \wedge \delta(y_i \leftarrow B) < h_{max} \right) \Longleftrightarrow \delta(Y \leftarrow B) < h_{max}$$

- and if all single label head rules $y_i \leftarrow B$ which correspond to the label attributes of the multilabel head Y reach h_{max}, the multilabel head rule $Y \leftarrow B$ reaches that performance as well (and vice versa):

$$\delta(\hat{y}_i \leftarrow B) = h_{max}, \forall y_i \left(y_i \in Y \right) \Longleftrightarrow \delta(Y \leftarrow B) = h_{max}$$

as well as δ must not be monotonous, considering the averaging and evaluation strategy. Decomposability is a stronger criterion than anti-monotonicity. Hence, if an evaluation function is decomposable, anti-montonicity is implied as well (check Rapp [2016] for the proof). Despite, pruning according to decomposability should always be preferred as it allows to prune more heavily, leading to a more efficient algorithm. Anti-monotonicity is to be used if decomposability is not met, as we will see is the case for subset accuracy for instance.

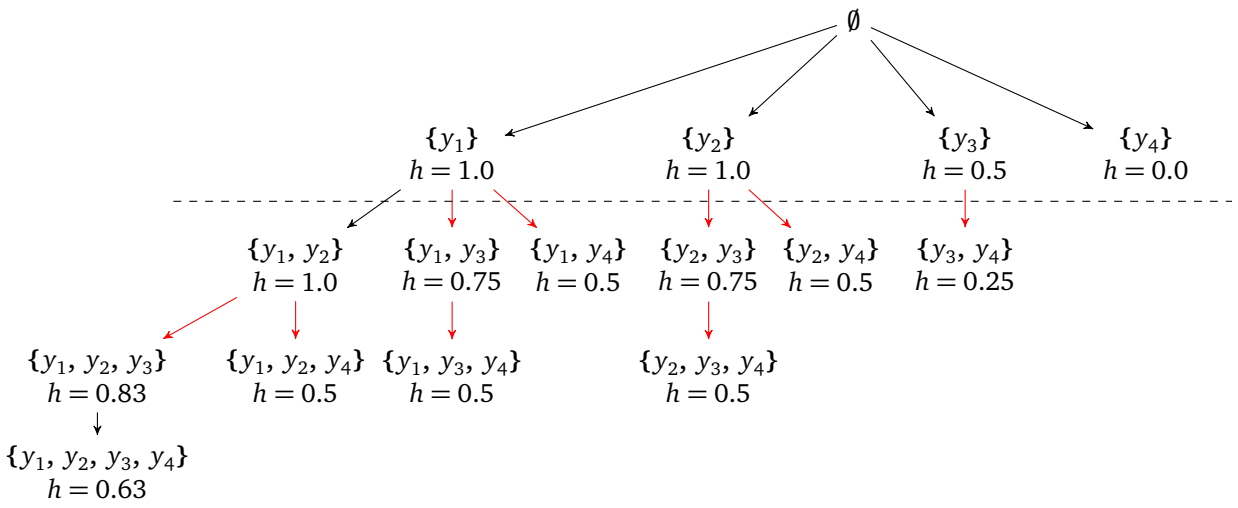

Figure 2: Searching through the label space and pruning label combinations based on decomposability. All heads below the dashed line do not need to be evaluated (on the training set). Red arrows indicate a drop in the heuristic value, whereas green arrows indicate an improvement. [Rapp, 2016, Presentation]

Figure 2 illustrates how the label search space can be pruned while exploiting decomposability. Naturally, all single-label heads need to be evaluated (on the training set). According to decomposability, a head consisting of several labels cannot reach the best possible performance if one of the corresponding single-label heads does not reach the best performance for the given body. Therefore, we can construct the best head by first evaluating all single-label head rules and subsequently combining all single-label heads that reach the best performance. For the example illustrated in Figure 2, a heuristic value of 1.0 is the best performance of any single-label head. Hence, we combine all single-label heads that reach that performance ($\{y_1\}$ and $\{y_2\}$). Due to decomposability we can be sure that the combined head reaches the same heuristic value. Note that as a result, we do not have to evaluate the combination on the training set and all other label combinations do not have to be regarded and can be pruned.

3.3 Rule-Dependent Evaluation

For evaluating rules, we distinguish between two evaluation strategies as defined by Rapp [2016]. In Section 2.3 we introduced aggregation and averaging strategies. We saw that we can either average over labels, examples or both at the same time. For evaluating a single rule, a natural approach is to consider all examples the rule covers and not covers in the calculation of the evaluation metric. The two mentioned evaluation strategies differ in which labels they consider for the calculation of the evaluation metric. They are defined as follows

- **Rule-Independent**: This evaluation strategy is not dependent on the labels an individual rule predicts. Rather, all labels $\lambda, \ldots, \lambda_n \in \mathbb{L}$ are considered in the calculation of the performance of a rule. As most rules do not predict a full label combination, labels not present in the head are assumed to be irrelevant, i.e. the rule is evaluated as if containing a label attribute $y_i = 0$ for a label λ_i that is not included in the head H of a rule.

- **Rule-Dependent**: This evaluation strategy is dependent on the labels an individual rule predicts. Only labels present in the head are taken into account in the calculation of the performance of the rule, i.e. all labels λ_i for which a corresponding label attribute $y_i \in$ H.

The reason for assuming not present labels to be irrelevant for the rule-independent evaluation strategy lies in the way the label vector is constructed for a test example. After applying the multilabel decision

list on the test example, all remaining unset labels in the label vector are predicted to be irrelevant. Therefore, it might be preferable to consider this in the evaluation of a rule. However, the labels that are assumed to be irrelevant in the evaluation of a rule, might have been previously set by preceding rules in the multilabel decision list. This situation arises due to rules mostly predicting only a few labels instead of complete label vectors. In fact, Rapp [2016] states that in most cases performance suffers if using a rule-independent evaluation strategy. Additionally, run time increases as well due to two reasons. The rule-independent metrics used in this work and by Rapp [2016] are not decomposable (cf. Section 3.4). Therefore, pruning can only be performed using anti-monotonicity, which is a weaker criterion. Furthermore, the rule-independent evaluation strategy introduces a bias towards learning full label combinations. Hence, only covering few examples per rule, which results in more rules needed to describe the data set. In conclusion, in this work we focus on learning rules using a rule-dependent evaluation strategy. If not stated otherwise, a rule-dependent evaluation strategy is used.

3.4 Anti-Monotonous and Decomposable Evaluation Metrics

Table 5 shows whether or not the in Section 2.3.3 presented evaluation functions are anti-monotonous or decomposable for the different averaging strategies presented in 2.3.2. Recall is omitted as it is not used in the evaluation. For detailed proofs of the characteristics confer Rapp [2016].

Evaluation Function	Evaluation Strategy	Averaging Strategy	A-M	D
Precision	Rule-Dependent	Micro-Averaging	√	√
		Label-Based Averaging	√	√
		Example-Based Averaging	√	√
		Macro-Averaging	√	√
	Rule-Independent	Micro-Averaging	√	
		Label-Based Averaging	√	
		Example-Based Averaging	√	
		Macro-Averaging	√	
F-Measure	Rule-Dependent	Micro-Averaging	√	√
		Label-Based Averaging	√	√
		Example-Based Averaging	√	√
		Macro-Averaging	√	√
	Rule-Independent	Micro-Averaging	√	
		Label-Based Averaging	√	
		Example-Based Averaging	√	
		Macro-Averaging	√	
Hamming Accuracy	Rule-Dependent	Micro-Averaging	√	√
		Label-Based Averaging	√	√
		Example-Based Averaging	√	√
		Macro-Averaging	√	√
	Rule-Independent	Micro-Averaging	√	
		Label-Based Averaging	√	
		Example-Based Averaging	√	
		Macro-Averaging	√	
Subset Accuracy	Rule-Dependent	Example-Based Averaging	√	
	Rule-Independent			

Table 5: Anti-monotonicity (A-M) and decomposability (D) of evaluation measures relevant to this work, regarding different averaging and evaluation strategies. [Rapp, 2016, Table 11]

4 Discovery of Multilabel Rules

In this chapter, the algorithm this work is based on is presented. The algorithm itself is a separate-and-conquer algorithm that was proposed by Loza Mencía and Janssen [2016] and further adapted by Rapp [2016]. In the following two sections, the conceptual changes made in each work are shortly explained. Then, we break down the algorithm into its components and take a more detailed look at how it works.

4.1 A SeCo Algorithm For Learning Single-Label Head Rules

The algorithm suggested by Loza Mencía and Janssen [2016] aims at learning single-label head rules. It follows the same basic structure of separate-and-conquer algorithms as shown in Algorithm 1. However, the algorithm had to be adapted in order to be capable of handling multilabel classification problems. According to Loza Mencía and Janssen [2016], their main motivation was to keep adjustments to the classical SeCo algorithm at a minimum. We already covered the first change. Instead of using a regular decision list, the algorithm uses a multilabel decision list in combination with stopping rules as outlined in Section 2.2.3. This becomes necessary as the algorithm learns single-label head rules, which in turn implies the need for multiple rules covering a single instance, which typically have several relevant labels. Additionally, training examples can be re-included in the training process. This stands in contrast to the approach used in a classical separate-and-conquer algorithm, where the examples are removed once they are covered by a rule. We further differentiate between re-including partially covered examples, those for which some but not all labels are set, and re-including fully covered examples, those for which all labels are set for the instance. This leads to three possibilities of re-adding covered examples, namely re-adding only partially covered-examples, re-adding partially and fully covered examples as well as re-adding none. Removing all fully covered examples might introduce inconsistencies [Loza Mencía and Janssen, 2016]. Following the argumentation of Loza Mencía and Janssen [2016], the reason being that when removing fully covered examples, subsequent rules might be inconsistent with the fully covered examples. Moreover, fully covered examples may serve as an anchor point for subsequent rules. Hence, improving the rule induction process. Another big difference lies in the iterative nature of the algorithm. In order to be able to learn label-dependent rules, label conditions are injected into the training process for labels for which a rule has already been learned. Therefore enabling the modeling of label dependencies. This is a key component of the algorithm as it aims at making dependencies between labels explicit by using rules. Additionally, by allowing the learning of negative heads, the definition of whether a label or instance is covered changes.

4.2 A SeCo Algorithm For Learning Multilabel Head Rules

Rapp [2016] adapted the algorithm by Loza Mencía and Janssen [2016] in order to be able to learn multilabel head rules. Naturally, they share a common structure. Both induce a multilabel decision list by successively learning rules and both eventually remove covered training examples. Nevertheless, both algorithms significantly differ in the way they learn individual rules. While the algorithm by Loza Mencía and Janssen [2016] is based on finding the best body for a given label, the algorithm by Rapp [2016] is based on finding the best performing head for a given body and trying all possible bodies. For this search, exploiting anti-monotonicity and decomposability of multilabel evaluation metrics is crucial. In this and the following sections, the individual components of said algorithm are presented. We do only cover the algorithm for learning multilabel head rules. However, some subroutines are identical for both variants. For clarity, we mostly omit explicit references to Rapp [2016] and Loza Mencía and Janssen [2016] as this entire chapter is based upon those works.

Algorithm 3 illustrates the training process of the rule learner. The algorithm takes a training data set, containing pairs of instances X and corresponding true label vectors $\hat{Y}$ as well as a set of parameters as an input. The algorithm keeps track of two label vectors per instance. The first represents the true label vector. The second represents the currently set labels for the instance, initialized with missing

values as no labels are predicted in the beginning. Then, the learning algorithm runs as long as no more than θ uncovered examples are left in the training set (cf. Algorithm 3, line 4). In each iteration, we first learn the best possible rule r using the subroutine FindBestGlobalRule. Depending on whether $G = \{1\}$ or $G = \{0, 1\}$, we consider only positive or positive and negative heads in the rule learning process. After the rule is learned, we append it to the multilabel decision list $\mathbb{D}$. Afterwards, we split the training set depending on the covering status using the subroutine GetCoveredSets (cf. Algorithm 3, line 7). The result is a triple of sets, containing the remaining uncovered training examples T, the partially covered examples T_{part} and the fully covered examples T_{full}, respectively. Depending on the parameters set by the user, the subroutine GetReAddSets returns the already covered examples to be re-added to the training set. If the percentage of fully covered examples among all covered examples exceeds parameter τ (usually close to 1), the prediction is assumed to be complete and the current rule is marked as a stopping rule. In this case, the subroutine GetReAddSet returns an empty set (cf. Algorithm 3, lines 8-10). If the percentage of fully covered examples does not exceed the specified threshold, the partially covered examples are re-added to the training set. Additionally, a parameter can be set as to whether or not to re-include the fully covered examples in order to benefit from the enriched information as mentioned previously. As soon as enough examples are covered, the loop terminates and the multilabel decision list is returned.

Algorithm 3 Training algorithm for both variants of the multilabel iterative separate-and-conquer algorithm. [Loza Mencía and Janssen, 2016, Figure 3]

Require: New training examples pairs $T = \{(X_1, \hat{Y}_1), \ldots, (X_m, \hat{Y}_m)\}$,
parameters θ,τ, heuristic h, targets G (either $G = \{1\}$ or $G = \{0, 1\}$),
whether using stopping rules, whether re-inserting fully covered examples

1: **procedure** MultilabelSeCo
2: $R = \emptyset$
3: $T = \{(X_1, \hat{Y}_1), \ldots, (X_m, \hat{Y}_m)\}$ with $Y_i = (?, \ldots, ?)$, $i = 1, \ldots, m$
4: **while** $|T|/m \geq \theta$ **do** ▷ until, e.g. 95% of examples covered
5: $r = \text{FindBestGlobalRule}(G, T)$ ▷ get best possible rule regardless the head
6: $R = R \cup r$
7: $(T, T_{part}, T_{full}) = \text{GetCoveredSets}(r, T)$ ▷ separate T according covering by r
8: $T_{add} = \text{GetReAddSet}(T_{part}, T_{full})$ ▷ depending on user parameters
9: **if** $T_{add} = \emptyset$ **then**
10: mark r as stopping rule ▷ only uncovered examples in T of next round
11: **else**
12: $T = T \cup T_{add}$ ▷ add also some covered examples, do not remove them
13: **end if**
14: **end while**
15: **return** multilabel decision list R
16: **end procedure**

4.3 Finding The Best Rule

The algorithm for finding the best global rule for the currently remaining training set is shown in Algorithm 4. The best rule is searched for in a top-down fashion, i.e. starting with the most generic rule and successively specializing it. The most generic rule is a rule with no conditions in its body. It covers all training examples as the body evaluates to true for each example.[2] The rule is then iteratively refined by adding conditions to its body. Implying, the rule is specialized as the refined rule covers only a subset

[2] The reason being that an empty conjunction evaluates to true, which can be thought of as no conditions being evaluated to true being required.

of examples the unrefined rule covered. Changing the body requires the head to be updated as well. This is necessary because for the new rule, predicting another head might yield a better performance as it now covers a different set of examples. The rule induction process stops if the performance, which is measured by a multilabel evaluation metric, cannot be improved anymore by adding more conditions (cf. Algorithm 4, line 4). Eventually, the best rule r_{best} is returned.

Algorithm 4 Algorithm FINDBESTGLOBALRULE for inducing a new multilabel head rule, based on the current training set. [Rapp, 2016, Algorithm 8]

Require: Original training data set T, current training data set $T_{current}$,
targets G (either $G = \{1\}$ or $G = \{0, 1\}$), evaluation function δ,
whether to use rule-dependent or rule-independent evaluation strategy, the averaging strategy to use

1: **procedure** FINDBESTGLOBALRULE
2: $r_{best} = \emptyset$
3: $r_{best}.h = -\infty$ ▷ initialize performance of new rule
4: $improved =$ **true**
5: **while** *improved* **do** ▷ until no improvements possible
6: $r =$ REFINERULE$(T, T_{current}, r_{best}, G, \delta)$ ▷ refine rule by adding condition and updating head
7: **if** $r.h > r_{best}.h$ **then**
8: $r_{best} = r$ ▷ replace by better rule
9: **else**
10: $improved =$ **false**
11: **end if**
12: **end while**
13: **return** best rule r_{best}
14: **end procedure**

4.4 Refinement of Rule Conditions

The subroutine REFINERULE, as depicted in Algorithm 5 and called by the subroutine FINDBESTGLOBALRULE (cf. Algorithm 4), is responsible for refining the rule in the best possible way as well as finding the best performing corresponding multilabel head. In order to find the best possible refinement, the algorithm appends each possible condition, be it an attribute or a label condition, to the body (cf. Algorithm 5, lines 2 and 5). Moreover, for each possible refinement, the best head must be discovered (cf. Algorithm 5, line 6). The best refinement is chosen in regard to a performance evaluation on the remaining training instances. The conditions considered in the refinement of the rule are extracted from the attributes and the labels of the given training set. The regular attribute-value test conditions result from calling the subroutine GETATTRIBUTECONDITIONS. They check whether or not the attributes of a given example have certain values. The label conditions result from calling the subroutine GETLABELCONDITIONS. Contrary to attribute conditions, label conditions check whether or not an example is associated with certain labels. Hence, enabling the induction of label-dependent rules and therefore exploiting dependencies between labels.

4.4.1 Attribute Conditions

The subroutine GETATTRIBUTECONDITIONS (cf. Algorithm 6) retrieves all possible attribute conditions from the training set T. These conditions can then be used to specialize a body in the refinement process of a rule (cf. Algorithm 5). Naturally, all available conditions are considered. The value that the attribute is tested against depends on whether the attribute is nominal or numeric. For a nominal attribute, a condition is created for each value the attribute can obtain (e.g. sunny, rainy or overcast for an attribute weather outlook, cf. Algorithm 6, lines 4-6). In case of a numeric attribute, the creation of conditions

Algorithm 5 Algorithm RefineRule for refining the body of a multilabel head rule by adding additional conditions. Each refinement requires to update the rule's head as well. [Rapp, 2016, Algorithm 9]

Require: Original training data set T, current training data set $T_{current}$,
rule r, targets G, evaluation function δ, the averaging strategy to use,
whether to use rule-dependent or rule-independent evaluation strategy

1: **procedure** RefineRule
2: $r_{best} = r$
3: **for each** possible condition $c \in \text{GetAttributeConditions}(T) \cup \text{GetLabelConditions}(T_{current})$ **do**
4: **if** $c \notin r.body$ **then**
5: $r_{refined} = r$
6: $r_{refined}.body = r_{refined}.body \cup c$ ▷ add condition to rule's body
7: $r_{refined} = \text{FindBestHead}(T, T_{current}, r_{refined}, G, \delta)$
8: **if** $r_{refined}.h > r_{best}.h$ **then**
9: $r_{best} = r_{refined}$ ▷ replace by refined rule
10: **end if**
11: **end if**
12: **end for**
13: **end procedure**

is dependent upon the values present in the training set T. At first, the training examples need to be sorted in ascending order in regard to the attribute value (cf. Algorithm 6, line 9). Then, a split point is calculated for each pair of neighboring examples (cf. Algorithm 6, line 10). At each split point, the sorted attribute values are separated using the $\leq$ and the $\geq$ operator, respectively. Hence, creating two conditions for each split point. Ideally, split points discriminate between positive and negative examples for the prediction of a rule. For instance, let a rule predict whether or not to eat ice cream on a certain day. Let us furthermore consider an attribute temperature, which models the temperature on a given day. Then, an ideal split point, say 25, splits the examples in a way that, for instance, all examples with a temperature value ≥ 25 are days on which ice cream was eaten (positive examples) and all examples with a temperature value ≤ 25 are counter-examples, i.e. days on which no ice cream was eaten (negative examples).[3] For that reason, it is not necessary to create conditions for split points between two neighboring examples, for which the associated label vectors are identical (cf. Algorithm 6, line 11). The reason being that the condition would inadequately discriminate between examples for which the same prediction is correct.

4.4.2 Label Conditions

Including label conditions in the rule learning process allows the induction of label dependent rules. Algorithm 7 determines which label conditions to include. Once a rule is learned, its head is applied to the examples it covers, i.e. the initially empty label vector is set piece by piece. This process is discussed in more detail in the next section. Therefore, the already predicted labels, those that are available for creating conditions, can be extracted from the current training set $T_{current}$. For each already predicted label two conditions are created, testing whether or not the label is relevant or irrelevant for an example (cf. Algorithm 7, line 5). Due to the way the multilabel decision list is applied, label conditions for which the respective label has not been predicted yet, cannot be used to enhance the rule learning process. As already discussed, the decision list is applied successively to an example. Initially, all labels of said example are unset. Implying they are neither relevant nor irrelevant. Let us then assume the multilabel decision list contains a rule that is dependent on a label which has not been predicted yet. Hence, there is no chance of the label being set. Implying the condition evaluates to false as a test for relevancy

[3] Note that often such an ideal split is not possible. Rather, we try to find the split with the best discrimination.

Algorithm 6 Algorithm GETATTRIBUTECONDITIONS for retrieving all possible conditions, which are based on the training data set's attributes. [Rapp, 2016, Algorithm 10]

Require: Training data set T
1: **procedure** GETATTRIBUTECONDITIONS
2: $C = \emptyset$ ▷ start with empty set of conditions
3: **for each** attribute a_k **do**
4: **if** A_k is nominal **then**
5: **for each** value v of a_k **do** ▷ add condition for each nominal value
6: $C = C \cup (a_k = v)$
7: **end for**
8: **else** ▷ attribute is numeric
9: sort examples $(X_j, \hat{Y}_j) \in T$ by values $v_k \in X_j$ in increasing order
10: **for each** example $(X_j, \hat{Y}_j) \in T$ with $j > 1 \wedge j < |T|$ **do**
11: **if** $\hat{Y}_{j-1} \neq \hat{Y}_j$ **then** ▷ only consider neighbours with different label vectors
12: $v = v_{k-1} + (v_k - v_{k-1})/2$ with $v_{k-1}, v_k \in X_j$ ▷ calculate split point
13: $C = C \cup (a_k \geq v) \cup (a_k \leq v)$ ▷ add two conditions for each split point
14: **end if**
15: **end for**
16: **end if**
17: **end for**
18: **return** attribute conditions C
19: **end procedure**

or irrelevancy is impossible on an unset label. Consequently, the rule can never cover the example, rendering the rule useless. In contrast, once a preceding rule predicts a label, there is a chance that the label is set, making it useful.

Algorithm 7 Algorithm GETLABELCONDITIONS for retrieving all possible conditions, which are based on already predicted labels. [Rapp, 2016, Algorithm 11]

Require: Current training set $T_{current}$
1: **procedure** GETLABELCONDITIONS
2: $C = \emptyset$ ▷ start with empty set of conditions
3: **for each** example $(X_j, \hat{Y}_j) \in T_{current}$ **do**
4: **for each** label attribute $\hat{y}_i \in \hat{Y}_j$ with $\hat{y}_i \neq ?$ **do**
5: $C = C \cup (y_i = 0) \cup (y_i = 1)$ ▷ add two conditions for each already learned prediction
6: **end for**
7: **end for**
8: **return** label conditions C
9: **end procedure**

4.5 Re-Adding Covered Examples

After learning and adding the best rule to the multilabel decision list (cf. Algorithm 3, lines 5 and 6), the learned rule is applied to the training examples it covers in the subroutine GETCOVEREDSETS (cf. Algorithm 8), i.e. the label vectors are updated in regard to the new rule. Hence, the coverage status of the training examples changes. To do so, each uncovered example (examples in the current training set $T_{current}$) is checked whether it is covered by the new rule. If so, the example is removed from the training set containing the uncovered examples $T_{current}$, because it is now covered. Then the head is applied to

the label vector of the covered example. Afterwards, depending on whether the newly covered example is partially or fully covered, it is added to the set of partially covered training examples T_{part} or the set of fully covered training examples T_{full}. For deciding the coverage status of an example, we must consider whether negative heads can be learned, i.e. whether $G = \{1\}$ or $G = \{0, 1\}$. In the case of allowing negative heads, the irrelevant labels of the example must be set as well in addition to the relevant ones, in order to be considered fully covered. The result of the subroutine is a triple of sets, containing the remaining uncovered examples, partially covered examples and fully covered examples (cf. Algorithm 3, line 7).

Algorithm 8 Algorithm GETCOVEREDSETS for computing the covering status of examples for a given rule. [Loza Mencía and Janssen, 2016, Figure 5]

Require: Original training data set T, current Training data set $T_{current}$, rule r

1: **procedure** GETCOVEREDSETS
2: $T_{part} = \emptyset, T_{full} = \emptyset$
3: **for each** example $(X, \hat{Y}) \in T_{current}$ **do** ▷ compute covering status for each example
4: **if** r covers X **then**
5: $T_{current} = T_{current} \setminus (X, \hat{Y})$ ▷ remove since it may not be re-added
6: apply head of r on Y ▷ replace corresponding value in Y if it was unset
7: **if** Y is fully set **then** ▷ depending on B, consider also unset zeros
8: $T_{full} = T_{full} \cup (X, \hat{Y})$
9: **else**
10: $T_{part} = T_{part} \cup (X, \hat{Y})$
11: **end if**
12: **end if**
13: **end for**
14: **return** uncovered (T), partially covered (T_{part}) and fully covered (T_{full}) training examples
15: **end procedure**

One of the main differences of the algorithm in comparison to the classical separate-and-conquer algorithm is the re-inclusion of training examples. The subroutine GETREADDSET (cf. Algorithm 9) determines the set of covered examples that is re-added to the training set. Depending on coverage status and parameters, none, only partially covered examples or fully covered examples are re-included. The algorithm requires two sets of examples as an input, the set of partially covered examples T_{part} and the set of fully covered examples T_{full}. Both have been determined by the previously executed subroutine GETCOVEREDSETS (cf. Algorithm 8). If no stopping rules are to be used, only the partially covered examples T_{part} are added to the set of examples to be re-added (cf. Algorithm 9, line 13). Otherwise, the full coverage rate is calculated. If the rate of fully covered examples among all covered examples is greater than parameter τ (usually close to 1), the prediction is assumed to be complete for the covered examples and no covered examples are re-included or, in other words, they are omitted from future iterations, i.e. $T_{add} = \emptyset$ (cf. Algorithm 9, lines 4 and 5). In the other case, i.e. the fully coverage rate is below the given parameter, the partially covered examples are re-added. Furthermore, depending on the preferences of the user, fully covered examples are re-inserted into the training set as well. Finally, the algorithm returns the set of examples to be re-added T_{add}.

4.6 Finding The Best Head For A Rule

The subroutine REFINERULE uses the subroutine FINDBESTHEAD (cf. Algorithm 10) to determine the best performing multilabel head for a given rule body. Naively, this requires an exhaustive search through the label space, i.e. evaluating each possible combination of labels in the head. As discussed in Section 3, however, the anti-monotonicity and decomposability properties of multilabel evaluation metrics can

Algorithm 9 Algorithm GETREADDSET for deciding whether covered examples are re-added to the training set. [Loza Mencía and Janssen, 2016, Figure 6]

Require: Partially and fully covered examples T_{part}, T_{full}, parameter τ,
whether using stopping rules, whether re-inserting fully covered examples

procedure GETREADDSET
  $T_{add} = \emptyset$
  if use stopping rules **then**
    if full coverage rate $|T_{full}|/(|T_{full}| + |T_{part}|) \geq \tau$ **then** ▷ e.g. 90%
      $T_{add} = \emptyset$ ▷ do not re-add any example although T_{part}, T_{full} non empty
    else ▷ too many partially covered examples
      $T_{add} = T_{part}$ ▷ re-add partially covered examples
      if re-insert fully covered examples **then**
        $T_{add} = T_{add} \cup T_{full}$ ▷ also re-add fully covered examples
      end if
    end if
  else
    $T_{add} = T_{part}$ ▷ no stopping rules: re-add partially covered examples
  end if
  return partially or fully covered examples T_{add} to be re-added to the training set
end procedure

be exploited in order to avoid such an exponential computational complexity, which is aggravated by a huge number of labels. Consequently, the following algorithms prune searches through the label space according to those properties. Algorithm 10 depicts how we decide which property to use. If the evaluation function δ in combination with the utilized evaluation and averaging strategy is decomposable, we opt for the subroutine DECOMPOSITE, which is capable of exploiting decomposability. Otherwise, if the used evaluation function only satisfies the anti-monotonicity criterion, the subroutine PRUNEDSEARCH is chosen, which is capable of exploiting anti-monotonous evaluation functions. Preferring the decomposability metric is critical as it allows for heavier pruning of the label space. Both algorithms further prefer heads with more labels in the head as the objective of the algorithm is to learn more multilabel head rules instead of single-label head rules. Note that metrics fulfilling neither property are not supposed to be employed with the algorithm. The properties of selected evaluation function are depicted in Table 5.

Algorithm 10 Algorithm FINDBESTHEAD for finding the best multilabel head for a given rule's body. [Rapp, 2016, Algorithm 12]

Require: Original training set T, current training set $T_{current}$,
current rule r, targets G, evaluation function δ,
whether to use rule-dependent or rule-independent evaluation strategy, the averaging strategy to use

procedure FINDBESTHEAD
  $r.head = \emptyset$
  if δ is decomposable using the given evaluation and averaging strategy **then**
    return DECOMPOSITE$(T, T_{current}, r, G, \delta)$
  else
    return PRUNEDSEARCH$(T, T_{current}, r, G, \delta, \emptyset)$
  end if
end procedure

4.6.1 Pruning Based On Anti-Monotonicity

Algorithm 11 illustrates the subroutine PRUNEDSEARCH. While in theory a breadth-first search is feasible as well, the depicted algorithm performs a depth-first search for finding the best multilabel head. Implying the algorithm recursively adds label attributes to the initially empty head of the given rule r. Meanwhile, it keeps track of the head with the best performance. Depending on the specified targets G, either only positive (e.g. $y_i = 1$) or positive as well as negative heads (e.g. $y_i = 0, y_i = 1$) are considered (cf. Algorithm 11, lines 3 and 6). Naturally, labels present in the body of the rule are not allowed to be added to the head (cf. Algorithm 11, line 3). In order to avoid unnecessary evaluations, the algorithm prunes the search according to the anti-monotonicity criterion (cf. Section 3.1). Hence, once the performance of the rule decreases by adding a label attribute, the subroutine PRUNEDSEARCH is not executed recursively any further as further additions cannot reach the best performance anymore (cf. Algorithm 11, lines 20 and 21). Instead, the algorithm continues with the next label (cf. Algorithm 11, line 3). Furthermore, the (initially empty) set H keeps track of the already evaluated heads. This prevents equivalent heads from unnecessarily being evaluated. Heads are considered to be equivalent if they contain the same label attributes, neglecting the attribute values and the order in which they appear. Excluding supersets of heads contained in H ensures that pruned heads are omitted in later iterations (cf. Algorithm 11, line 11). Due to heads being prevented from being evaluated by placing them in H, the corresponding search tree is unbalanced, where nodes of the tree correspond to the evaluation of a certain label combination and edges correspond to adding an additional label.

4.6.2 Exploiting Decomposable Evaluation Metrics

Deriving the best performing head is much more efficient if using a decomposable evaluation metric, as deep searches through the label space can be avoided. In fact, determining the best multilabel head comes at a linear cost. Given a training set with n labels, depending on whether to only learn positive heads or negative heads as well, exactly n or $2n$ single-label head rules must be evaluated, respectively. Algorithm 12 illustrates the exploitation of decomposability. Recall that decomposability states that the performance of a multilabel head can be derived by considering each single-label head performance. Furthermore, if a multilabel head contains a single-label for which the performance does not reach the best performance, the multilabel head cannot reach the best performance either. The algorithm uses this knowledge by keeping track of the so far best performing head. If the next evaluated single-label head reaches a better performance than the currently best performing head, the previous best head is discarded and the single-label head is the new best head (cf. Algorithm 12, lines 16 and 17). The previous best head can easily be discarded as it cannot be part of the best performing head anymore, as it would drag the performance of the combined head down, i.e. the performance would be worse than the performance of the single-label head. If the next evaluated single-label head reaches a worse performance, we do not need to consider it for the same reasons. If the next evaluated single-label head reaches the same performance as the currently best head, however, we can add the single-label to the so far best performing head (cf. Algorithm 12, lines 18 and 19). The combined head then reaches the same performance despite containing more labels. Implying, we combine the single-labels head which reach the maximum performance. The algorithm terminates if all single-label heads have been considered. In the case of using targets $G = \{0, 1\}$, it is asserted that only one of both label attributes is included in the head. A head cannot set a label to two different values simultaneously. Again, label conditions already contained in the body are not considered.

4.7 Measuring The Performance

Previously shown algorithms depend on the subroutine EVALUATERULE, which measures the performance of multilabel head rules. Hence, the subroutine enables the comparison of different rules to each other. Throughout this work, several parameters, which influence the evaluation of a single rule, have been presented: The evaluation function δ, the averaging strategy and whether to use a rule-dependent or

Algorithm 11 Algorithm PRUNEDSEARCH, which performs a pruned search through the label space, according to the properties of anti-monotonicity. [Rapp, 2016, Algorithm 13]

Require: Original training set T, current training set $T_{current}$, current rule r,
targets G, evaluation function δ, already considered heads H,
whether to use rule-dependent or rule-independent evaluation strategy, the averaging strategy to use

1: **procedure** PRUNEDSEARCH
2: $r_{best} = r$
3: **for each** label λ_i not already contained in $r.body$ **do** ▷ for each label, all targets are considered
4: $r_{current} = r$ ▷ currently considered label attribute
5: $r_{current}.h = -\infty$
6: **for each** target $t \in G$ **do**
7: $y_i = t$
8: **if** label attribute y_i is not already in $r.head$ **then**
9: $r_{refined} = r$
10: $r_{refined}.head = r_{refined}.head \cup y_i$ ▷ add label attribute to head
11: **if** no head in H is a subset of $r_{refined}.head$ **then** ▷ prunes label combinations
12: $r_{refined}.h = \text{EVALUATERULE}(T, T_{current}, r_{refined}, \delta)$
13: **if** $r_{refined}.h > r_{current}.h$ **then** ▷ determines the best prediction for each label
14: $r_{current} = r_{refined}$
15: **end if**
16: **end if**
17: **end if**
18: **end for**
19: **if** $r_{current}.h \neq -\infty$ **then** ▷ if label combination has not been pruned
20: **if** $r_{current}.h \geq r_{best}.h$ **then** ▷ recursively add label attributes, unless performance decreases
21: $r_{rec} = \text{PRUNEDSEARCH}(r_{current}, G, T, \delta, H)$
22: **if** $r_{rec}.h > r_{best}.h$ **or** $(r_{rec} = r_{best}.h$ **and** $|r_{rec}.head| > |r_{best}.head|)$ **then**
23: $r_{best} = r_{rec}$ ▷ heads with more label attributes are preferred
24: **end if**
25: **end if**
26: $H = H \cup r_{current}.head$
27: **end if**
28: **end for**
29: **return** rule r_{best} with best head
30: **end procedure**

Algorithm 12 Algorithm Decomposite, which exploits the properties of decomposable evaluation metrics to determine the best possible multilabel head for a specific rule. [Rapp, 2016, Algorithm 14]

Require: Original training set T, current training set $T_{current}$, current rule r, targets G, evaluation function δ,
whether to use rule-dependent or rule-independent evaluation strategy, the averaging strategy to use

1: **procedure** DECOMPOSITE
2: $r_{best} = r$
3: **for each** label λ_i not already contained in $r.body$ **do**
4: $r_{current} = r$
5: $r_{current}.h = -\infty$
6: **for each** target $t \in G$ **do** ▷ for each label, all targets are taken into consideration
7: $y_i = t$ ▷ currently considered label attribute
8: **if** label attribute y_i is not already in $r.head$ **then**
9: $r_{single} = y_i \leftarrow r.body$
10: $r_{single}.h = \text{EVALUATERULE}(T, T_{current}, r_{single}, \delta)$
11: **if** $r_{single}.h > r_{current}.h$ **then** ▷ determines the best prediction for each label
12: $r_{current} = r_{single}$
13: **end if**
14: **end if**
15: **end for**
16: **if** $r_{current}.h > r_{best}.h$ **then** ▷ if outperformed, discard previous head
17: $r_{best} = r_{current}$
18: **else if** $r_{current}.h = r_{best}.h$ **then** ▷ best performance is reached, label attribute added to head
19: $r_{best}.head = r_{best}.head \cup r_{current}.head$
20: **end if**
21: **end for**
22: **return** rule r_{best} with best head
23: **end procedure**

rule-independent evaluation strategy (cf. Section 2.3 and 3.3). These parameters influence how the confusion matrix of a rule is constructed and how each entry is combined to calculate the heuristic value. The subroutine EvaluateRule, which is depicted in Algorithm 13, measures the performance of a multilabel rule r on the training set T. At first, the set of relevant labels is retrieved using the subroutine GetRelevantLabels, which is dependent on the evaluation strategy. These are the labels that are taken into account for the subsequent performance evaluation. Depending on the chosen averaging strategy, one out of the four subroutines MicroAveraging, LabelBasedAveraging, ExampleBasedAveraging and MacroAveraging is called.

Algorithm 13 Algorithm EvaluateRule for measuring the performance of a multilabel head rule. [Rapp, 2016, Algorithm 15]

Require: Original training set T, current training set $T_{current}$, rule r, targets G, evaluation function δ, whether to use rule-dependent or rule-independent evaluation strategy, the averaging strategy to use

```
1:  procedure EvaluateRule
2:      L = GetRelevantLabels(r)
3:      if use micro-averaging then
4:          MicroAveraging(T, T_current, r, δ, L)
5:      else if use label-based averaging then
6:          LabelBasedAveraging(T, T_current, r, δ, L)
7:      else if use example-based averaging then
8:          ExampleBasedAveraging(T, T_current, r, δ, L)
9:      else if use macro-averaging then
10:         MacroAveraging(T, T_current, r, δ, L)
11:     end if
12: end procedure
```

The algorithm employs more criteria for comparing multilabel head rules than shown in Algorithm 13, which solely relies on the heuristic value. These criteria are omitted in favor of simplicity. For instance, the algorithm does not benefit from inducing rules, which do not cover any examples – regardless the heuristic value. Moreover, a situation in which two or more rules reach the same performance can occur. Hence, a strategy for dealing with this situation is required – typically referred to as *tie breaking*. Commonly, rules that cover more positives or contain fewer conditions in their body are preferred. If those additional criteria are not sufficient to determine a better rule, one rule is chosen randomly.

Algorithm 14 Algorithm GetRelevantLabels for retrieving all relevant labels according to the used evaluation strategy. [Rapp, 2016, Algorithm 16]

Require: Rule r, whether to use rule-dependent or rule-independent evaluation strategy

```
1: procedure GetRelevantLabels
2:     if use rule-dependent evaluation strategy then
3:         return {λ_i | λ_i ∈ 𝕃 ∧ y_i ∈ r.head}      ▷ return labels, which are predicted by the given rule
4:     else
5:         return {λ_i | λ_i ∈ 𝕃}                      ▷ return all available labels
6:     end if
7: end procedure
```

Algorithm 14 illustrates the subroutine GetRelevantLabels, which retrieves all labels that must be used in the evaluation of a rule. Which labels are returned is dependent on the utilized evaluation strategy. For a rule-dependent evaluation the algorithm returns the labels contained in the head of the rule, in

confirmation with the definition in Section 3.3. For a rule-independent evaluation the algorithm returns the complete set of labels $\mathbb{L}$, regardless of the rule.

All four subroutines MicroAveraging, LabelBasedAveraging, ExampleBasedAveraging and MacroAveraging have to resort to the subroutine Aggregate for building a confusion matrix (cf. Algorithm 15). The algorithm calculates the atomic confusion matrix for a prediction of a rule and a label of a given training example, i.e. it determines whether or not the prediction is counted as a true positive, false positive, true negative or false negative. As a follow up, the respective entry in the confusion matrix is increased by one. The algorithm differentiates between rules that cover an example and rules that do not. If a rule does not cover an example, relevant labels are counted as false negatives, whereas irrelevant labels are counted as true negatives. If a rule does cover an example, the prediction of the rule for the regarded label needs to be considered. In such cases, the prediction is counted as a true positive if it is correct, regardless whether the label is a relevant or an irrelevant one. Similarly, an incorrect prediction is counted as a false positive. Note that this definition contradicts the definition given in Section 2.3.1, which is supposed to be used for evaluating the quality of a rule set on a test data set. The definition given in this section, however, is supposed to be used for evaluating the performance of an individual rule during the rule induction process. Rapp [2016] states that the reason for the altered semantics is the believe that every label present in the head of a rule should be treated equally and that relevant labels should not be preferred over irrelevant ones. For demonstrating the intention behind this change, consider using precision as an evaluation function and the definition in Section 2.3.1. A label attribute $y_i = 0$ in the head of a rule would never have a positive impact on said rule. This is the case due to precision being exclusively calculated from true and false positives as well as the prediction being counted as either a true or false negative.

4.7.1 Micro-Averaging

The subroutine MicroAveraging is called if micro-averaging is to be used for the performance evaluation of a rule. The subroutine is depicted in Algorithm 16 and performs according to to the definition of micro-averaging in Section 2.3.1. The algorithm iterates over all examples in the training set. For each example, the covering status determines whether or not the example is included in the construction of the global confusion matrix. Covered examples for which all labels are already predicted are excluded from the building process (cf. Algorithm 16, line 4). This check can be performed on the current training set $T_{current}$. The exact same condition, which was used in by Loza Mencía and Janssen [2016] in their separate-and-conquer algorithm, is also utilized in the subroutines LabelBasedAveraging, ExampleBasedAveraging and MacroAveraging. It was introduced in order to prevent identical rules from being learned in subsequent iterations of the algorithm in the case that no examples are removed from the training data set after inducing a new rule. If an example is eligible to be taken into account for the construction of the confusion matrix, the subroutine Aggregate is called for every relevant label that was retrieved by the subroutine GetRelevantLabels. Eventually, Algorithm 16 applies the evaluation function δ to the built global confusion matrix.

4.7.2 Label-Based Averaging

Algorithm 17 illustrates the label-based averaging, which was introduced in Section 2.3.2. For every averaging strategy subroutine presented in this work, the main components are very similar. However, they differ in the order of execution. For label-based averaging, do not build a global confusion matrix but one for each relevant label. For each label, the algorithm iterates over all examples and calls the subroutine Aggregate if the previously described conditions are met. Eventually, the evaluation function δ is applied to the label confusion matrix and the individual heuristic for this label is added to a global value. In the end, the global heuristic value is divided by the number of labels in order to average the performance label-wise.

Algorithm 15 Algorithm AGGREGATE for aggregating the true positives, false positives, true negatives and false negatives of a confusion matrix. [Rapp, 2016, Algorithm 17]

Require: Rule r, training example $(X_j, \hat{Y}_j)$, label λ_i, confusion matrix C

procedure AGGREGATE
  if r covers X_j **then**
    if $r.head$ contains a label attribute y_i **then**
      if $y_i \neq \hat{y}_i \in \hat{Y}_j$ **then**
        $C.fp+ = 1$ ▷ prediction is incorrect
      else
        $C.tp+ = 1$ ▷ prediction is correct
      end if
    else
      if $\hat{y}_i \in \hat{Y}_j$ **then** is set
        $C.fp+ = 1$ ▷ relevant label predicted as irrelevant
      else
        $C.tp+ = 1$ ▷ irrelevant label predicted as irrelevant
      end if
    end if
  else
    if $\hat{y}_i \in \hat{Y}_j$ is set **then**
      $C.fn+ = 1$
    else
      $C.tn+ = 1$
    end if
  end if
end procedure

Algorithm 16 Algorithm MICROAVERAGING for measuring the performance of a multilabel head rule using micro-averaging. [Rapp, 2016, Algorithm 18]

Require: Original training data set T, current training data set $T_{current}$,
rule r, evaluation function δ, relevant labels L

procedure MICROAVERAGING
  $C = (0, 0, 0, 0)$ ▷ initialize global confusion matrix
  for each example $(X_j, \hat{Y}_j) \in T$ **do**
    if r does not cover X_j **or** not all labels $y_i \in r.head$ are set in $Y_j \in T_{current}$ **then**
      for each relevant label $\lambda_i \in L$ **do**
        AGGREGATE$(r, (X_j, \hat{Y}_j), \lambda_i, C)$
      end for
    end if
  end for
  $h = \delta(C)$ ▷ apply evaluation function on confusion matrix
  return performance h
end procedure

Algorithm 17 Algorithm LABELBASEDAVERAGING for measuring the performance of a multilabel head rule using label-based averaging. [Rapp, 2016, Algorithm 19]

Require: Original training data set T, current training data set $T_{current}$,
rule r, evaluation function δ, relevant labels L

procedure LABELBASEDAVERAGING
  $h = 0$
  for each relevant label $\lambda_i \in L$ **do**
    $C = (0, 0, 0, 0)$ ▷ initialize confusion matrix for each label
    for each example $(X_j, \hat{Y}_j) \in T$ **do**
      if r does not cover X_j **or** not all labels $y_i \in r.head$ are set in $Y_j \in T_{current}$ **then**
        AGGREGATE$(r, (X_j, \hat{Y}_j), \lambda_i, C)$
      end if
    end for
    $h += \delta(C)$ ▷ apply evaluation function on confusion matrix
  end for
  $h = h/|L|$ ▷ average performance label-wise
  return performance h
end procedure

4.7.3 Example-Based Averaging

While the subroutine LABELBASEDAVERAGING constructs a confusion matrix for every label and then averages the performance over those, the subroutine EXAMPLEBASEDAVERAGING, whose operation is illustrated in Algorithm 18, constructs a confusion matrix for every example and then averages over those. Again, true positives, false positives, true negatives and false negatives are counted using the subroutine AGGREGATE. Furthermore, the same conditions apply as for MICROAVERAGING and LABELBASEDAVERAGING in order to prevent identical rules from being learned in subsequent iterations of the algorithm. The operation of the algorithm corresponds to the definition of example-based averaging given in Section 2.3.2.

Algorithm 18 Algorithm EXAMPLEBASEDAVERAGING for measuring the performance of a multilabel head rule using example-based averaging. [Rapp, 2016, Algorithm 20]

Require: Original training data set T, current training data set $T_{current}$,
rule r, evaluation function δ, relevant labels L

procedure EXAMPLEBASEDAVERAGING
  $h = 0$
  for each example $(X_j, \hat{Y}_j) \in T$ **do**
    if r does not cover X_j **or** not all labels $y_i \in r.head$ are set in $Y_j \in T_{current}$ **then**
      $C = (0, 0, 0, 0)$ ▷ initialize confusion matrix for each example
      for each relevant label $\lambda_i \in L$ **do**
        AGGREGATE$(r, (X_j, \hat{Y}_j), \lambda_i, C)$
      end for
    end if
    $h += \delta(C)$ ▷ apply evaluation function on confusion matrix
  end for
  $h = h/|T|$ ▷ average performance example-wise
  return performance h
end procedure

4.7.4 Macro-Averaging

The final averaging strategy is macro-averaging, which averages over both labels and examples. The operation of the subroutine MacroAveraging is illustrated in Algorithm 19. At first, an atomic confusion matrix is constructed for each example and label. Subsequently, the heuristic value for this atomic confusion matrix, which is obtained by applying the evaluation function δ, is added up for each label within the example and the arithmetic mean is calculated. Similarly, after iterating over all examples, the added up heuristic value is averaged example-wise. Hence, the heuristic value is averaged over both labels and examples. The calculation could also be done the other way around, i.e. first iterate over all labels and then for each label iterate over all examples and consequently first average example-wise followed by a label-wise averaging. However, both possibilities return the same result as the averaging is commutative. Again, examples for which all labels are already set by previous rules are omitted from the confusion matrix building process.

Algorithm 19 Algorithm MacroAveraging for measuring the performance of a multilabel head rule using macro-averaging. [Rapp, 2016, Algorithm 21]

Require: Original training data set T, current training data set $T_{current}$,
rule r, evaluation function δ, relevant labels L

procedure MacroAveraging
  $h = 0$
  for each example $(X_j, \hat{Y}_j) \in T$ **do**
    if r does not cover X_j **or** not all labels $y_i \in r.head$ are set in $Y_j \in T_{current}$ **then**
      $h_j = 0$
      for each relevant label $\lambda_i \in L$ **do**
        $C = (0, 0, 0, 0)$ ▷ initialize confusion matrix for each example and label
        Aggregate$(r, (X_j, \hat{Y}_j), \lambda_i, C)$
        $h_j += \delta(C)$ ▷ apply evaluation function on confusion matrix
      end for
      $h += h_j / |L|$ ▷ average performance label-wise
    end if
  end for
  $h = h / |T|$ ▷ average performance example-wise
  return performance h
end procedure

5 Discovery of Multilabel Rules by Relaxed Pruning

This chapter explains the basics of discovering multilabel rules by relaxed pruning. At first, the general idea behind relaxing the pruning constraints is outlined. Afterwards, relaxation lift functions are introduced as a way of performing the relaxation. Furthermore, three different types of functions are suggested and shortly theoretically compared.

5.1 Relaxing The Pruning Constraints

Multilabel classification may greatly benefit from exploiting dependencies between labels [Loza Mencía and Janssen, 2016]. As the binary relevance approach does not consider these dependencies and only considers each label independently, it is not capable of modeling label dependencies. However, several changes have been made in order to enable exploiting label dependencies, such as classifier chains for instance [Zhang et al., 2018; Read et al., 2011]. These approaches are somewhat restricted in the order in which they learn label dependencies, i.e. they cannot learn dependencies in both directions. For instance, a classifier chain may learn a dependency $\lambda_i \rightarrow \lambda_k$ but it is impossible to learn the reverse relationship in the same model. Hence, Loza Mencía and Janssen [2016] argue that for full expressiveness, multilabel head rules in combination with label conditions are required. Rapp [2016] therefore adapted the single-label head rule algorithm proposed by Loza Mencía and Janssen [2016] in order to learn multilabel head rules. The basic idea is to find the best possible head (out of all possible heads, i.e. all label combinations) for a given rule body in the hope that his leads to the best model. His work is especially concerned with the efficiency of searching through the space of possible (multilabel) heads. The label search space is pruned according to the decomposability and anti-monotonicity characteristics [Rapp et al., 2018]. Therefore guaranteeing finding the best possible head and pruning as much as possible. However, Rapp [2016] concluded that using this algorithm, in most cases, the best possible head is a single-label head, unless using the precision metric as a baseline learner. As precision is not necessarily the best metric for learning rules, in his future work section he proposed introducing a bias towards longer heads by relaxing the pruning constraints. The idea being that a longer multilabel head is tolerated if the heuristic value of the rule does not decrease significantly.

h	Rule
0.479	red = 1 ← landmass = 3
0.475	white = 1 ← landmass = 3

Table 6: Example of two single-label heads for the same body, taken from the rule learning process performed on the data set FLAGS.

A good but extreme example of why a relaxed pruning approach is needed is depicted in Table 6. The example shows two single-label head rules that were evaluated for the same body. The second rule has a just minimally worse heuristic value. Pruning based on decomposability would, however, learn the first head as it is the one with the highest heuristic value and combining it with the second head would worsen the heuristic value. Especially for this example it makes sense to learn the combination of both heads, despite the decline in the heuristic value. By learning a longer head, coverage is increased as two labels in the head cover twice the number of entries in the training set matrix in comparison to a single-label head rule. Hence, we trade off coverage and the heuristic quality of the rule on the training set. Thus, we relax the strict pruning constraints of the original algorithm so that we allow the induction of longer heads if the performance of the rule does not decrease significantly.

5.2 Challenges

Several challenges arise by introducing a bias towards multilabel head rules. For once, it may be hard to decide how much of a lift to apply, i.e. how much of a decrease in the heuristic value is tolerable.

If the lift is too small, a potentially good rule might not be generated. If the lift is too big, a seemingly bad rule might be discovered. Considering there are many such cases during a training phase, even a lift difference of only one percent can have a huge impact. Furthermore, the question arises of how many labels in the head are desired or optimal – where to draw the line? Sometimes a rule with five labels in the head may seem good. Other times, one with two is bad. As the label cardinality (average number of labels per example) of data sets is usually rather small, learning of too large multilabel head rules must be prevented. For instance, for a data set with 100 labels and a label cardinality of five it might be unwise to learn rules with 50 labels in the head. When also learning negative heads, determining the right amount of labels in the head becomes even more difficult. In addition, efficiency must be considered as it is not possible to find the best possible lifted head and simultaneously prune as Rapp et al. [2018] does. This problem is further discussed and solutions are presented in Section 6.

5.3 Relaxation Lift Functions

In this work, we relax the pruning constraints by lifting the heuristic value of each rule depending on the number of label in its head. The lift is performed by multiplying the heuristic value with a certain multiplier, which we call *relaxation lift*. In order to introduce a bias towards rules with longer multilabel heads, the extent of the relaxation lift must increase with the number of labels in the head. Hence, offsetting the decline in the heuristic value, which commonly occurs by extending the head. Due to the multiplication, each rule is lifted relative to its performance. Implying, rules with a high heuristic value are lifted more than rules with low heuristic values. Therefore, a relaxation lift of 1.00 does not lead to any changes in the heuristic value, a relaxation lift of smaller than 1.00 and greater than 1.00 decreases and increases the heuristic value, respectively. In order to reasonably search through the label space, we need to define a relaxation lift for every possible head length. For a data set with n labels this amounts to defining a relaxation lift for labels $1, \ldots, n$. To simplify the process of defining the extent of the relaxation lift, we use *relaxation lift functions*. Using this concept, other properties can be expressed comfortably as well. Formally, a relaxation lift function

$$\rho : \mathbb{R}_+ \to \mathbb{R}$$

maps a number of labels to a number representing the relaxation lift. Naturally, the number of labels in the head cannot be negative. Although the function is only evaluated for whole numbers (heads with 2.5 labels do not exist), expressing the function with real numbers facilitates the definition. By expressing the relaxation lift in terms of a relaxation lift function, we do not have to define the lift individually for each number of labels. Instead, parameterizable functions enable an easier handling of the relaxation lift values and serve as an orientation. Using this definition, relaxation lift functions can be visualized by portraying the number of labels on the x-axis and portraying the corresponding relaxation lift on the y-axis. In the remainder of the work, we will often refer to a relaxation lift function as a lift function and to relaxation lift as lift. Furthermore, when applying a relaxation lift of 1.17 for instance, we often refer to it as a lift of 17 percent.

5.4 Lifted Heuristic Value

In the remainder of this work, we distinguish between two types of heuristic values – the normal heuristic value and the lifted heuristic value. The *lifted heuristic value* is the value that emerges from multiplying the normal heuristic value h of a rule r with the relaxation lift. As a result, the lifted heuristic value, denoted by $\vec{h}$, of the heuristic value h, obtained by applying a multilabel evaluation function, of a rule r with x labels in its head is defined as

$$\vec{h} = h \cdot \rho(x)$$

The lifted heuristic value is then used to find the best performing head. By using this new value, we change the definition of "best performing". Previously, finding the best performing head amounted to

finding the head with the best (normal) heuristic value. Form now on, finding the best performing head amounts to finding the head with the best lifted heuristic value. This is a major difference. In Section 6 we will see further consequences of utilizing the lifted heuristic value for finding the best head.

An example may illustrate the concept of lift and lift function. Let us consider the three rules with different head lengths that are depicted in Table 7. By adding more labels to the head, the heuristic value decreases. Furthermore, we assume a lift function that prefers heads with three labels over heads with two labels. Then, we can calculate the lifted heuristic values as shown in Table 7.

Head	h	Relaxation Lift	$\vec{h}$
red = 1	0.70	$\rho(1) = 1.00$	$0.7 \cdot 1.0 = 0.7$
red = 1, white = 1	0.67	$\rho(2) = 1.07$	$0.67 \cdot 1.07 = 0.7169$
red = 1, white = 1, orange = 0	0.63	$\rho(3) = 1.12$	$0.63 \cdot 1.12 = 0.7056$

Table 7: Calculation of the lifted heuristic value $\vec{h}$ by multiplying the heuristic value h with the relaxation lift function value for three rules of different head lengths.

As a result, despite having a (slightly) worse heuristic value than the first rule, the second rule is better performing in terms of lifted heuristic value and is considered to be better by the find best head subroutine as it covers more labels. Nevertheless, the head of length three does not get chosen as the decline in the heuristic value is too big to be offset by this specific relaxation lift.

5.5 Desired Characteristics of Relaxation Lift Functions

In this section, we shortly and rather informally describe some desired characteristics of relaxation lift functions. Our intention is to learn more multilabel heads. Therefore, we only lift heads with more than one label. Implying the relaxation lift for a head length of one should have no effects on the heuristic value. Hence, $\rho(1) = 1$. Consequently, when visualizing a lift function, it should be fixed in $(1, 1)$. Moreover, in order to prevent too large heads from being learned, the relaxation lift function should ideally flatten with more labels. Implying the increase in the lift should decrease the more labels are added. Formally, this implies

$$\forall x : (x > 1 \wedge x < n) \rightarrow (\rho(x) - \rho(x-1) \geq \rho(x+1) - \rho(x))$$

Furthermore, the lift function should be easily computable to ensure minimal overhead in that regard. In Section 6 we will see that it is important for pruning that the lift function is (easily) computable in advance.

5.6 The Peak Relaxation Lift Function

The *peak relaxation lift function* takes the desired characteristic of preventing too many labels in the head to an extreme. The function starts off by monotonously increasing the lift up until the *peak*, the point with the highest lift. Starting from the peak, however, the lift starts to (monotonously) decline again. Implying, heads with the peak amount of labels in the head are allowed to have a bigger decline in the heuristic value than heads with more labels. The peak lift function is a parameterizable function that can be used to specifically target a desired number of labels in the head. Parameter $m \geq 1$ defines the number of labels for which the relaxation lift is maximal – the *peak label*. As a result, $\rho(m)$ is the highest achievable lift. Parameter $l \geq 1$ defines the relaxation lift at the peak label, i.e. $\rho(m) = l$. Finally, the curvature can be set using the parameter $k \geq 0$. The higher k, the steeper the curvature and the lower, the flatter the curvature. Note that $k = 1$ corresponds to a linear gradient. Moreover, the function requires the number of labels in the data set n. The peak relaxation lift function is then defined as

$$\rho_{peak}(x) = \begin{cases} f(x) & \text{if } x \leq m \\ g(x) & \text{if } x > m \end{cases}$$

Figure 3: The function $x^{1/k}$ for the values $k = 2.2$ (left) and $k = 0.6$ (right).

$$f(x) = 1 + (\frac{x-1}{m-1})^{1/k} \cdot (l-1) \tag{5}$$

$$g(x) = 1 + (\frac{n-x}{n-m})^{1/k} \cdot (l-1) \tag{6}$$

In order to further the understanding of this function, we will cover the individual components in more detail. Initially, this function was a logarithmic function. By setting parameters, the peak label and the lift could be changed. However, allowing the curvature to be adjusted posed a problem. As a result, the foundation of the peak label relaxation function needed to be changed. Figure 3 depicts the basic component of the new function. Adjusting the curvature using $x^{1/k}$ becomes very easy – just change parameter k. By stretching and shifting the depicted function it is possible to model the desired peak label function with parameters m and l. First, we shift the function by one so that the lift for a single label is always one. Moreover, the desired property of $x^{1/k}$ is limited to $[0,1]$ on the x- and y-axis. Hence, we stretch the function in the direction of the y-axis by $(l-1)$. By doing so, we achieve the desired maximal lift. Note that we only need to stretch by $(l-1)$ and not l due to previous shift by one. At last, we stretch the function in the direction of the x-axis by normalizing the scale according to

$$\frac{x - x_{min}}{x_{max} - x_{min}}$$

For $g(x)$, the denominator is further multiplied by -1 as from the peak onward the function should decrease. Figure 4 shows that the relaxation lift function is not differentiable at the peak. As the function is only evaluated for whole numbers and as we have no intention to differentiate it, this does not pose a problem.

5.7 The KLN Relaxation Lift Function

Another function that can be used to relax the pruning constraints is the *KLN relaxation lift function*. In comparison to the peak lift function, it is a rather simple function. The KLN lift function consists of a natural logarithm, which is multiplied with a parameter $k \geq 0$. In order to ensure that single-label heads are not lifted, we shift the function by one in the direction of the y-axis. In this regard, note that $\ln(1) = 0$. Thus, the KLN relaxation lift function is defined as

$$\rho(x)_{KLN} = 1 + k \cdot \ln(x)$$

The higher the parameter k, the higher the extent of the lift. Typically, k should under no circumstances have to be bigger than one. Due to the natural logarithm, the function flattens with an increasing number of labels, as described in Section 5.5. Figure 5 (1) illustrates the KLN relaxation lift function.

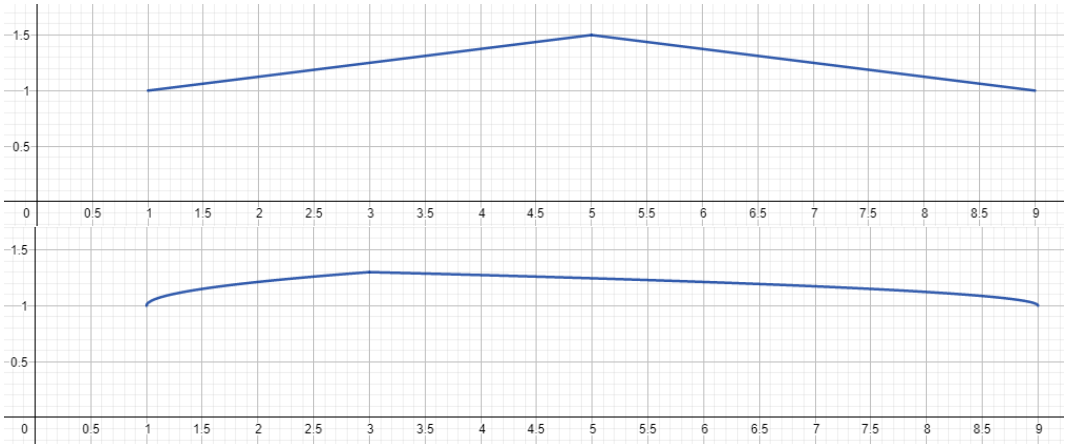

Figure 4: The peak relaxation lift function with parameters (1) $m = 5$, $l = 1.5$, $k = 1.0$ and $n = 9$ and (2) $m = 3$, $l = 1.3$, $k = 2.0$ and $n = 9$.

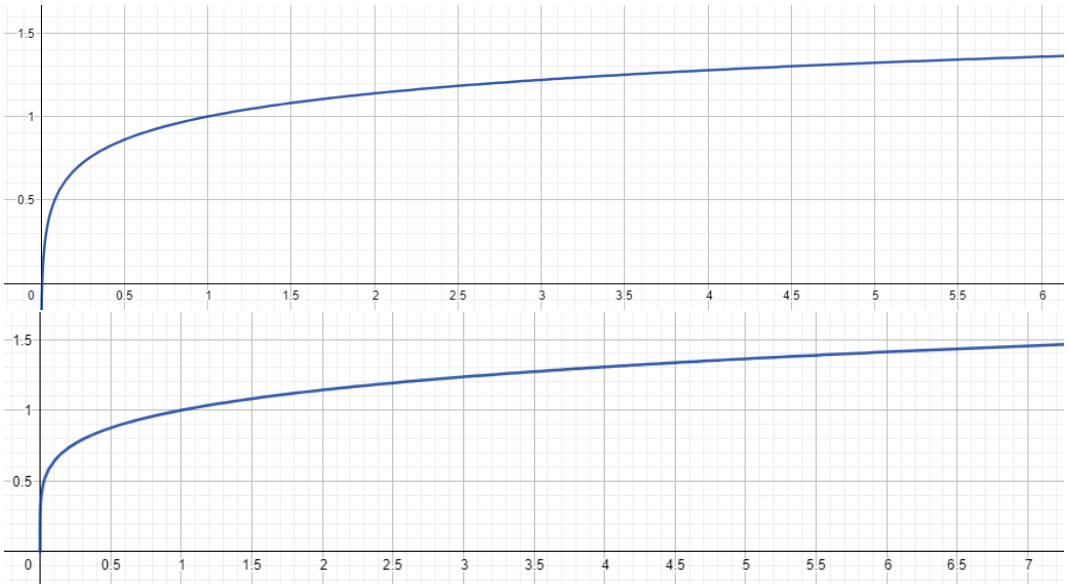

Figure 5: The (1) KLN relaxation lift function with parameter $k = 0.2$ and (2) the root relaxation lift function with parameter $k = 5.2$.

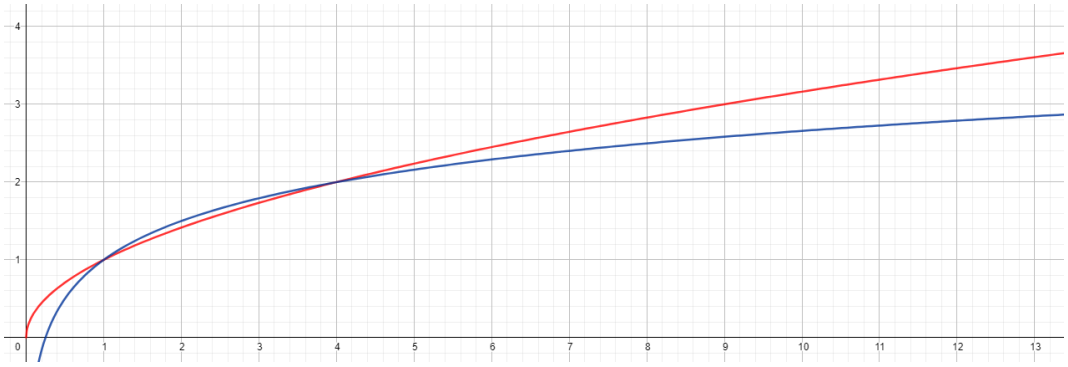

Figure 6: The difference between the KLN (blue) and the root (red) relaxation lift function. The lift is set to be the same for both functions – a lift of two at label four.

Defining the relaxation lift by setting parameter k is not intuitive. Therefore, the KLN relaxation lift function is typically set by stating a label x and a desired lift for said label l, i.e. $\rho_{KLN}(x) = l$. The parameter is then given by

$$k = \frac{l-1}{\ln(x)}$$

5.8 The Root Relaxation Lift Function

The last relaxation lift function is called the *root relaxation lift function*. The origin of the name lies in the usage of the root function. This function is quite similar to the KLN relaxation lift function. The root relaxation lift function is defined as

$$\rho_{root}(x) = x^{1/k}$$

where parameter $k \geq 1$ defines the extent of the lift. A higher value of k implies a lower relaxation lift. Note that this is the complete function of the function depicted Figure 3. The root relaxation lift function is visualized in Figure 5 (2). Again, setting k directly is not intuitive. As a result, by stating a label x and a desired lift for said label l, i.e. $\rho_{root}(x) = l$, the parameter is given by

$$k = \frac{\ln(x)}{\ln(l)}$$

5.9 Comparing The KLN and Root Relaxation Lift Function

Theoretically, the KLN and root relaxation lift functions differ in their curvature. Figure 6 visualizes the difference between both functions. For doing so, the lift has been set to the exact same value for both functions (a lift of two at label four). Hence, parameters are $k = 0.721$ for the KLN and $k = 2$ for the root relaxation lift function. We can observe that the lift increase is steeper for the KLN lift function. At the defined lift, both functions intersect. After that, the root function rises much quicker in value. However, such high lift values do not usually occur in practice. Therefore, Figure 7 illustrates the KLN and the root lift function for a lower lift – a lift of 1.3 at label four. This is a much more realistic setting, although the lift is still rather high. Nevertheless, we can observe that both functions are practically identical in the relevant section (from one onward) until the defined lift. The difference is so small it can be neglected. Afterwards, the root lift function rises more quickly again. Hence, implying too big of a lift. As a result, we will limit ourselves to utilizing the KLN relaxation lift function in the evaluation in Section 9.

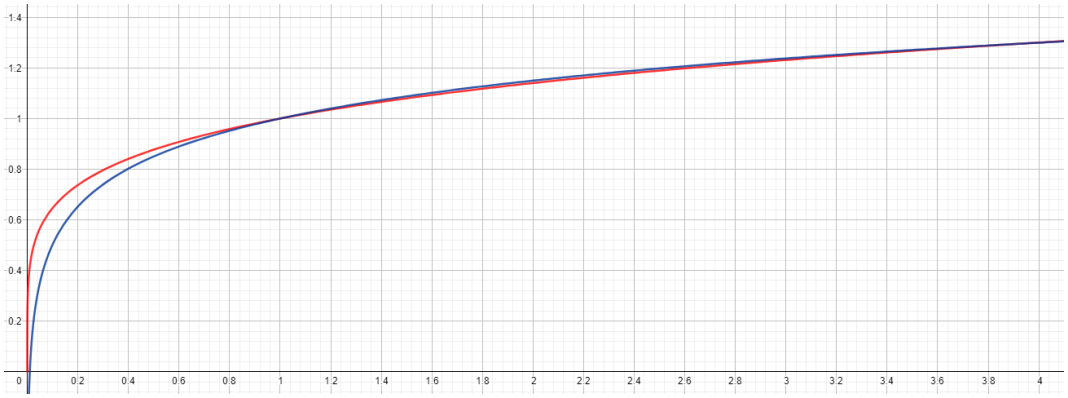

Figure 7: The similarity between the KLN (blue) and the root (red) relaxation lift function. The lift is set to be the same for both functions – a lift of 1.3 at label four.

5.10 Comparing The Peak Relaxation Lift Function

The big advantage of the peak relaxation lift function is its efficiency, especially for a small peak label. Due to the peak and the corresponding decline of the lift, we can prune heavier. Furthermore, we can define the curvature of the peak relaxation lift function – something which is not possible for the others. The peak lift functions also allows us to specifically target a number of labels in the head in a simple way. A disadvantage of the peak relaxation lift function is that it is more complex. Moreover, the peak relaxation lift function might be more susceptible to mistakes. Due to the importance of the peak label and the additional definition of the curvature, setting the correct values is more difficult. In contrast, setting the lift for the KLN and root lift functions is rather simple and requires only minimal knowledge of the concept. Nevertheless, the KLN and root relaxation lift functions are more susceptible to lifting too much as the lift continues to increase, especially in the case of the root lift function.

5.11 Discussion

We have suggested three relaxation lift functions – the peak lift function, the KLN lift function and the root lift function. All three relaxation lift functions can be used to lift the label search space, therefore preferring longer multilabel heads and enable an easier definition of the extent of the lift than setting the relaxation lift for each label individually. Depending on the data set at hand and the desired number of labels in the head, the choice which relaxation function to use varies. We have furthermore described advantages and disadvantages of each function. While configurating the KLN and root relaxation lift function is easier, the peak relaxation lift function can be used to specifically limit the number of labels in the head and thus increase efficiency of the search for the best performing head. Due to the similarity of the KLN and root relaxation lift function, we restrict the evaluation to the KLN relaxation lift function.

6 Pruning A Lifted Label Search Space

This chapter explains why we cannot prune a lifted label search space as easily as a normal label search space and further outlines how we perform the pruning for anti-monotonous and decomposable evaluation functions. Moreover, additional changes to the separate-and-conquer algorithm are presented.

6.1 Suboptimal Pruning

By changing the definition of the best performing head to be the head with the highest lifted heuristic value, we cannot prune as easily as proposed by Rapp [2016]. The reason being that the definition of anti-monotonicity does not hold for the lifted heuristic value. A counter-example for pruning a lifted label search space with anti-monotonicity can be found in Table 8. If we apply pruning according to anti-monotonicity in this example, we would evaluate the heads of length one and two and then stop because from length one to two the lifted heuristic value decreased. However, the lifted heuristic value does increase for the head of length three. Hence, pruning a lifted search space simply with anti-monotonicity does not guarantee finding the best performing head. Similarly, we can also construct counter-examples for higher depths, i.e. that the heuristic value decreases for a certain number of times before reaching its highest value. Thus, we must use the fact that anti-monotonicity holds for the normal heuristic value for pruning the lifted label search space.

Head	h	Lift	$\vec{h}$
$\{y_1\}$	0.8	1	0.8
$\{y_1, y_2\}$	0.65	1.2	0.78
$\{y_1, y_2, y_3\}$	0.63	1.3	0.819

Table 8: Counter-example for pruning with anti-monotonicity while using lifted heuristic values for searching the best performing head.

Exploiting decomposability in the same way as proposed by Rapp [2016] is not possible either. Previously, we could determine the head with the highest heuristic value by evaluating all single-label heads and then combining the single-label heads that all reach the highest performance. With a lifted label search space this is not possible as easily. The same strategy cannot be employed as combining the highest performing single-label head with a worse performing single-label head might still increase the lifted heuristic value as the combined head is multiplied with the relaxation lift.

Head	h	Lift	$\vec{h}$
$\{y_1\}$	0.8	1	0.8
$\{y_2\}$	0.75	1	0.75
$\{y_1, y_2\}$	0.775	1.1	0.8525

Table 9: Counter-example for determining the best performing head by exploiting decomposability while using lifted heuristic values for searching the best performing head.

A counter-example for this scenario is given in Table 9. The algorithm by Rapp [2016] would determine the head consisting of y_1 to be the best performing head as it reaches the highest heuristic value and cannot be combined with y_2 as the heuristic value of the combined head would decrease in comparison to the single-label head. Using the lifted heuristic value, however, the combined head is lifted and hence is considered better performing than either single-label head. Hence, we additionally need an adjusted algorithm for finding the best head while taking advantage of decomposability.

6.2 Decomposability

Rapp [2016] exploits decomposability by first evaluating every possible single-label head rule, determining the best performance and finally combining all heads that reach the best performance. Pruning in such a way is possible because decomposability guarantees that the performance of a rule decreases by adding another worse performing label to its head. As a result of using the lifted heuristic value, however, we cannot exploit decomposability in the same way as shown in Algorithm 12. While we can still be sure that the normal heuristic value decreases by adding a worse performing label to the best performing head, the same does not hold for the lifted heuristic value. Due to the applied relaxation lift, the performance in terms of lifted heuristic value may increase by adding a label to the head of a rule. Even if the label that is added has a worse performance than the head it is added to. Hence, another way of exploiting decomposability is required.

For exploiting decomposability in order to prune a lifted label search space, two observations are important. First, the best lifted head of length n results from applying the lift to the head with the highest normal heuristic value of length n. As all heads of length n are multiplied with the same lift, a head of length n with a worse normal heuristic value cannot possibly achieve a higher lifted heuristic value. Therefore, by determining the best head of a certain length in terms of normal heuristic value, we additionally attain the best lifted head of that length as well. Second, we can construct the best head of length n by combining the n best single-label heads. Table 10 underlines the construction of the best head of length n. If we combined y_1 with y_3 instead of y_2, the heuristic value would be 0.37 – a worse performing head than $\{y_1, y_2\}$. By using these two observations, we can exploit decomposability for pruning a lifted label search space.

n	Head	h	Single-Label Head	h
1	$\{y_1\}$	0.390	$\{y_1\}$	0.39
2	$\{y_1, y_2\}$	0.380	$\{y_2\}$	0.37
3	$\{y_1, y_2, y_3\}$	0.370	$\{y_3\}$	0.35
4	$\{y_1, y_2, y_3, y_4\}$	0.353	$\{y_4\}$	0.30
5	$\{y_1, y_2, y_3, y_4, y_5\}$,	0.336	$\{y_5\}$	0.27

Table 10: Constructing the best head of length n by combining the n best single-label heads.

6.2.1 An Algorithm For Exploiting Decomposability With Relaxed Pruning

Algorithm 20 illustrates the operation of finding the best lifted head while exploiting decomposability of multilabel evaluation metrics. We start off by evaluating all possible single-label heads for the given body and ordering them in decreasing order according to their normal heuristic value (cf. Algorithm 20, line 2). In the case that two or more rules have the same heuristic value, we use the same tie breaking as described in Section 4.7. We furthermore keep track of the best candidate so far and initialize the current candidate with an empty head rule (cf. Algorithm 20, lines 3 and 4). In the worst case, we need to construct all n heads – one for each possible head length. In each iteration of the for-loop (cf. Algorithm 20, line 5), we add the best performing remaining label to the current candidate using the subroutine RefineCandidate, i.e. we construct the best performing head of the current length. Additionally, we update the best lifted head if need be (cf. Algorithm 20, lines 7 and 8). In each iteration, we check whether or not a higher lifted heuristic value is still achievable using the subroutine PrunableWithDecomposability. If a higher lifted heuristic value is possible, the algorithm constructs the next longer head. Otherwise, the algorithm terminates and returns the highest performing multilabel rule for the given body. Furthermore, we add a lower boundary for the quality of the rule. Each rule must have at least as many true positives as false positives in order to be eligible to be induced. Hence, we

filter out rules that make more mistakes than correct predictions. By introducing this condition, we also decrease the chances of inducing too bad rules due to a mistakenly set too high lift. By setting too high of a relaxation lift, a label that would otherwise would not be learned, could theoretically be induced if it is combined with a very good label. Additionally, we do not want to make too bad predictions as we cannot correct wrong predictions later on. The condition of requiring at least as many true positives as false positives will play an important role in Section 8. Note that the condition is not explicitly depicted in the algorithm in favor of simplicity and clearness.

Algorithm 20 Algorithm DECOMPOSABLE, which exploits decomposability to determine the head with the highest lifted heuristic value.

```
1:  procedure DECOMPOSABLE(∅ ← B)
2:      S := {λ = t ← B : λ ∈ 𝕃 ∧ t ∈ {0,1}}        ▷ sorted single label heads for each target value
3:      c := ∅ ← B
4:      c_best := c
5:      for 1,...,n do                                  ▷ for all head lengths
6:          c = REFINECANDIDATE(S, c)
7:          if c.h⃗ ≥ c_best.h⃗ then                      ▷ update best lifted head
8:              c_best = c
9:          end if
10:         if PRUNABLEWITHDECOMPOSABILITY(c_best, c) then
11:             return c_best                           ▷ return rule if TP ≥ FP
12:         end if
13:     end for
14:     return c_best          ▷ return rule with the highest lifted heuristic value if TP ≥ FP
15: end procedure
```

The subroutine REFINECANDIDATE, which is depicted in Algorithm 21, is responsible for adding the best remaining label to the head of the given rule. The algorithm first retrieves the best remaining label from the sorted set (cf. Algorithm 21, line 2). Naturally, all labels that are already present in the head are not eligible to be added, regardless the label condition. Implying, the algorithm checks for a label λ_i whether or not a label attribute $\lambda_i = 0$ or $\lambda_i = 1$ is already contained in the head. This check is important as a head can only contain one label attribute y_i for a label λ_i. Afterwards, the head is extended by the best remaining label condition (cf. Algorithm 21, line 3). Finally, the rule is evaluated, the lifted heuristic value is set (cf. Algorithm 21, line 4) and the new candidate is returned.

Algorithm 21 Algorithm REFINECANDIDATE, which extends the head of a rule by merging it with the best remaining single-label head.

Require: sorted single label heads for each target value in decreasing order S, rule r

```
1:  procedure REFINECANDIDATE
2:      s = TOP(S \ r.head)     ▷ get best remaining single label target (excluding labels already in head)
3:      c = (r.head ∪ s) ← r.body                       ▷ add to head
4:      c.h⃗ = EVALUATE(c)
5:      return c
6:  end procedure
```

Algorithm 22 shows how a rule is evaluated. At first, the rule is evaluated as usual using the subroutine EVALUATERULE, which evaluates the rule according to the chosen averaging and evaluation strategy as well as evaluation function. Following the evaluation, the lifted heuristic value is calculated from the heuristic value and the chosen relaxation lift function (cf. Algorithm 22, line 3). Finally, the rule value is set to the heuristic or lifted heuristic value and the lifted heuristic value is returned. The need for this distinction

might be unclear at first. While finding the best head for a given body, we consistently utilize the lifted heuristic value as a performance measure. However, after finding the best head, we have two choices of how to rate a rule. According to this rating (the rule value), the best rule among all rules in this iteration is chosen (cf. Algorithm 5). By setting the rule value to either the normal or lifted heuristic value, the remaining algorithm for searching the best rule remains unchanged. An example may illustrate the distinction further.

Rule	h	$\vec{h}$
orange = 1 ← landmass = 3	0.7	0.7
red = 1, white = 0 ← area ≥ 0	0.72	0.80
blue = 0, yellow = 1, green = 1 ← language = 4	0.65	0.76

Table 11: Three example rules with their heuristic and lifted heuristic values.

Table 11 shows three rules with different heuristic and lifted heuristic values. If we set the rule value of those rules to their respective lifted heuristic value, the third rule get chosen by the subroutine RefineRule as its lifted heuristic value is the best. If we set the rule value to the normal heuristic value instead, the second rule gets chosen. Of course, it might also be the case that a single-label head rule achieves the highest normal heuristic value. While still introducing a bias towards longer heads, rating rules with their normal heuristic value leads to the algorithm being quite difficult to understand. Furthermore, rating rules with their lifted heuristic values is more transparent and more in line with our intention. That is why in the remainder of this work, we always rate rules with their lifted heuristic value, unless stated otherwise.

Algorithm 22 Algorithm Evaluate, which evaluates a rule and calculates its lifted heuristic value as well as sets the rule value accordingly.

Require: relaxation lift function ρ, rule r, whether to rate rules by their lifted or normal heuristic value

1: **procedure** Evaluate
2: $h_{unlifted} = \text{EvaluateRule}(T, T_{current}, r, \delta)$ ▷ get (unlifted) heuristic value
3: $\vec{h} = h_{unlifted} \cdot \rho(|r.head|)$ ▷ apply relaxation lift
4: **if** use lifted heuristic value as rule value **then**
5: $r.h = \vec{h}$
6: **else**
7: $r.h = h_{unlifted}$
8: **end if**
9: **return** $\vec{h}$
10: **end procedure**

Algorithm 23 shows how we decide whether to evaluate more heads or stop the search. The check is based on calculating an upper boundary for the highest yet achievable lifted heuristic value. By constructing the best head of a certain length the way we do, we can be sure that the normal heuristic value cannot increase by adding more heads. As a result, we can use the heuristic value of the currently regarded head as an upper boundary for the heuristic value for the remaining head sizes. In order to calculate an upper bound for the lifted heuristic value, we additionally need an upper bound for the relaxation lift. This upper boundary can simply be attained by calculating the highest lift value of all head sizes to come, which is done by the subroutine GetMaximunLift. Thus, we can calculate the upper boundary for the lifted heuristic value (cf. Algorithm 23, line 3). If the upper boundary is smaller than the so far seen highest lifted heuristic value, we can be sure that we cannot achieve a higher lifted heuristic value for the remaining possible head sizes. Hence, we can safely prune at this point. Note that the upper boundary must be strictly smaller (cf. Algorithm 23, line 4). Otherwise, we could still attain the best lifted head as we prefer longer heads if the lifted heuristic value is equal.

Algorithm 23 Algorithm PRUNABLEWITHDECOMPOSABILITY, which determines whether or not the label space can be pruned for a certain relaxation lift function.

```
1: procedure PRUNABLEWITHDECOMPOSABILITY(Rule r_best, Rule r)
2:     ρ_max = GETMAXIMUMLIFT(|r.head|)   ▷ get the maximum remaining lift (of succeeding head sizes)
3:     h⃗_max = r.h_unlifted · ρ_max                ▷ calculate maximal achievable lifted heuristic value
4:     if h⃗_max < r_best.h⃗ then          ▷ if maximal achievable lifted heuristic value worse than best value
5:         return true
6:     else
7:         return false
8:     end if
9: end procedure
```

The subroutine GETMAXIMUMLIFT, which is depicted in Algorithm 24, is used in order to determine the highest lift of the remaining head sizes. Naturally, it requires the current head size as an input. In the case that the peak relaxation lift function is utilized, we can return the lift function value at the peak label if the current head size is smaller than the peak label. Otherwise, we search the highest lift for all longer head sizes (cf. Algorithm 24, line 6). Finally, we return the so found maximum lift value. As this calculation is performed very often, the lift function values for all head lengths are calculated in advance and then retrieved from a hash map when needed.

Algorithm 24 Algorithm GETMAXIMUMLIFT, which determines the maximum relaxation lift value of the remaining head sizes.

Require: the current head size s, the relaxation lift function ρ

```
1: procedure GETMAXIMUMLIFT
2:     if using the peak lift function and s smaller than the peak label l then
3:         return ρ(l)
4:     end if
5:     ρ_max = −1
6:     for i = s + 1, ..., n do
7:         ρ_current = ρ(i)
8:         if ρ_current > ρ_max then
9:             ρ_max = ρ_current
10:        end if
11:    end for
12:    return ρ_max
13: end procedure
```

6.2.2 Example

An example may illustrate the operation of the algorithm. In order to demonstrate the operation, we apply the algorithm on the example depicted in Table 10. The data set the example was taken from contains seven labels. However, we only regard five labels in our example. Furthermore, we are lifting the label space with the peak relaxation lift function with peak label three and a relaxation lift of 1.13 at the peak (cf. Table 12).

n	1	2	3	4	5	6	7
Lift	1	1.09	1.13	1.11	1.09	1.07	1

Table 12: Values of the peak relaxation lift function with a peak label of three, a lift of 1.13 at the peak and a curvature of two for a data set with seven labels.

The algorithm starts with constructing the best head of length one, which is $\{y_1\}$ with $h = 0.390$. For this head, we can now calculate the lifted heuristic value: $\vec{h} = 0.390 \cdot 1 = 0.390$. As we know that the heuristic value cannot get better than this value and that the highest remaining lift value is 1.13, we can calculate an upper bound for the lifted heuristic value:

$$\vec{h}_{upperBound} = 0.390 \cdot 1.13 = 0.441$$

Due to the upper bound on the lifted heuristic value being greater than the best lifted heuristic, we cannot prune yet as the lifted heuristic value may still increase by adding more labels. Thus, the algorithm continues with constructing the best head of length two $\{y_1, y_2\}$ with a heuristic value of 0.380 and a resulting lifted heuristic value of $\vec{h} = 0.380 \cdot 1.09 = 0.414$. As a result of the increased lifted heuristic value, the best performing head is updated. Again, we know that the heuristic value cannot be greater than 0.380 by adding more labels and that the highest remaining relaxation lift value is 1.13. Therefore, we can calculate the upper bound as follows:

$$\vec{h}_{upperBound} = 0.380 \cdot 1.13 = 0.42941$$

As the upper boundary on the lifted heuristic value is still higher than the best seen lifted heuristic value so far, we cannot terminate the algorithm yet if we want to guarantee finding the best performing head. Hence, the algorithm constructs the head of length three with the highest normal heuristic value $\{y_1, y_2, y_3\}$ with a heuristic value of 0.370 and a resulting lifted heuristic value of $\vec{h} = 0.370 \cdot 1.13 = 0.418$ as we are at the peak label. Again, the best performing head is updated. From now on, the relaxation lift starts to decrease. Therefore, the highest remaining relaxation lift value is 1.11 and for the upper bound follows:

$$\vec{h}_{upperBound} = 0.370 \cdot 1.11 = 0.4107$$

As the upper bound on the lifted heuristic value is strictly smaller than the one of the best performing head so far, we can be sure that the lifted heuristic value cannot increase anymore. As a result, we can terminate the algorithm and do not have to check the remaining heads. If we had used the KLN relaxation lift function with the same lift at head length three, the highest remaining lift value would be 1.19. Thus, for the upper bound would follow:

$$\vec{h}_{upperBound} = 0.370 \cdot 1.19 = 0.4403$$

Due to the upper bound being higher still, we would not be able to prune with another relaxation lift function. This is the reason why the peak relaxation function is more efficient. Note that the constructed heads do not actually have to be evaluated on the training set as the heuristic values can be calculated from the heuristic values of the single-label heads.

6.3 Anti-Monotonicity

Exploiting anti-monotonicity in pruning a lifted label search space is based on the same basic check whether or not the lifted heuristic value may still increase. Algorithm 25 illustrates the adjusted subroutine PRUNEDSEARCH. The subroutine now searches the best performing head according to the lifted heuristic value (cf. Algorithm 25, lines 13 and 20-22). Otherwise, the algorithm stays the same. Again, we introduce a lower boundary for the quality of the rule by requiring a rule to have at least as many true positives as false positives. The search for the best head is continued if either the lifted heuristic value increased or we cannot prune the remaining path in the label space yet, which is determined by the PRUNABLEWITHANTIMONOTONICITY subroutine (cf. Algorithm 25, line 20).

Algorithm 25 Algorithm AntiMonotonous, which performs a pruned search through the lifted label space according to the properties of anti-monotonicity.

Require: Original training set T, current training set $T_{current}$, current rule r,
targets G, evaluation function δ, already considered heads H,
whether to use rule-dependent or rule-independent evaluation strategy, the averaging strategy to use

1: **procedure** AntiMonotonous
2: $r_{best} = r$
3: **for each** label λ_i not already contained in $r.body$ **do** ▷ for each label, all targets are considered
4: $r_{current} = r$ ▷ currently considered label attribute
5: $r_{current}.\vec{h} = -\infty$
6: **for each** target $t \in G$ **do**
7: $y_i = t$
8: **if** label attribute y_i is not already in $r.head$ **then**
9: $r_{refined} = r$
10: $r_{refined}.head = r_{refined}.head \cup y_i$ ▷ add label attribute to head
11: **if** no head in H is a subset of $r_{refined}.head$ **then** ▷ prunes the evaluation of label combinations
12: $r_{refined}.\vec{h} = \text{Evaluate}(T, T_{current}, r_{refined}, \delta)$
13: **if** $r_{refined}.\vec{h} > r_{current}.\vec{h}$ **then** ▷ determines the best prediction for each label
14: $r_{current} = r_{refined}$
15: **end if**
16: **end if**
17: **end if**
18: **end for**
19: **if** $r_{current}.\vec{h} \neq -\infty$ **then** ▷ if label combination has not been pruned
20: **if** $r_{current}.\vec{h} \geq r_{best}.\vec{h}$ **or** !PrunableWithAntiMonotonicity$(r_{best}, r_{current})$ **then**
21: $r_{rec} = \text{AntiMonotonous}(r_{current}, G, T, \delta, H)$ ▷ performance did or can increase further
22: **if** $r_{rec}.\vec{h} > r_{best}.\vec{h}$ **or** $(r_{rec}.\vec{h} = r_{best}.\vec{h}$ **and** $|r_{rec}.head| > |r_{best}.head|)$ **then**
23: $r_{best} = r_{rec}$ ▷ heads with more label attributes are preferred
24: **end if**
25: **end if**
26: $H = H \cup r_{current}.head$
27: **end if**
28: **end for**
29: **return** r_{best} ▷ return best rule if TP $\geq$ FP
30: **end procedure**

Algorithm 26 shows how we decide whether or not the remaining path in the label space is prunable if using an anti-monotonous evaluation function. The subroutine is dependent upon a parameter d, which trades off efficiency and guaranteeing finding the best lifted head. If $d = -1$, finding the best head is guaranteed, but more heads might need to be evaluated. Otherwise, for calculating the upper boundary of the lifted heuristic value, the next d lift values are checked. This is in contrast to our previous approach for decomposability, which utilizes the highest possible remaining lift value. For instance, if parameter $d = 2$, the algorithm checks whether or not a higher lifted heuristic value can be achieved by the next two head lengths. A higher value of d induces less pruning and therefore a lower efficiency but also increases the chances of finding the best possible head in accordance to the lifted heuristic value. Likewise, a lower value induces more pruning and therefore a higher efficiency but also increases the chances of missing the best possible head. Hence, we let the user decide and trade off efficiency and increasing the chance of missing the best performing head. The calculation of the highest remaining lift value is again performed by the subroutine GetMaximumLift (cf. Algorithm 24). For $d \neq -1$ we utilize the subroutine GetMaximumLookaheadLift. Again, we calculate the (upper) boundary for the lifted heuristic value (cf. Algorithm 26, line 7) and check whether or not the lifted heuristic value could still increase, depending on parameter d (cf. Algorithm 26, line 8). For pruning with anti-monotonicity

an additional condition is required. We may only prune if the heuristic value has previously decreased on the path in the label search space. If that is the case, we can be sure that the normal heuristic value does not reach the best performance anymore due to definition of anti-monotonicity.

Algorithm 26 Algorithm PRUNABLEWITHANTIMONOTONICITY, which determines whether or not the label space can be pruned for a certain relaxation lift function and parameter d.

Require: the best rule r_{best}, rule r, the relaxation lift function ρ, parameter d

```
1:  procedure PrunableWithAntiMonotonicity
2:      if (d = −1) then                                   ▷ guarantee finding of best head
3:          ρ_max = GetMaximumLift(|r.head|)               ▷ get the maximum remaining lift
4:      else                                               ▷ get the highest lift out of the next d values
5:          ρ_max = GetMaximumLookaheadLift(|r.head|, d)
6:      end if
7:      h⃗_max = r.h_unlifted · ρ_max                       ▷ calculate maximal achievable lifted heuristic value
8:      if h⃗_max < r_best.h⃗ and h decreased on path then
9:          return true                                    ▷ maximal achievable lifted heuristic value worse than best value
10:     else
11:         return false
12:     end if
13: end procedure
```

Algorithm 27 illustrates the operation of the subroutine GETMAXIMUMLOOKAHEADLIFT, which determines the highest lift value of the next d head lengths. If using the peak relaxation lift function and the peak label is within the checked head lengths, we can just return the lift function value at the peak label (cf. Algorithm 27, line 2). Otherwise, we iterate over the next d head lengths and keep track of the highest seen lift value (cf. Algorithm 27, line 6). Finally, the highest lift within the specified interval is returned.

Algorithm 27 Algorithm GETMAXIMUMLOOKAHEADLIFT, which determines the maximum relaxation lift value of the next d head sizes.

Require: current head size s, relaxation lift function ρ, parameter d

```
1:  procedure GetMaximumLookaheadLift
2:      if using the peak lift function and s < the peak label l ≤ s + d then
3:          return ρ(l)
4:      end if
5:      ρ_max = −1
6:      for i = s + 1, ..., s + d do
7:          ρ_current = ρ(i)
8:          if ρ_current > ρ_max then
9:              ρ_max = ρ_current
10:         end if
11:     end for
12:     return ρ_max
13: end procedure
```

Finally, after introducing both adjusted algorithms for finding the head with the highest lifted heuristic value, we can adjust the subroutine FINDBESTHEAD. Instead of using the subroutines proposed by Rapp [2016], we utilize the adjusted algorithms that are capable of handling the lifted heuristic value. In the implementation, which is done using the SECO-Framework [Janssen and Fürnkranz, 2010; Janssen and Zopf, 2012], the user can specify whether to use the relaxed pruning approach or not. The adjusted subroutine FINDBESTHEAD is illustrated in Algorithm 28.

Algorithm 28 Adjusted algorithm FINDBESTHEAD for finding the best (lifted) multilabel head for a given body of a rule.

Require: Original training set T, current training set $T_{current}$,
current rule r, targets G, evaluation function δ, whether to use relaxed pruning,
whether to use rule-dependent or rule-independent evaluation strategy, the averaging strategy to use

```
1: procedure FINDBESTHEAD
2:     r.head = ∅
3:     if use relaxed pruning then
4:         if δ is decomposable using the given evaluation and averaging strategy then
5:             return DECOMPOSABLE(T, T_current, r, G, δ)
6:         else
7:             return ANTIMONOTONOUS(T, T_current, r, G, δ, ∅)
8:         end if
9:     else
10:        if δ is decomposable using the given evaluation and averaging strategy then
11:            return DECOMPOSITE(T, T_current, r, G, δ)
12:        else
13:            return PRUNEDSEARCH(T, T_current, r, G, δ, ∅)
14:        end if
15:    end if
16: end procedure
```

green = 0, yellow = 0, red = 0 ← area ≤ 9462.0	(13, 6)
yellow = 0, red = 0, white = 1 ← area ≤ 9462.0, circles ≤ 1.0	(14, 4)
white = 1 ← area ≤ 9462.0, circles ≤ 1.0, colours ≥ 2.0	(4, 1)
red = 1, white = 1 ← colours ≥ 2.5	(166, 38.0)
red = 1, white = 1 ← colours ≥ 2.5, crescent = 0	(164, 36.0)
red = 1, white = 1 ← colours ≥ 2.5, crescent = 0, bars ≤ 3.0	(163, 35)
red = 1 ← colours ≥ 2.5, crescent = 0, bars ≤ 3.0, area ≤ 2641.5	(85, 9)

Figure 8: Negative examples for the rule refinement process without fixing the head. True positives and false positives are depicted by (TP, FP).

6.4 Fixed Head During Rule Refinement

The intention of the relaxed pruning was to learn more multilabel head rules in the hope of also learning more label dependencies and a more expressive model as a result. An observation that often occurred is depicted in Figure 8. During the refinement process, in which we would like to specialize the found rule, the learned multilabel head rule is lost. By specializing the rule too much, the (lifted) heuristic value increases at the expense of coverage. The first refinement process in Figure 8 shows two multilabel heads being learned. During the third refinement step, however, the algorithm now considers a single-label head to be the best performing head. Therefore, coverage decreases and the multilabel head is not refined, which might have yielded a better performance than that of the second multilabel head.

The second refinement process in Figure 8 shows a similar observation. We refine a multilabel head three times before the algorithm decides to drop it in favor of a single-label head. Of course, in terms of performance the single-label head might be slightly better, but we lose a label co-occurrence and coverage of the rule. Again, it might have been better to refine the multilabel head rule more instead of settling for a single-label head.

black = 0 ← stripes ≤ 1.0	(24.0 8.0)
black = 0, blue = 1, orange = 0 ← stripes ≤ 1.0, colours ≥ 3.0	(58, 26)
black = 0, blue = 1, orange = 1 ← stripes ≤ 1.0, colours ≥ 3.0, bars ≤ 0.0	(50, 15)
black = 0, blue = 1 ← stripes ≤ 1.0, colours ≥ 3.0, bars ≤ 0.0, orange = 0	(48, 13)

Figure 9: Positive example for the rule refinement process without fixing the head. True positives and false positives are depicted by (TP, FP).

It should be mentioned that this observation, especially the second refinement process in Figure 8, can also be countered by increasing the relaxation lift. If we increase the lift enough, the refined rule does also learn the combination of both labels. While this approach might be a good idea for refining this rule, the quality of the rules in the other iterations of the rule learning process might suffer from this increased lift. Thus, the refinement process may benefit, but we are likely to learn worse rules in the other iterations of the algorithm.

In order to tackle this problem, we fix the head during the rule refinement. Therefore, we can focus on refining the found multilabel head rule and do not lose coverage. A nice side effect of this change is a faster run time of the algorithm. As a result of fixing the head, we do not need to perform the subroutine FindBestHead for all possible refinements. Contrary to evaluating at least all single-label heads, we only need to evaluate exactly one head for each refined body.

Of course, the change is not always beneficial. In Figure 9 we can observe that a better rule could be found by searching for the best head for each refined body. After the first refinement, the number of labels in the head increases. Hence, so does coverage. Another interesting observation for this example is the placement of the label *orange*. In the first multilabel head, it is predicted to be irrelevant. After adding another condition to the body, the best head predicts the same labels, but predicts *orange* to be relevant now. Finally, the prediction of label *orange* reverts again and is additionally moved to the body of the rule instead of the head. In this case, the final head in combination with the refined body is preferable to the initial found rule.

Although we can see that fixing the head is not always the better choice, we use the fixed heads for the evaluation in Section 9. The reason being that the case shown in Figure 9 is rare and not typical for the rule refinement process. Often, we can observe that the number of labels in the head does not increase by refining the body of the rule. Therefore, the chance of missing out on good rules is low. In contrast to that, we may learn more expressive rules and may refine the multilabel rules further. While we think that fixing the head is an improvement, we allow the user to toggle this feature and therefore be able to use the original refinement process.

6.5 Discussion

In this chapter, we have given two examples of why anti-monotonicity and decomposability do not hold for the lifted heuristic value. Thus, we proposed a different way of pruning the lifted label search space, which relies on anti-montonicity and decomposability of the heuristic value and the calculation of a upper bound for the lifted heuristic value. Additionally, the reason for rating rules with their lifted heuristic value has been explained. Finally, fixing the head was presented as a way better way of refining and inducing multilabel rules.

7 Sensitivity Analysis

The objective of this sensitivity analysis is to characterize the influence of the relaxation lift parameters and thus further the understanding of the concept of relaxed pruning. Naturally, different lifting settings have to be evaluated and compared. Therefore we take a look at different settings for the data set FLAGS. The reason for choosing FLAGS lies in its practicality. It is easy to understand and can be executed quickly, simplifying the conduct of the sensitivity analysis. As we have already shown that for our purpose the KLN and the root boost function are very similar, we restrict ourselves to utilizing the KLN lift function in this analysis.

7.1 Setup

The parameter setting is, for the sake of comparison, the same as for the evaluation in Section 9. For the KLN relaxation lift function, we start off with a lift of 1.01 and increase it by steps of 0.02 until a lift of 1.29. For the peak lift function, we start off with the peak at label two and proceed identical as for the KLN lift function. After reaching the maximum lift of 1.29, the peak label is increased by one. The process is repeated until label five is reached. The heuristic that was used for learning the models is the micro-averaged f-measure as we try to focus on the relevant labels. Furthermore, we set the curvature of the peak lift function to a value of two. It is important to differentiate between using only positive heads and additionally allowing negative heads to be induced. As a result, we perform the analysis for both settings. By allowing negative heads, the chance of learning multilabel head rules increases. Especially if the data set has a low label cardinality.

7.2 Negative and Positive Head Rules

This section covers the sensitivity analysis for the approaches that aim at explicitly predicting the relevancy as well as irrelevancy of labels using rules. We start with the analysis of the KLN relaxation lift function and proceed to analyze the effects of the peak relaxation lift function. At last, the fit of the learned models on the training set is regarded.

7.2.1 KLN Relaxation Lift Function

In this section, the results attained from utilizing the KLN relaxation lift function are presented. We first take a look at the characteristics of the learned models, followed by the heuristic values.

Model Characteristics

At first, we regard the characteristics of the learned models when using the KLN relaxation lift function. The most important and maybe obvious effect of the relaxation lift can be observed in Figure 10 (1). We can see that utilizing the relaxation lift does indeed lead to more multilabel head rules being learned, as was intended. Additionally, a higher lift results in longer heads being learned, as Figure 10 (2) illustrates. This is the result of the KLN lift function being monotonously increasing, i.e. the lift does steadily increase for a higher number of labels. By increasing the lift we strengthen the bias towards longer heads. Due to more labels being covered by a single (multilabel) rule, fewer rules are needed for covering the training data. For instance, a rule with two labels in the head covers twice as many entries in the training set label matrix as the same rule with a single-label head. This is a huge factor in run time as the rule learning process does not need to be executed as often. In fact, we will see that despite the need to evaluate more heads, the relaxed approach is often faster than its unrelaxed counterpart. Furthermore, in comparison to only utilizing single-label head rules for expressing a concept, more compact models can be learned. This is especially the case for subsequent rules without any conditions. As they are always applicable, the heads of these rules can easily be combined without information loss if none of the rules are stopping rules. Hence, expressing the same information with fewer rules. A downside of this behaviour is illustrated in Section 9.7.2. Figure 10 (3)

shows the relationship between the relaxation lift and the number of learned rules. Contrary to the number of rules, the number of stopping rules does not decrease but stays mostly constant (cf. Figure 10 (4)).

Similarly, the average number of conditions per rule declines (cf. Figure 11 (1)). A rule with more labels in the head is more difficult to refine as conditions need to be found that improve the performance of the rule with all three labels in the head (for a fixed head). Likewise, fewer label conditions are learned. Exposing these label conditions was one of the main intentions of exploring the concept of relaxation lift. However, not only are fewer label conditions learned, but also a smaller percentage of label conditions among all conditions are learned (cf. Figure 11 (2)). Whether this is because of the specific data set or because of the algorithm remains unclear when only considering one data set. We will examine this finding further in Section 9. Nevertheless, let us cover possible reasons. A simple reason could be that perhaps there are just not sufficiently enough dependencies between a label and several others. Especially for the data set FLAGS, the absence of a color might not indicate the presence or absence of another color. Single-label heads express dependencies between two labels, e.g. $\text{red} = 1 \leftarrow \text{white} = 1$. By introducing the relaxation lift, we prefer rules with more labels in the head. As a result, a rule with a long head that is not dependent on any label conditions might decline less than one that is label dependent. For instance, let us consider the rules

$$\text{black} = 1, \text{red} = 1, \text{yellow} = 0 \leftarrow \text{blue} = 1 \tag{7}$$

$$\text{black} = 1, \text{red} = 1, \text{yellow} = 0 \leftarrow \text{population} \geq 80 \tag{8}$$

The first rule is then likely to be worse, as the dependency does not actually exist or is not that strong. Hence, the latter rule is the better rule with three labels in the head and would therefore be preferred by the algorithm. Albeit a fewer percentage of label conditions, the impact of the conditions might increase. If we indeed learn a label condition in a rule with several labels in the head, the label condition has impact on more labels than a label condition with a single-label head. Moreover, we need to consider the position in the decision list at which the rule with the label condition is learned. A rule in the later stages of the rule learning process covers much fewer examples than a rule in the early stages. Hence, the former is less useful in generalizing the training data.

Heuristic Evaluation

Secondly, we regard the quality of the learned models, i.e. how well they perform on the test data. Theoretically, allowing worse rules should decrease precision and increase recall (on the training set). Figure 12 (1) depicts the hamming loss and subset accuracy values for the learned models. Considering these heuristics, a lift between 7 and 15 percent performs best. For a lower lift, we seem to not lift enough in order to induce the good rules. For a bigger lift, we tolerate too much of a decline in the heuristic value.

Figure 12 (2) shows the micro-averaged values for precision, recall and f-measure. The range for which hamming loss and subset accuracy does perform well also reaches the highest f-measure values. After declining slightly for a small lift, the values pick up again and increase slightly. Especially recall is higher in the range from 7 to 15 percent lift, resulting in the higher f-measure. After this range, recall drops for several values and then reaches its maximum. However, due to the very high recall, precision drops at this point. Resulting in a worse f-measure value. Here, the values are far from balanced. Interestingly, the theoretic expectation does not hold for the micro-averaged values as both recall and precision increase (on the test data). A reason for this could be better generalization properties of the learned model by inducing rules with a slightly worse performance on the training set. However, it is difficult to judge whether said effects are a result of the relaxation lift or the introduced lower boundary of the quality of the rule.

The macro-averaged values for precision, recall and f-measure, as shown in Figure 12 (3), behave differently. Precision and recall and therefore also the f-measure (almost) constantly decline until a lift

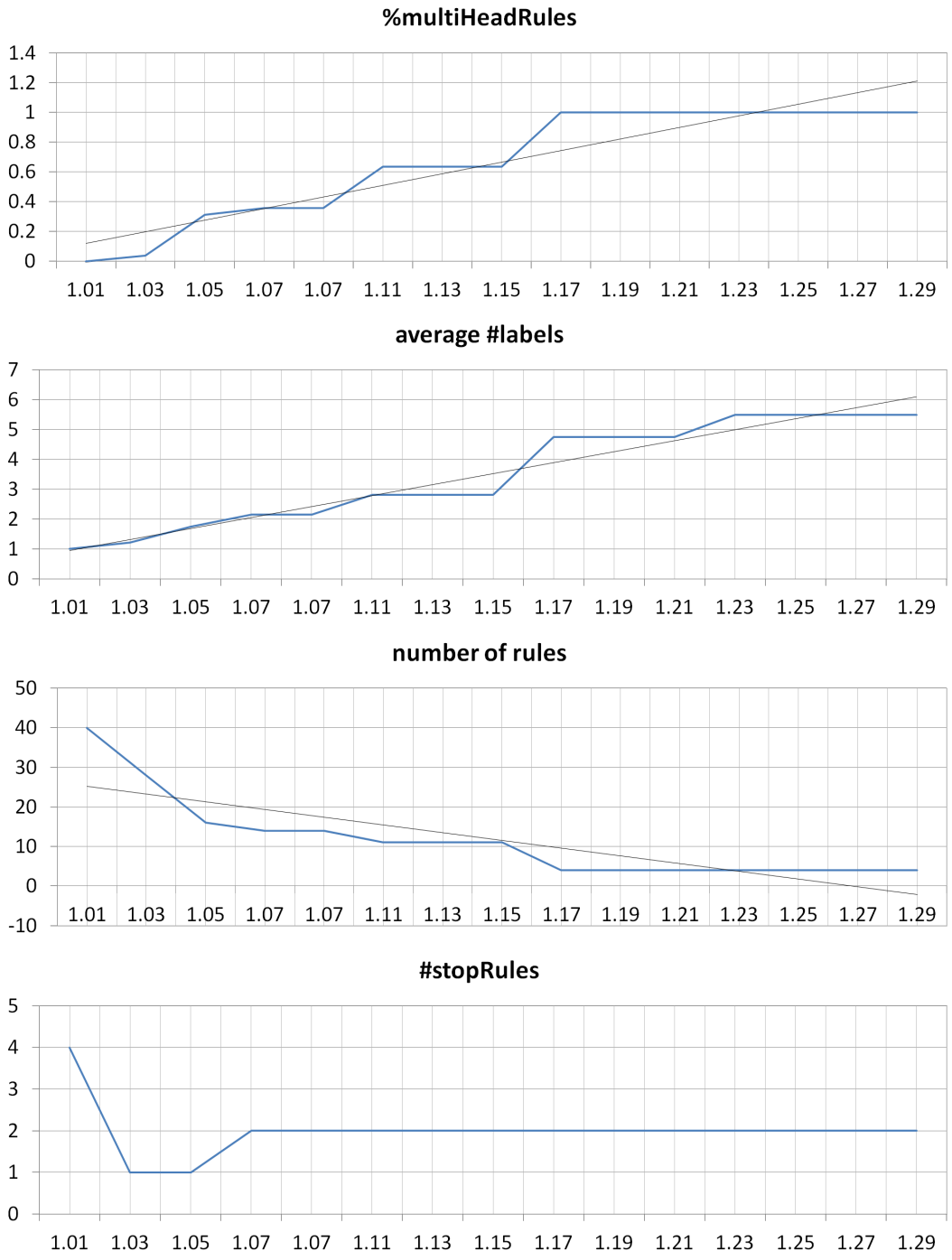

Figure 10: The (1) percentage of multi head rules among all rules, (2) average number of labels in the head, (3) number of rules and (4) number of stop rules for each model in comparison to the relaxation lift applied while using the KLN relaxation lift function as well as positive and negative heads.

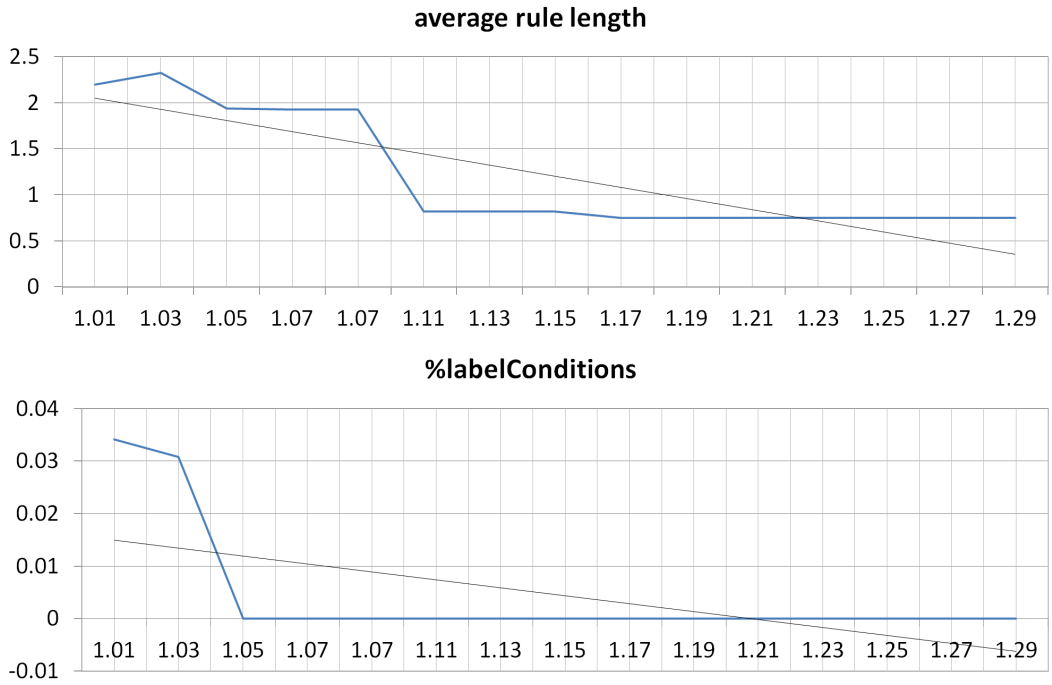

Figure 11: The (1) average rule length, i.e. the average number of conditions, and the (2) percentage of label conditions for each model in comparison to the relaxation lift applied while using the KLN relaxation lift function as well as positive and negative heads.

of 21 percent is achieved. From there on, we have a similar picture as for the micro-averaged values. The decline in the macro-averaged values may very well be as frequent labels are preferred over infrequent values the higher the lift or the fact that due to the condition of requiring at least as many true positives as false positives not all relevant labels can be learned. Hence, the infrequent labels dragging the heuristic value down in the case of the former reason.

7.2.2 Peak Relaxation Lift Function

In this section, the results attained from utilizing the peak relaxation lift function are presented. We again first take a look at the characteristics of the learned models, followed by the heuristic values. Furthermore, we compare and distinguish the results for different peak values.

Model Characteristics

The general behaviour regarding the percentage of learned multi head rules is identical to the previous analysis. What changes, however, is the number of induced rules. Whereas only four rules are being learned starting with a lift of 17 percent when utilizing the KLN lift function, more rules are generated when utilizing the peak function. Nevertheless, with a higher peak label the number of rules approaches the, it seems, minimum number of rules (for this data set). For peak label five, the same number of rules as for the KLN lift function is reached, but only for a lift of 29 percent. Moreover, the average rule length and the number of stop rules are alike as well. That the peak value does indeed have an effect on the length of the heads is depicted in Figure 13. Thus, the peak label sets a threshold on the maximum number of labels in the head. Note that in the beginning, for a rather small lift, the order is reversed, i.e. peak label two has longer heads than peak label five. The reason for this lies in the relaxation lift function. For the peak lift function with peak label five and lift five percent, the lift at label two is a lot smaller than for the function with peak label two and lift five percent. Hence, the induction of multilabel head rules is less likely.

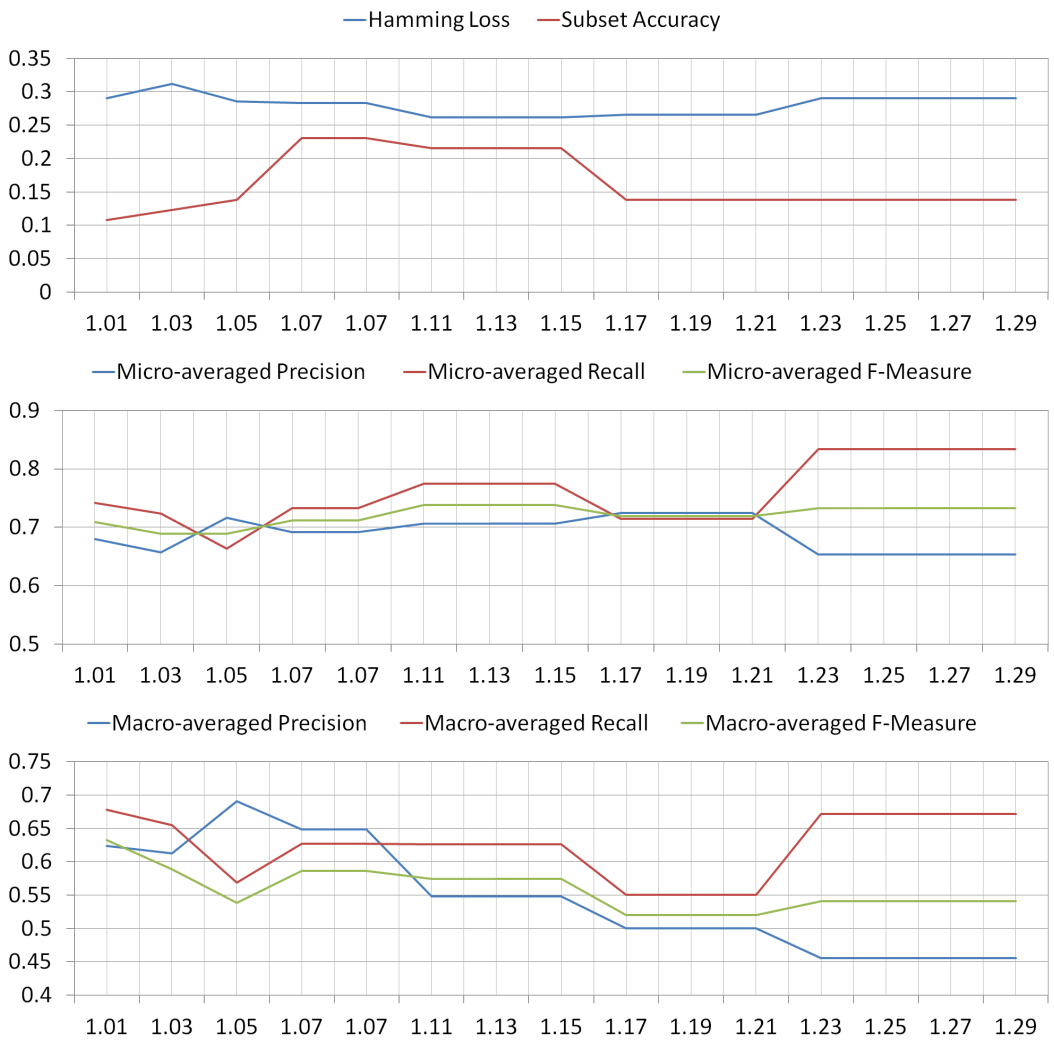

Figure 12: The values for (1) hamming loss and subset accuracy (2) micro-averaged precision, recall and f-measure as well as (3) macro-averaged precision, recall and f-measure for each model in comparison to the relaxation lift applied while using the KLN relaxation lift function as well as positive and negative heads.

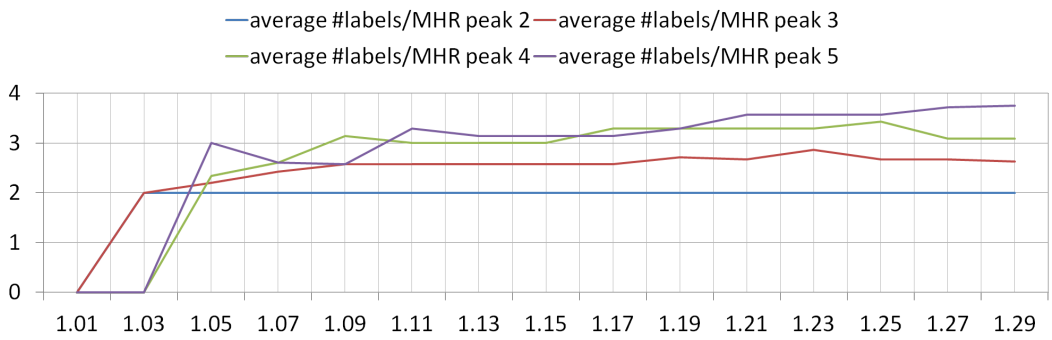

Figure 13: The average number of labels per multilabel head rule in the head for each peak label in comparison to the relaxation lift applied while using the peak relaxation lift function as well as positive and negative heads.

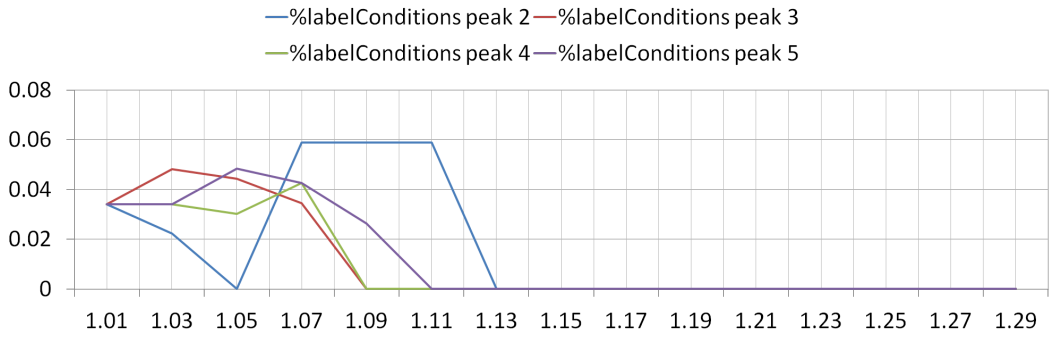

Figure 14: The percentage of label conditions for each peak label in comparison to the relaxation lift applied while using the peak relaxation lift function as well as positive and negative heads.

Interestingly, a slightly bigger percentage of label conditions seems to be induced for a small lift (cf. Figure 14). The most likely reason is that as a result of the lift, fewer rules and thus fewer conditions are learned. Hence, if the number of label conditions stays constant, the percentage of label conditions increases. Another possible reason is the increased coverage. Due to more labels being set by the first few rules, they can be utilized more effectively in the exploitation of label dependencies as more entries in the label matrix are set. For a higher lift, no label conditions are learned anymore. The reasons are similar to the reasons of why fewer conditions per rule are learned in general with a higher lift, which was explained in the previous section. As previously mentioned, rules with more labels in the head are not as easily refined as with more labels the chances of improving the performance declines.

Heuristic Evaluation

For hamming loss and subset accuracy, the peak relaxation lift function with peak label four seems to achieve the best results (cf. Figure 15 (1)). Although, for the other peak functions the results are equally good in comparison to the KLN relaxation lift function. The main difference between Figure 12 (1) and 15 is where the maximum performance is reached. For the peak lift function with peak label four, the arguably best performance is reached for a bigger lift. This is not the case because of less lift being applied but rather a difference in the expression of the lift. For the KLN lift function, the lift is set differently – fixed at label three the lift is steadily increased, resulting in the overall lift increase. For the peak lift function, the lift is set according to the peak label and interpolated in between the other labels.

Figure 15 (2) shows the micro-averaged evaluation results for the peak relaxation function with peak label two. It becomes apparent that the variance of the values is very constant. The same is true for the

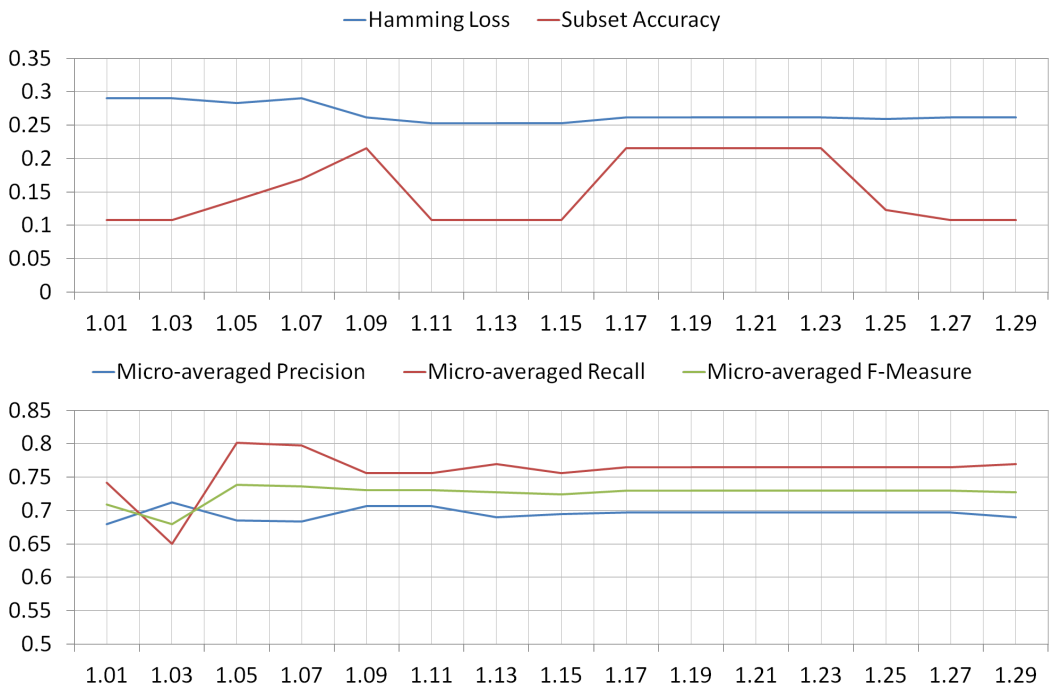

Figure 15: (1) The values for hamming loss and subset accuracy for peak label four and (2) the values for micro-averaged precision, recall and f-measure for peak label two in comparison to the relaxation lift applied while using the peak relaxation lift function as well as positive and negative heads.

hamming loss and subset accuracy value for this peak label. Due to the limit imposed on the number of labels in the head, a bigger lift does not make a difference anymore from some point on. If all or most rules are rules with two labels in the head already, an increased lift will not change the rule set. Hence, the very constant values. A similar observation emerges for the macro-averaged values for peak label two. For both averaging-strategies, however, recall is higher than precision.

The micro-averaged prevision, recall and f-measure values for the peak relaxation lift function with peak label three are rather constant again (cf. Figure 16 (1)). Starting with a lift of 21% things start to change. Recall increases and precision decreases. Nevertheless, the f-measure rises. Especially for a lift of 29%, where precision picks up again. For this peak label it might be interesting to investigate what happens beyond a lift of 29%. However, it is possible that shortly after the same situation as for peak label two arises, the values become constant. For the macro-averaged values a slightly different picture than for peak label two presents itself (cf. Figure 16 (2)). While for peak label two all values are rather constant, the function for peak label three varies more for the bigger lifts. Interestingly, precision and recall change roles for a lift of 21%. The same lift at which the micro-averaged values start to change. Before this lift value, precision dominates and is significantly higher than recall. Afterwards, recall does. In the end, however, the macro-averaged values drop slightly. A reason for the change starting with about a lift of 19 % might be that from there on, multilabel heads with three labels in the head get induced more often. This theory is supported by the average number of labels per head, as shown in Figure 17. We can see that from a lift of 19% on, more rules with three labels in the head are induced due to the average number of labels per head significantly rising above a value of two. In this case, the average number of labels is a better indicator than the average number of labels per multilabel head rule, as it incorporates the number of such rules.

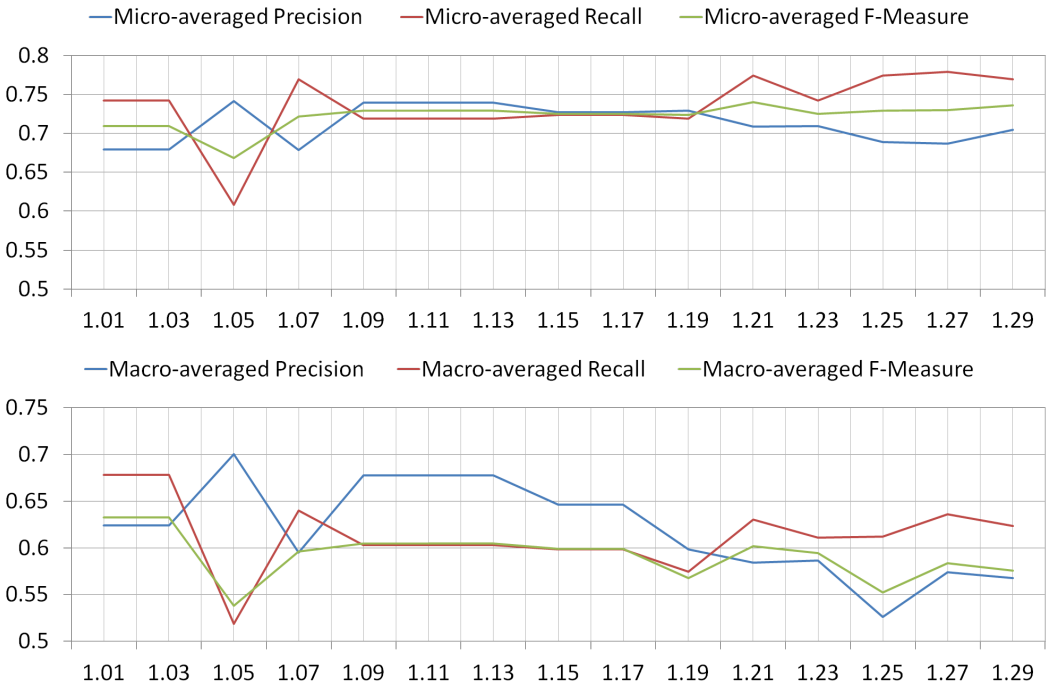

Figure 16: The values for (1) micro-averaged precision, recall and f-measure as well as (2) macro-averaged precision, recall and f-measure for peak label three in comparison to the relaxation lift applied while using the peak relaxation lift function as well as positive and negative heads.

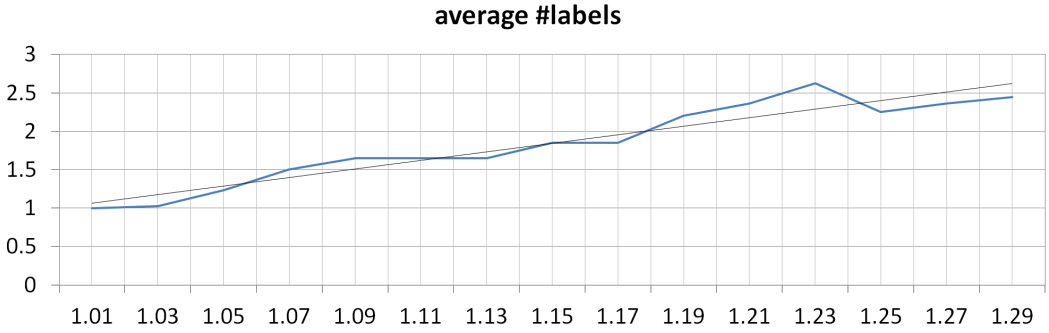

Figure 17: The average number of labels for peak label three in comparison to the relaxation lift applied while using the peak relaxation lift function as well as positive and negative heads.

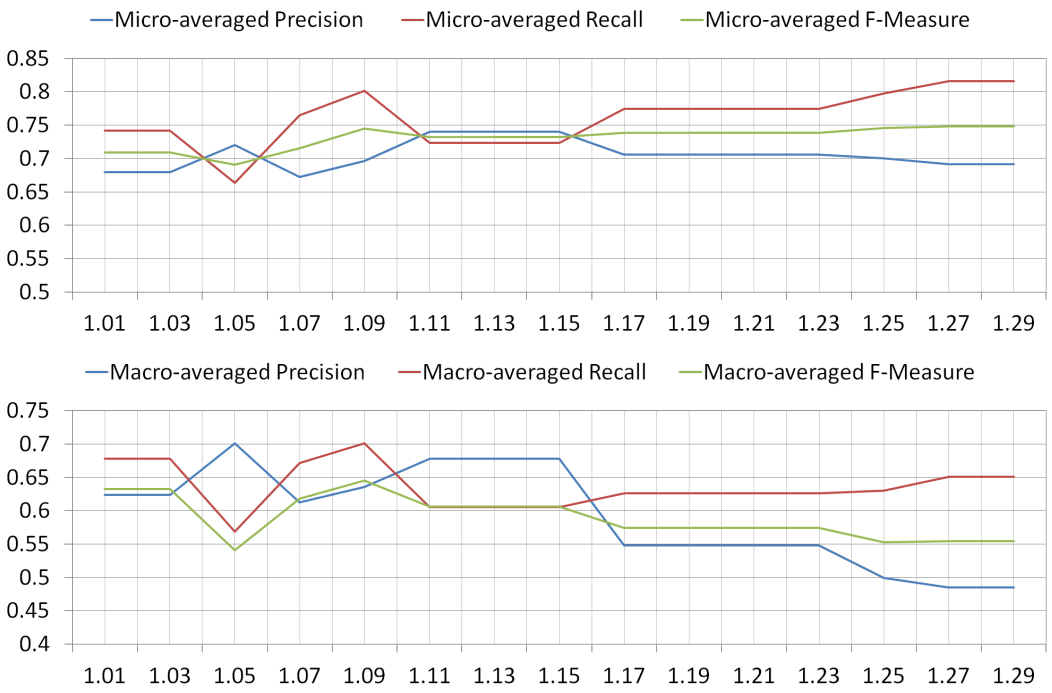

Figure 18: The values for (1) micro-averaged precision, recall and f-measure as well as (2) macro-averaged precision, recall and f-measure for peak label four in comparison to the relaxation lift applied while using the peak relaxation lift function as well as positive and negative heads.

Figure 18 (1) shows the micro-averaged heuristic values for the peak relaxation lift function with peak label four. Again, we see that for a bigger lift recall seems to generally increase and precision seems to generally decrease. What becomes apparent is that there always seems to be a trade off between the two. If one rises, the other suffers. In Figure 18 (2) we can see the macro-averaged values. It appears to be the case that lift increases the micro-averaged f-measure values at the cost of the macro-averaged values. Generally, the micro- and macro-averaged values change likewise, i.e. if micro-averaged precision drops, so does macro-averaged precision. Note that out of all tested peak labels, the function with peak label four reaches the highest f-measure values. For peak label five, the behaviour is rather similar and is therefore not covered in detail. The micro- and macro-averaged values can be regarded in Figure 19 (1) and Figure 19 (2), respectively. Note that this peak relaxation lift function reaches the second highest f-measure values.

7.2.3 Training Set Fit

Next, we take a look at the effects of the relaxation lift on the fit on the training set. This may be important in order to understand for which settings the approach overfits or generalizes the training data too much. Table 20 shows the performance on the training set for the peak relaxation lift function with peak label four. In general, we can observe that the fit seems to decrease with a higher relaxation lift. This is the case for hamming and subset accuracy, as well as the micro- and macro-averaged values. The micro- and macro-averaged values start to increase again at some point, but do not reach as good of a fit as previously. A reason for this behaviour might be that starting with a lift of 1.25, more rules with four or five labels in the head get learned (cf. Figure 13). Moreover, the behaviour of subset accuracy on the training data somewhat matches the behaviour of subset accuracy on the test set (cf. Figure 15). For a lower lift, the behaviour is reversed, i.e. a worse fit on the training set implies a better performance on the test set. Hence, due to an increased generalization, the predictive performance also increases. For the other peak labels, a similar behaviour can be observed. However, the higher the peak label, the more does the learned model generalize the training set. This can be observed in Figure 21, which shows

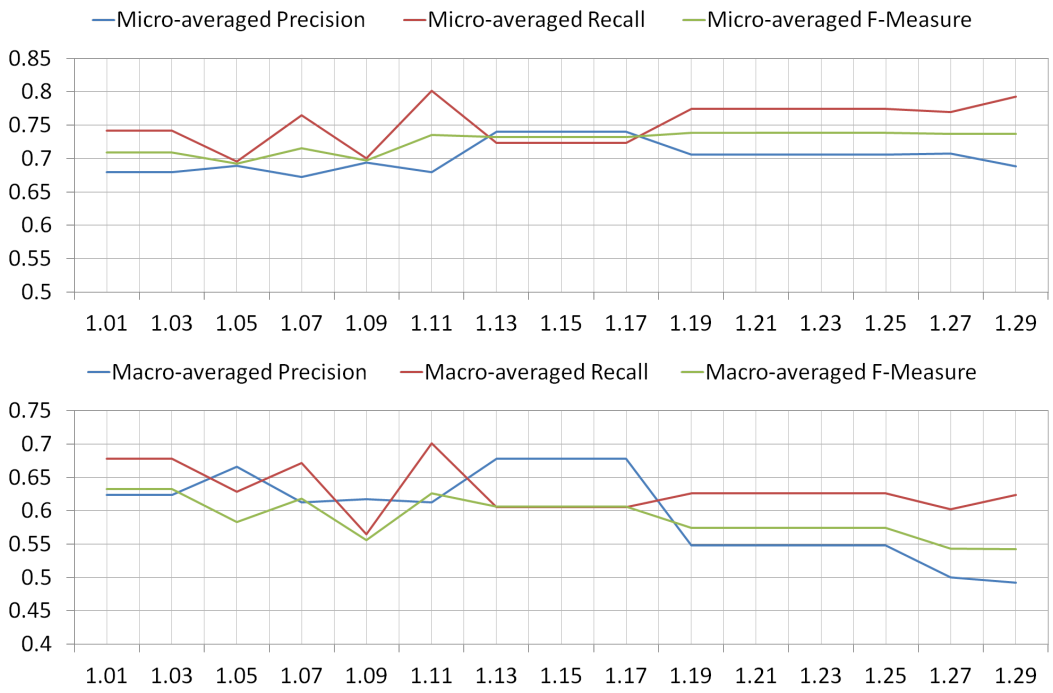

Figure 19: The values for (1) micro-averaged precision, recall and f-measure as well as (2) macro-averaged precision, recall and f-measure for peak label five in comparison to the relaxation lift applied while using the peak relaxation lift function as well as positive and negative heads.

the macro-averaged evaluation function values for peak label five and three. For peak label five, we can identify the point at which rules with three or more labels in the head start to get induced – a relaxation lift of 17 percent. The values for peak label three do not decline as much as the values for peak label five. Hence, we can conclude that the longer heads are learned, the more the training set is generalized. This might also be the result of the previously mentioned declining average number of conditions per rule. With fewer conditions per rule, the rule sets becomes more general.

7.2.4 Summary

For this data set and the utilization of negative heads, a higher number of peak labels seems to be preferable. In general, the higher the peak label, the higher the lift should be in order to achieve good results. The reason being the interpolation between the first and the peak label. Often, a high peak label is more robust to inducing bad rules as due to the interpolation, i.e. a lower lift for a lower label but maintaining a high lift for a high label, only the best rules with a high number of labels in the head get induced. Usually, the quality of a rule suffers the more labels are added to the head. By regarding this observation, the question arises why the KLN relaxation lift function does not deliver the best results. The reason being that, despite the additional lift declining for each additional label, there is no maximum lift. Hence, we do not risk inducing rules with simply too many labels in the head. Of course, this is dependent on the label cardinality of the data set at hand. However, for FLAGS we have a medium label cardinality of 3.392. For the peak relaxation lift function and peak label three or higher, a lift of at least 10% should be applied.

7.3 Positive Head Rules

This section covers the sensitivity analysis for the approaches that can only explicitly predict the relevancy of labels. We start with the analysis of the KLN relaxation lift function and proceed to analyze the effects of the peak relaxation lift function. At last, the fit of the learned models on the training set is regarded.

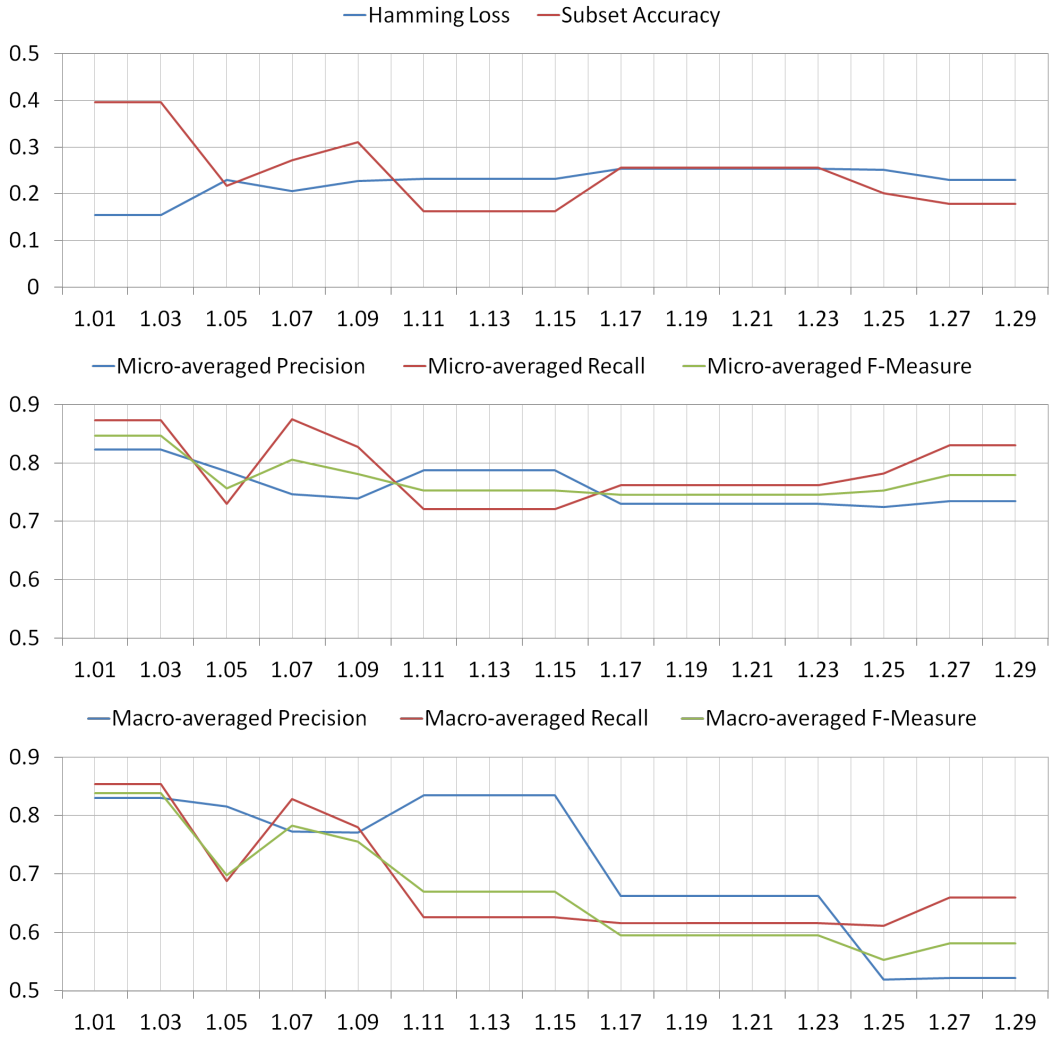

Figure 20: The (1) hamming loss and subset accuracy, (2) micro-averaged precision, recall and f-measure values and (3) macro-averaged precision, recall and f-measure values *on the training set* in comparison to the relaxation lift applied while using the peak relaxation lift function with peak label five and learning positive as well as negative heads.

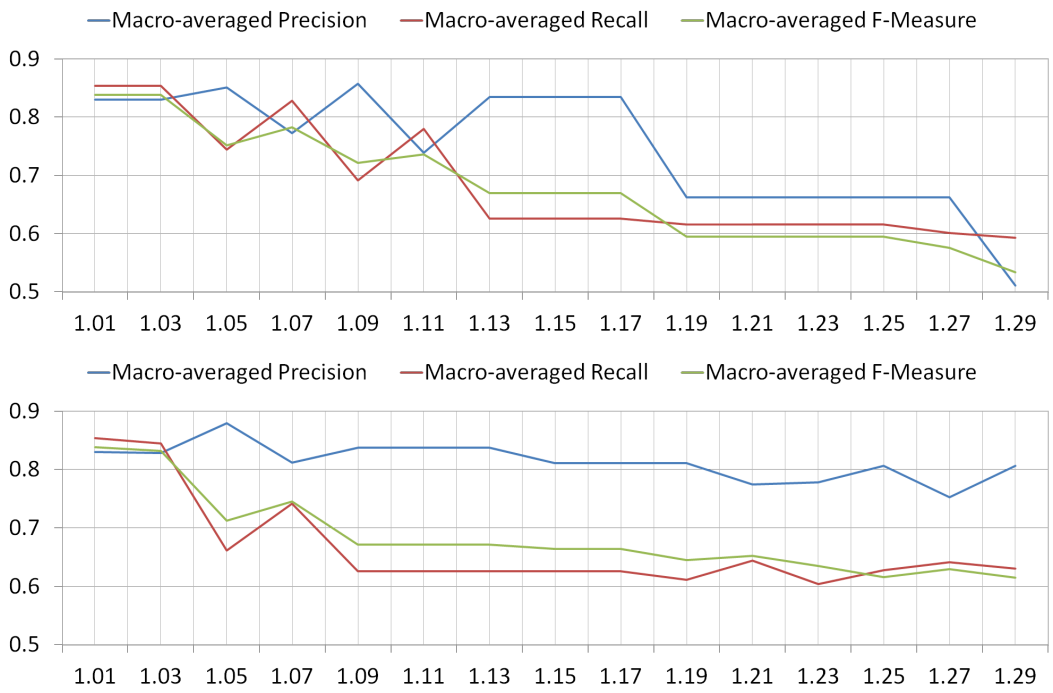

Figure 21: Macro-averaged precision, recall and f-measure values *on the training set* in comparison to the relaxation lift applied while using the peak relaxation lift function with peak label (1) five and (2) three while learning positive as well as negative heads.

7.3.1 KLN Relaxation Lift Function

In this section, the model characteristics and the values of the heuristic functions when using the KLN relaxation lift function and exclusively learning positive heads are interpreted.

Model Characteristics

Applying the pruning relaxation on the data set FLAGS and only utilizing positive heads does not induce as many multilabel head rules as when also using negative heads, as can be observed in Figure 22 (1). Figure 22 (2) further shows that the average number of labels per multilabel head rule does not change throughout the lifts 1.05 to 1.23. Only for a very high lift the algorithm starts to induce heads of length three or higher. The consistency of the head length may very well be a reason for the relatively high (constant) percentage of label conditions. This stays in contrast to the heavy drop in the percentage of label conditions when also inducing negative heads.

Heuristic Evaluation

In comparison to the analysis for negative heads, recall is quite high and stays a lot higher than precision (cf. Figure 23) because no negative heads are predicted and the algorithm thus focuses on the relevant labels. This can be observed for later values as well. For the data set FLAGS, the heuristic values when utilizing only positive heads are slightly better than for the setting that also uses negative heads. Increasing the lift results in a slight increase in micro-averaged and macro-averaged values, before dropping rather heavily for a lift of 1.27 due to an especially high recall and therefore lower precision (cf. Figure 23 (2) and (3)).

7.3.2 Peak Relaxation Lift Function

In this section, the model characteristics and the values of the heuristic functions when using the peak relaxation lift function and exclusively learning positive heads are interpreted.

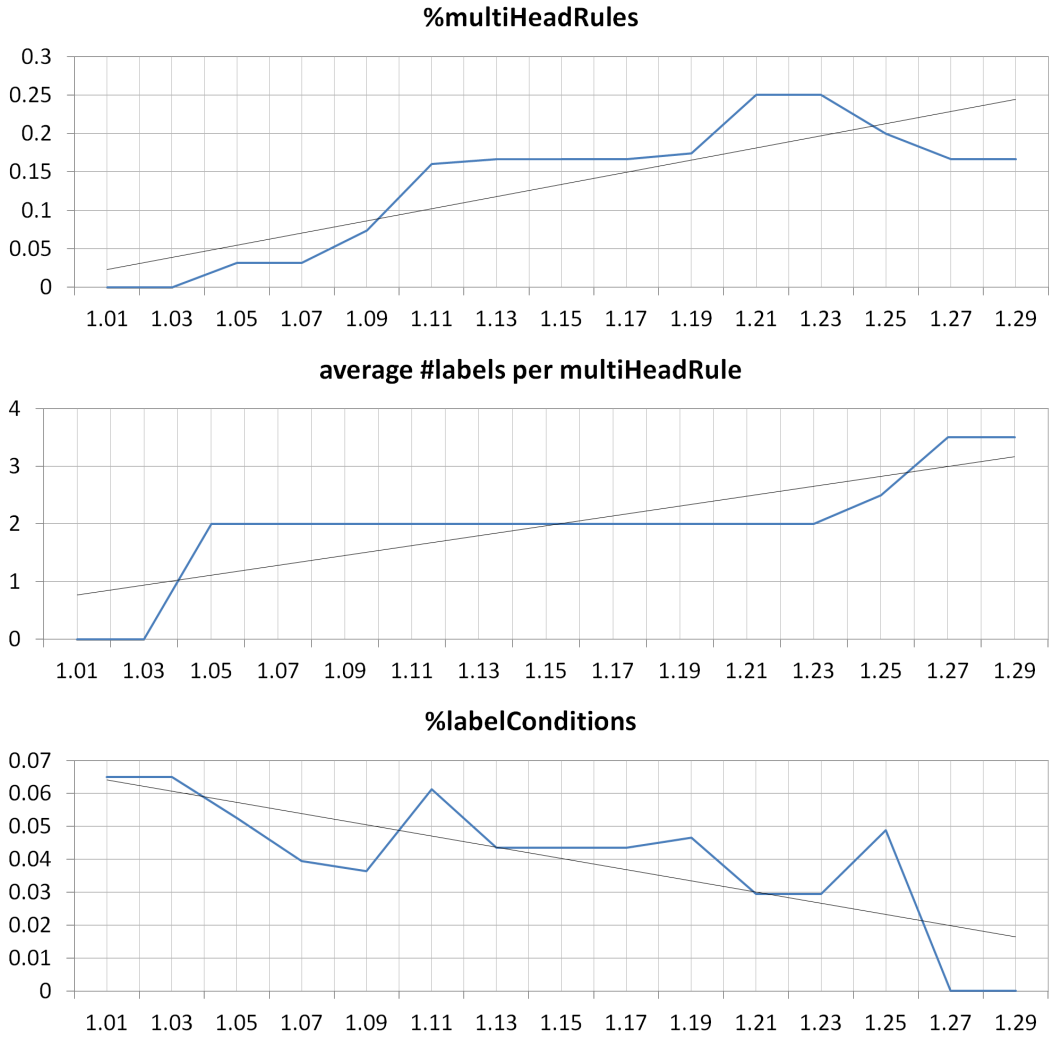

Figure 22: The (1) percentage of multi head rules, (2) average labels per multi head rule and (3) percentage of label conditions in comparison to the relaxation lift applied while using the KLN relaxation lift function and only positive heads.

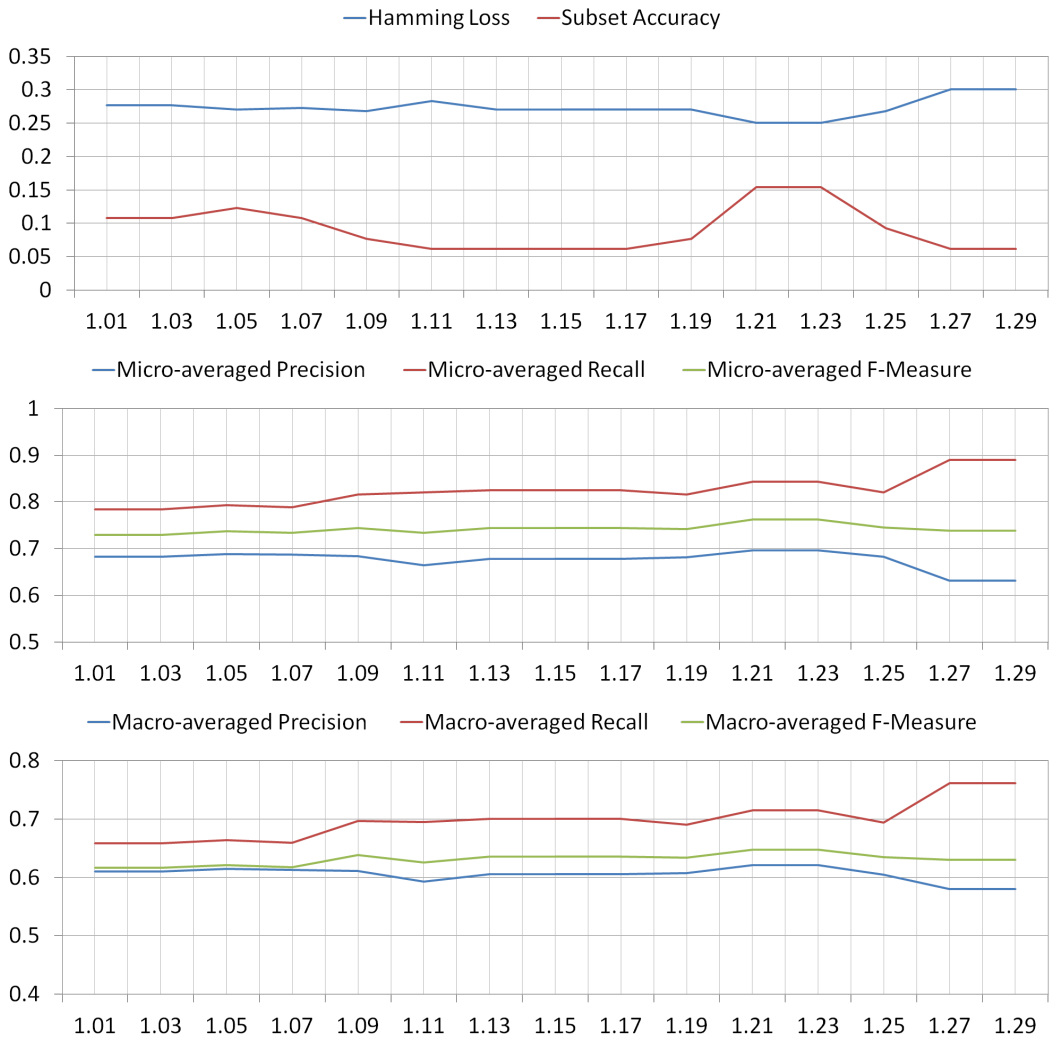

Figure 23: The (1) hamming loss and subset accuracy, (2) micro-averaged precision, recall and f-measure values and (3) macro-averaged precision, recall and f-measure values in comparison to the relaxation lift applied while using the KLN relaxation lift function and only positive heads.

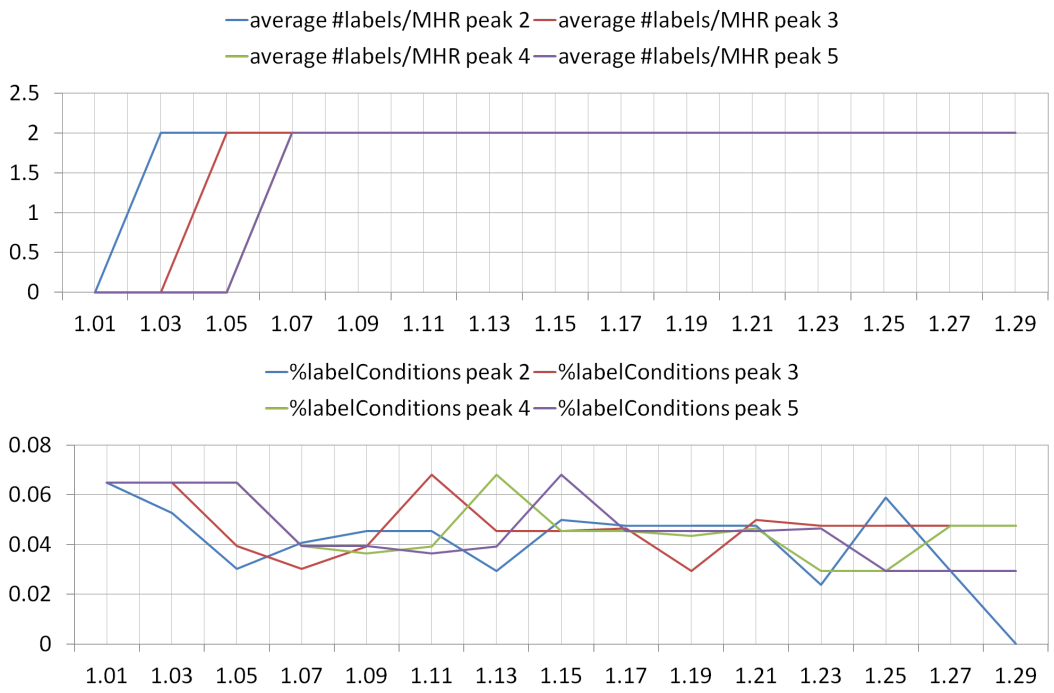

Figure 24: The (1) average labels per multi head rule and (2) percentage of label conditions for each peak label in comparison to the relaxation lift applied while using the peak relaxation lift function and only positive heads.

Model Characteristics

Figure 24 depicts an important finding for this data set and the utilization of positive heads. No rules with more than two labels in the head are learned. Even for a lift of almost 30 percent at peak label three. Therefore, as we observed in Figure 22 (1), more multilabel head rules are learned but they always contain two labels in the head. This may also explain why the percentage of label conditions does, in contrast to the utilization of negative heads, not decrease as much (cf. Figure 24 (2)).

Heuristic Evaluation

Figure 25 shows the hamming loss and subset accuracy values for the peak lift function with peak labels two and three, respectively. The best setting can be deduced easily – a lift of 1.13 for peak label two and a lift of 1.19 for peak label three. For this data set and only positive head rules, the peak label seems to just shift the best lift value to the right. Another sign for this theory is the drop at lift 1.07 for peak label two and the drop at lift 1.09 for peak label three. A similar observation can be done for peak labels four and five. The reason for this observation can be concluded from regarding the average number of labels per multi head rule (cf. Figure 24 (1)). Due to two labels being the longest reasonable head length for this data set and only positive heads, a higher peak label does not make a difference. The lift is just expressed differently. Therefore, for a higher peak label, the same lift is achieved at a higher lift than for a lower peak label. Resulting in the observed shift.

In contrast to the macro-averaged values when utilizing negative heads, the macro-averaged values when only using positive heads do not drop significantly but instead increase (cf. Figure 26 (2)). The micro-averaged values, as depicted in Figure 26 (1), stay rather constant. Again, the reason for both observations being that the head length seems to be limited to a maximum of two labels. As a result, utilizing only positive heads and relaxing the pruning should be further analyzed on another data set (cf. Section 9).

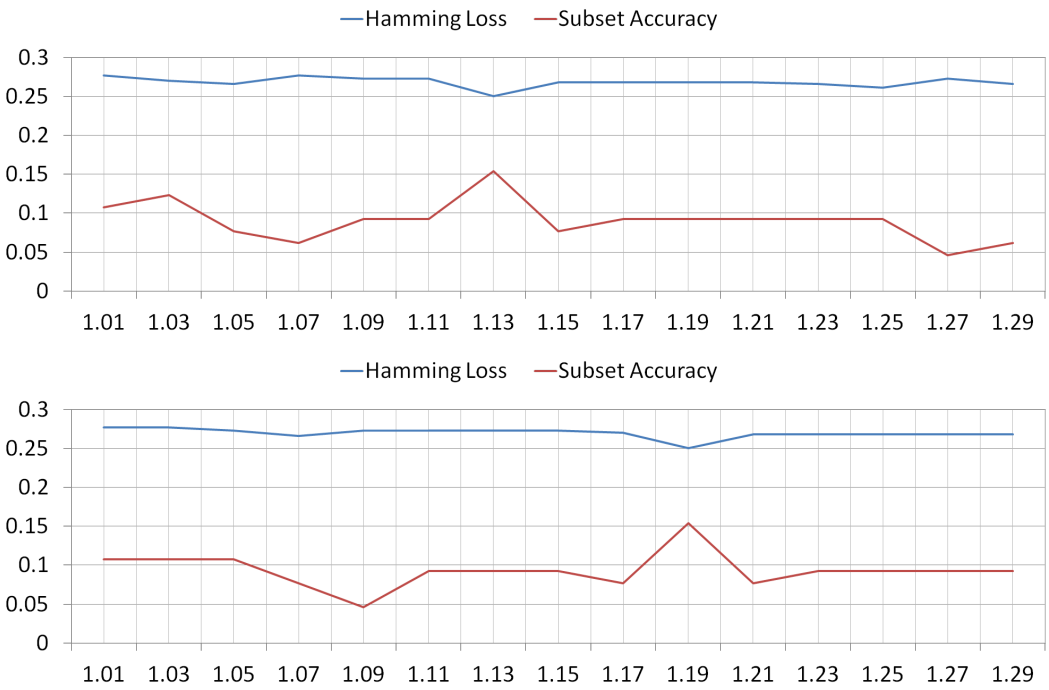

Figure 25: The hamming loss and subset accuracy values for (1) peak label two and (2) peak label three in comparison to the relaxation lift applied while using the peak relaxation lift function and only positive heads.

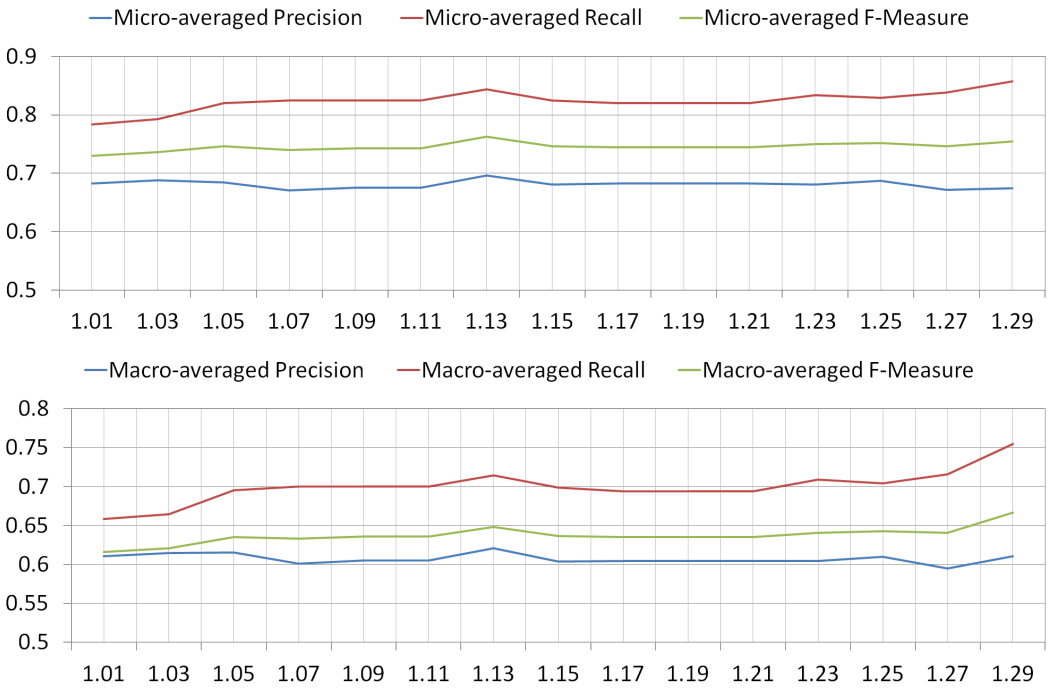

Figure 26: The (1) micro-averaged and (2) macro-averaged precision, recall and f-measure values for peak label two in comparison to the relaxation lift applied while using the peak relaxation lift function and only positive heads.

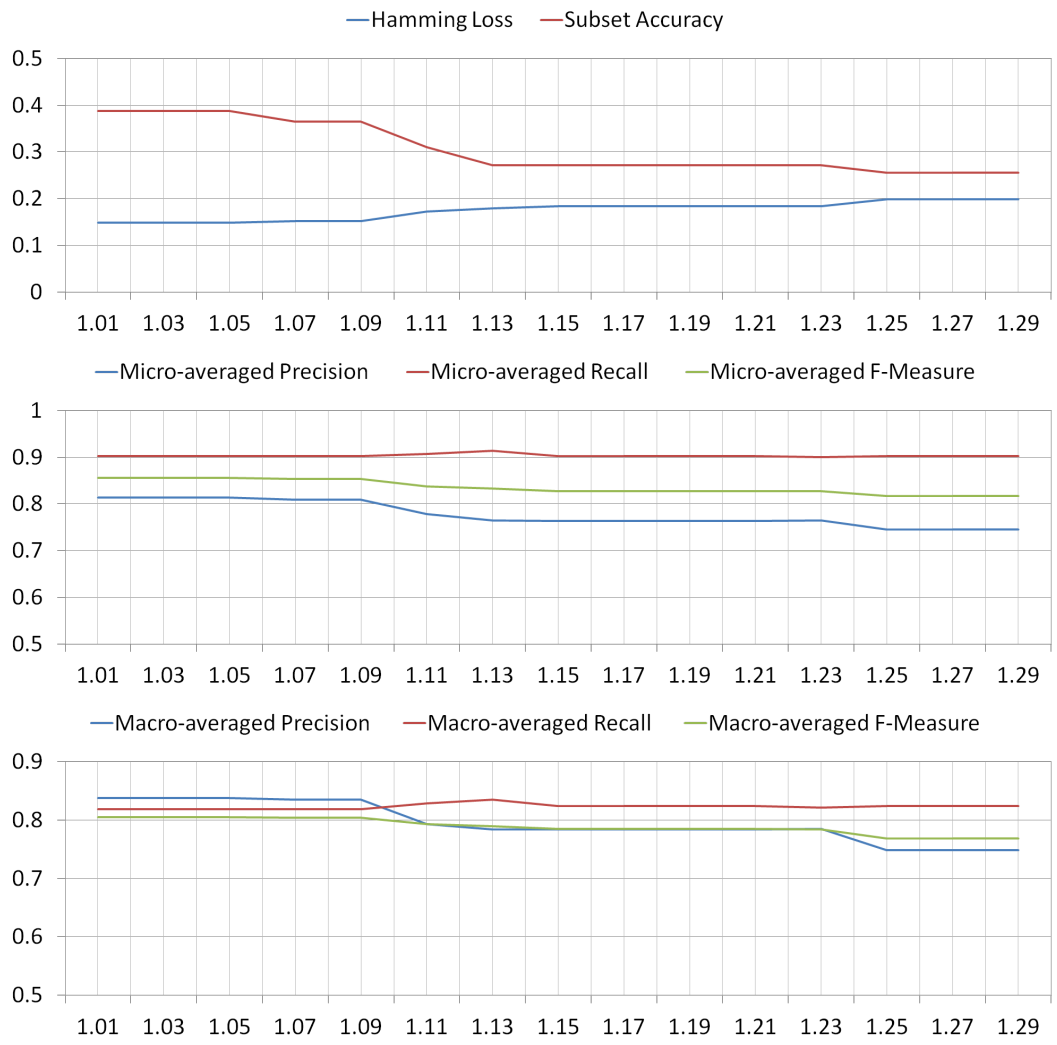

Figure 27: The (1) hamming loss and subset accuracy, (2) micro-averaged precision, recall and f-measure values and (3) macro-averaged precision, recall and f-measure values *on the training set* in comparison to the relaxation lift applied while using the peak relaxation lift function with peak label five and only positive heads.

7.3.3 Training Set Fit

Contrary to the fit on the training set while learning positive and negative head rules, the fit on the training set learning only positive heads does not decrease as much. Figure 27 illustrates the behaviour of the peak relaxation function with peak label five on the training data. We can observe that while the values still decline, they do not decline as much. The behaviour is a bit different for subset accuracy, however. When learning also negative heads, subset accuracy increased again for some values (cf. Figure 20). A likely reason for both observations is that rules with fewer labels in the head are learned when only learning positive head rules. We have discovered earlier that for positive head rules only multilabel rules with a head of length two get induced at maximum. Thus, the training set does not get generalized too much, as is the case when learning also negative heads, which typically results in longer heads. This may also be the reason why subset accuracy increases at some point again. If more labels are learned and set simultaneously, the chances of learning and setting frequently co-occurring labels so that they perfectly classify some of the examples increases. Thus, the setting of the labels is not as dependent on individual rules for each label. Rather, the labels are set in groups. Due to the maximum head length being two for positive multilabel head rules, the behaviour of other peak labels is very similar.

7.3.4 Summary

Despite a label cardinality of 3.392 for the data set FLAGS, at maximum rules with two label in the head get induced. This is a trend that is also observable in the evaluation in Section 9. As a result, utilizing the peak relaxation lift function with peak label two seems to be advantageous. A bigger peak label does not imply longer heads, the lift is just expressed differently. However, with a peak label of two, efficiency is better than for higher peak labels as we can start pruning regularly from the peak onward. For the data set FLAGS, utilizing only positive heads seems to be better. The micro-averaged heuristic values slightly increase and the macro-averaged values stay rather constant and do not drop significantly, as is the case for negative heads.

7.4 Discussion

In this chapter, we have analyzed the effects of different parameters on the learned models and the predictive performance. We have seen that the relaxation lift does indeed result in more multilabel heads and thus in most cases in a faster run time. Furthermore, the average number of conditions per rule decreases for a higher lift. The most likely reason being that a longer head in combination with fixing the head is more difficult to refine. Moreover, the percentage of label conditions does not increase significantly for a higher lift, but rather approaches zero. Additionally, we have studied the fit on the training set and concluded that a higher lift results in a more general model, which may very well be due to fewer conditions per rule. Another important finding is that no rules with more than two labels in the head are induced when only learning positive heads. Hence, making the utilization of the peak relaxation lift function with peak label two a very good idea. In general, (micro-averaged) precision seems to increase while recall decreases. This is the case due to the lower boundary on the quality of the rule. For some of the observations it is not entirely clear whether they are a result of the lift or the condition of requiring at least as many true positives as false positives or fixing the head during the refinement process. While we have only performed the sensitivity analysis for the data set FLAGS, we expect similar results from other data sets in general.

8 Prepending

In this chapter, we pursuit another idea for the separate-and-conquer algorithm – prepending. At first, the motivation behind prepending is described, followed by the required changes in the algorithm and a brief comparison between the in this work presented approaches on the basis of a rule set.

8.1 Motivation

Sadly, learning more multilabel head rules using a relaxed pruning approach does not necessarily lead to more label conditions in the rule set. Furthermore, the presented algorithm still has some problems with more difficult data sets. For those, the algorithm predicts every label to be irrelevant or relevant, resulting in a bad model. Another reason for trying a different approach is the fact that due to the updated coverage and therefore excluded examples, later rules might be more difficult to understand in comparison to previously induced rules. This fact can also lead to somewhat contradictory rules being learned. Moreover, due to the relaxed pruning approach, we make more mistakes (on the training set) – especially for a high lift. As a result of these reasons, we try to use an approach we call prepending.

The general idea is to initially set all labels in the training set to be irrelevant, to learn the default rules in the beginning so to speak. Hence the name prepending (cf. [Webb, 1993]). After the initial default rules, we let the algorithm correct its (wrong) predictions, which can then also take in account the default rules in the learning process. Thus, we mainly focus on the relevant labels, because all labels are considered to be irrelevant initially. By correcting previous predictions, the rule set might also be more expressive and easier to understand because later predictions do directly incorporate previously made predictions. Additionally, we are able to correct the incorrectly set labels not only by the relaxed approach, but also the not avoidable mistakes of the original algorithm. Furthermore, we hope to learn more label dependencies by using prepending. A possible reason for more label conditions could be the explicit modeling of exceptions for instance.

8.2 Correction of Predictions

In order to be able to effectively correct the predictions of the classifier, we need to redefine the semantics of the entries in the confusion matrix. With our new semantics we aim at correcting the values in a reasonable way as described in the motivation. As a result, we redefine:

- **TP**: The value is corrected by the new prediction.
- **FP**: The value was correct but our new prediction distorts the value.
- **FN**: The value needs yet to be correct, which also includes uncovered examples.
- **TN**: We do not use true negatives in this context.

Note that the definition of true positives and false positives only applies to examples covered by a rule. By changing the semantics of the confusion matrix entries we also change the semantics of the multilabel evaluation metrics:

- **Precision** is defined as $\frac{TP}{TP+FP}$. Applying our new semantics, precision calculates the percentage of predictions that do indeed correct the value.
- **Recall** is defined as $\frac{TP}{TP+FN}$. Applying our new semantics, recall calculates the percentage of correcting predictions among the values that needed correction.

Consequently, the f-measure trades off these two values with their new semantics, i.e. trying to maximizing the total number of corrections as well as the percentage of correcting predictions.

Now that we have informally described the new semantics, we need to define the entries of the confusion matrix more formally. The conditions and how they are counted can be gathered from Table 13. The value of the true label vector for the given label is denoted as *True Value*, which is available as all calculations are executed on the training data. *Set Value* describes the value that is currently set for an example and label – zero initially. Our objective is then to match the set value and the true value. Furthermore, *Prediction* represents the value of the label attribute for the current label of the rule to be evaluated. Note that the rule must cover the given example for the rule to be applicable.

True Value	Set Value	Prediction	Type
0	0	0	-
0	1	0	TP
0	0	1	FP
0	1	1	FN
1	0	0	FN
1	1	0	FP
1	0	1	TP
1	1	1	-

Table 13: Definition of true positives, false positives and false negatives if an example is covered according to whether or not they correct a previous prediction.

In the case that all values match for a covered example, the value is already correct. As a result, no correction can be made and we do not count this case in the confusion matrix. If the true value and the set value of a covered example do not match and the prediction would set the value to the true value, we count this case as a true positive as the value is corrected. If the true value and the set value match for a covered example and we would distort this value so that it is not correct anymore, we count this case as a false positive. Moreover, if a rule covers an example, the true value does not match the set value and the prediction does not correct the value, we count this case as a false negative. Furthermore, if a rule does not cover an example and the true value does not match the set value, it is counted as a false negative as well. Hence implying the change in the semantics.

8.3 Algorithm Adjustments

This section covers the necessary changes to the previously presented separate-and-conquer algorithm and their resulting effects. At first, the implementation of the default rules for prepending and required changes to construction of the confusion matrix are explained. Moreover, the overwriting of values in the training and test set is elaborated. As a result, we also need to make some adjustments in terms of coverage of examples. Finally, the adjustments of the subroutines for finding the best head are discussed.

8.3.1 Prepending Default Rules and Aggregation

There are two possibilities for implementing the default rules for setting all labels to be irrelevant initially. The first possibility is to initially generate a rule with no condition in the body and a label condition in the head that sets the label to be irrelevant for every label. Then, the rule can be applied to the data set. Using this approach, we would need to change the way we calculate rule statistics. The second approach, which we use in the implementation, is to interpret missing values in the training data as zeros in the subroutine AGGREGATE. In order for this approach to work, we need to set all labels to be eligible to be used as label conditions initially. This requires adjusting or completely removing the

subroutine GETLABELCONDITIONS. By allowing all label conditions from the beginning, we theoretically also allow label conditions that require a label to be relevant. Despite the fact that no labels are set to be relevant yet, this is not a problem as these rules do not cover any example in the training set. Hence, these rules cannot have any true positives, which prevents the induction of these rules.

Algorithm 29 The adjusted algorithm AGGREGATE for aggregating the true positives, false positives, true negatives and false negatives of a confusion matrix if using prepending.

Require: Rule r, training example $(X_j, \hat{Y}_j)$, label λ_i, confusion matrix C

```
 1: procedure AGGREGATE
 2:     if r covers X_j then
 3:         if r.head contains a label attribute y_i then        ▷ always true for rule-dependent evaluation
 4:             if ẏ_i ∈ Y_j ≠ ŷ_i ∈ Ŷ_j then                    ▷ set value of example does not match true value
 5:                 if y_i = ŷ_i ∈ Ŷ_j then
 6:                     C.tp+ = 1                                ▷ value is corrected
 7:                 else
 8:                     C.fn+ = 1                                ▷ value still needs correction
 9:                 end if
10:             else
11:                 if y_i ≠ ŷ_i ∈ Ŷ_j then
12:                     C.fp+ = 1                                ▷ value was correct, distorted by prediction
13:                 end if
14:             end if
15:         end if
16:     else
17:         if ẏ_i ∈ Y_j ≠ ŷ_i ∈ Ŷ_j then
18:             C.fn+ = 1                                        ▷ still needs correction
19:         end if
20:     end if
21: end procedure
```

Algorithm 29 illustrates how the true positives, false positives, true negatives and false negatives are counted in accordance to the definition in Section 8.2. The current label vector of training example $(X_j, \hat{Y}_j)$, which we keep track of, is denoted as Y_j. Furthermore, the fact that missing values are considered to be irrelevant is omitted in the depicted algorithm due to simplicity. Basically, if $\dot{y}_i \in Y_j = ?$, we treat the value like it was zero. Moreover, it should be mentioned that the depicted algorithm does only work for rule-dependent evaluation, because the subroutine AGGREGATE is then only called for labels contained in the head. Otherwise, we would need to consider the case if the label attribute is not present in the head, which occurs if using a rule-independent evaluation strategy. Additionally, a rule-dependent evaluation seems to make more sense for prepending, because the other labels are typically already set to be irrelevant.

In order for the true positives, false positives and false negatives to be counted correctly, we also need to adjust the subroutines MICROAVERAGING, LABELBASEDAVERAGING, EXAMPLEBASEDAVERAGING and MACROAVERAGING, which were presented in Section 4.7. Exemplary, we adjust the subroutine MICROAVERAGING to be usable by prepending. The other subroutines can be adjusted analogously. In each algorithm, we need to remove the condition that prevents fully covered examples from being considered in the construction of the confusion matrix (cf. Algorithm 16, line 4). Again, the reason being that we may still correct values, even for fully covered examples. That being said, the adjusted subroutine MICROAVERAGING is illustrated in Algorithm 30. Similarly, the condition can be removed from the other averaging subroutines.

Algorithm 30 Adjusted algorithm MICROAVERAGING for measuring the performance of a multilabel head rule using micro-averaging if using prepending.

Require: Original training data set T, current trainign data set $T_{current}$,
rule r, evaluation function δ, relevant labels L

```
1:  procedure MICROAVERAGING
2:      C = (0, 0, 0, 0)                          ▷ initialize global confusion matrix
3:      for each example (X_j, Ŷ_j) ∈ T do
4:          for each relevant label λ_i ∈ L do
5:              AGGREGATE(r, (X_j, Ŷ_j), λ_i, C)
6:          end for
7:      end for
8:      h = δ(C)                                  ▷ apply evaluation function on confusion matrix
9:      return performance h
10: end procedure
```

8.3.2 Overwriting of Values

In order to apply the corrections, previously set values must be overwritten. This stands in contrast to the approach of the separate-and-conquer algorithm, for which each label in every example can only be set once. Furthermore, in the original algorithm, once an example is covered sufficiently, it is removed. The overwriting of values must be permitted on the training set as well as on the test set. Hence, we need to change the application of the multilabel decision list, which was introduced in Section 2.2.3.

Algorithm 31 Adjusted application of a multilabel decision list to a test example if using prepending.

Require: Test example X, multilabel decision list $\mathbb{D}$

```
1:  Y = (?, ..., ?)
2:  for each rule r in decision list 𝔻 do          ▷ in the order of insertion
3:      set all labels in Y to be irrelevant initially
4:      if r covers X then
5:          apply head of r on Y
6:          if r marked as stopping rule then
7:              return Y
8:          end if
9:      end if
10:     return Y
11: end for
```

The adjusted application of the multilabel decision list is illustrated in Algorithm 31. First of all, the condition for applying the head on the label vector needed to be removed (cf. Algorithm 31, line 4). This corresponds to the overwriting of the values. Moreover, just as we learn rules with the assumption that all label entries are irrelevant initially, we need to do the same for the test examples. Otherwise, label conditions that depend on irrelevant labels in the body of a rule do not get applied correctly to the test example. Hence, instead of assuming all remaining labels in Y to be irrelevant in the end, we prepend this assumption to the beginning of the classification process. Furthermore, the possibility of stopping rules has to be regarded. More information on the induction of stopping rules can be found in the next section. By possibly trying and applying all learned rules on each test example, the classification process of test examples is expected to take longer than for the unadjusted multilabel decision list.

Sunset = 0 ← FallFoliage = 1
FallFoliage = 0 ← FallFoliage = 1, Att110

Figure 28: Example of why the head should not be fixed in the rule refinement process of prepending. The second rule is the refinement of the first rule, taken from the learning process of the algorithm.

8.3.3 Coverage

Prepending requires to overwrite previously set values of labels. As a result, we may not remove examples from the training set because even if the example is fully covered we could still correct predictions. This leads to numerous complications. For once, we cannot rely on coverage as a stopping criterion. Previously, once we cover all labels of an example or most of the training set, we remove the example or terminate the algorithm, respectively. Thus, prepending does not have a stopping criterion currently – it learns rules until there are no rules left to learn. Due to the lower bound on the quality of the rule, the number of rules are limited and the algorithm terminates at some point. As coverage cannot be used in the same way as by Loza Mencía and Janssen [2016] and Rapp [2016], we need to remove all coverage related parts in Algorithm 3. This includes, as mentioned, the stopping criterion that is dependent on the percentage of covered examples (cf. Algorithm 3, line 4) as well as the re-inclusion of training examples, which is not required for prepending (cf. Algorithm 3, lines 7 and 8). Additionally, stopping rules cannot be learned as they are based on coverage as well. Summing up, we need a new way of terminating the algorithm and inducing stopping rules, which we will address again in the evaluation and the future work section. Nevertheless, having all training examples available for the induction of label-dependent rules might result in the revelation of more valid dependencies between labels.

8.3.4 Finding The Best Head

Contrary to the relaxed approach, we do not fix the head for prepending. The reason for that lies in the rule learning process and the correction of predictions. The rule learning process starts with evaluating each possible body of length one. After determining the best rule (with the best body of length one), the algorithm tries to refine the rule by adding more conditions. Therefore, if we fix the head we cannot exploit the specialized body. For prepending, this is important because we try to correct predictions. These corrections are often exceptions to certain other rules or label values. Especially after setting labels to be relevant, we typically want to correct some values that have been incorrectly set to be relevant. Thus, by not fixing the head, we increase the chances of finding a body that models these exact exceptions. Figure 28 illustrates the need for allowing the head to change. Note that we omit the exact value of the attribute for clearness. In the shown rule refinement process, we first learn a fully label-dependent rule. If *FallFoliage* is relevant for an example, label *Sunset* is irrelevant. By trying all refinements of this label condition, we hit a condition that models an exception to the relevancy if *FallFoliage* is relevant. As a result, the head needs to be changed to be predict the irrelevancy of *FallFoliage*. If we had fixed the head, we would probably not have found this interesting rule, which further models a label dependency. Of course we may also reveal exceptions without label conditions by allowing the head to change again, as is the case in the original algorithm by Rapp [2016].

A key change to the subroutines for finding the best lifted head is the lower boundary for the quality of the rule. In Section 6.2 we introduced the condition that a rule must have at least as many true positives as false positives in order to be induced. For prepending, however, rules must have more true positives than false positives. By requiring this condition, we prevent infinite loops in the rule learning process, because we make sure that there is always some progress in the correction of the values. We cover this problem in more detail in Section 8.4

Class12 = 1, Class13 = 1	(2250, 750)
Class4 = 1, Class5 = 1 ← Att61	(230, 94)
Class2 = 1, Class3 = 1	(1280, 1720)
Class2 = 0, Class3 = 0	(1720, 1280)
Class2 = 1, Class3 = 1	(1280, 1720)
Class2 = 0, Class3 = 0	(1720, 1280)

Figure 29: Example of an infinite loop while learning a model for the data set CAL500 and omitting the lower boundary.

Instrument-Drum-Set = 0, NOT-Song-Very-Danceable = 0	(269, 269)
Instrument-Drum-Set = 1, NOT-Song-Very-Danceable = 1 ← NOT-Emotion-Angry-Agressive = 1	(269, 269)
Instrument-Drum-Set = 0, NOT-Song-Very-Danceable = 0	(269, 269)
Instrument-Drum-Set = 1, NOT-Song-Very-Danceable = 1 ← NOT-Emotion-Angry-Agressive = 1	(269, 269)
Instrument-Drum-Set = 0, NOT-Song-Very-Danceable = 0	(269, 269)

Figure 30: Example of an infinite loop while learning a model for the CAL500 data set and using only a condition that requires at least as many true positives as false positives.

8.4 Loop Prevention

As a result of allowing values to be overwritten, we need to prevent loops from occurring. Otherwise, the algorithm can get stuck infinitely by always switching the values of the same labels. For preventing loops, the lower boundary for the quality of the rule is crucial. By requiring more true positives than false positives, we make sure the algorithm makes progress. Implying, we always correct more values than we distort. Therefore, the algorithm terminates at some point. If we excluded this condition, a situation such as shown in Figure 29 is likely to occur. Again, each rule is augmented with its number of true positives and false positives (TP, FP). Due to the missing condition on the true and false positives, the algorithm always switches the same labels back and forth. Resulting in no progress being made and therefore an infinite loop.

To prove that the number of true positives must actually be strictly greater than the number of false positives, contrary to the lower boundary in the separate-and-conquer algorithm, we regard the example given in Figure 30. We observe that an infinite loop can still occur. Therefore, we need to change the implementation of the subroutines DECOMPOSABLE, ANTIMONOTONOUS and REFINERULE. Instead of requiring the condition for each individual rule, we enforce the strictly greater condition when the algorithm is done with refining a rule. Before that, the rules are allowed to have as many true positives as false positives. This results in better rule sets as it is very likely that rules with an equal number of true positives and false positives are still refined so that, in the end, the number of true positives is strictly greater than the number of false positives. If that is not the case after refining the rule, the learned rule is removed from the rule learning process and the learning process iteration is repeated. In this repeated iteration, the strictly greater condition is enforced for every rule. Afterwards, the algorithm continues as usual. Pseudocode for this operation is omitted due to clarity. To simplify the implementation and not having to implement (almost) the same subroutines twice, it might make sense to just use the same procedure in the separate-and-conquer algorithm using the relaxed approach as well.

Due to this condition not present in the original separate-and-conquer algorithm for learning multi-label head rules by Rapp [2016], it cannot be used for prepending. In order to change that, said condition must be included in the subroutines DECOMPOSITE and PRUNEDSEARCH.

8.5 Rule Set Comparison

In this section, we briefly compare the rule sets for the different approaches presented in this work and give an example of how prepending works. Table 14 shows the rule sets obtained by executing the original algorithm, the relaxed pruning approach and prepending on the data set WEATHER. The data set WEATHER is a small toy data set and has three labels that represent whether or not to play a game, do not play a game or maybe play a game on a day given the weather forecast.

The rule sets show the differences between the algorithms (cf. Table 14). The rule set learned by the original approach has the most rules. We can see that the relaxed approach learns more multilabel rules. The first two rule sets furthermore both learn the implication that if dontplay = 0, then play = 1. However, the relaxed pruning approach learns this rule a bit later. Nevertheless, the relaxed pruning approach learns another label dependency (the last rule), whereas the original algorithm learns the same label-dependent rule twice. Moreover, the relaxed pruning approach models more label co-occurrences. For instance, the original algorithm learns no rules that relate the label *playmaybe* to the other two labels, whereas the relaxed pruning approach learns three such rules. Nevertheless, the first two rule sets are more difficult to understand as we have to consider the removed training examples and thus learned label dependencies do not apply globally.

dontplay = 1 ← outlook = sunny, humidity ≥ 82.5 (3, 0)
dontplay = 0 ← temperature ≥ 71.5 (5, 0)
playmaybe = 0 ← outlook = sunny (5, 0)
play = 1 ← dontplay = 0 (5, 0)
play = 0, dontplay = 1 ← temperature ≥ 70.5 (5, 0)
play = 1, dontplay = 0 ← temperature ≥ 66.5 (6, 0)
dontplay = 1 ← temperature ≥64.5 (1, 0)
dontplay = 0 (1, 0)
play = 1 ← dontplay = 0 (1, 0)
play = 0 (1, 0)
playmaybe = 1 ← temperature ≤ 70.5, play = 1 (3, 0)
playmaybe = 1 ← temperature ≥ 82.0 (1, 0)
playmaybe = 0 (5, 0)
playmaybe = 0 ← outlook = sunny (5, 0)
play = 1, dontplay = 0 ← outlook = overcast (8, 0)
dontplay = 1 ← humidity ≥ 82.5, humidity ≤ 95.5 (4, 0)
dontplay = 0 ← temperature ≥ 66.5 (5, 0)
play = 1 ← dontplay = 0 (5, 0)
play = 0, dontplay = 1 (6, 0)
dontplay = 0, playmaybe = 0 ← temperature ≥ 70.5, temperature ≥ 72.0, temperature ≤ 82.0 (6, 0)
play = 0, playmaybe = 1 ← dontplay = 0 (4, 0)
play = 0, dontplay = 0, playmaybe = 0 (2, 0)
play = 1 ← humidity ≤ 80.0, temperature ≥ 66.5 (5, 0)
play = 1 ← outlook = overcast (3, 0)
play = 1 ← humidity ≥ 95.5 (1, 0)
play = 1, playmaybe = 1 ← temperature ≤ 70.5, humidity ≥ 80.0, humidity ≥ 80.0 (2, 0)
play = 1, playmaybe = 1 ← temperature ≤ 64.5, temperature ≤ 84.0 (1, 0)
play = 0, dontplay = 1 ← outlook = sunny, humidity ≥ 72.5 (3, 0)
play = 1, playmaybe = 1 ← temperature ≥ 82.0, humidity ≥ 85.5 (1, 0)
play = 0, dontplay = 1, playmaybe = 0 ← temperature ≤ 66.5, humidity ≥ 70.0 (1, 0)

Table 14: Three rule sets obtained by executing (1) the original algorithm, (2) the relaxed pruning approach with peak label three and a lift of 1.10 and (3) prepending (same lift setting) on the data set WEATHER.

In comparison, prepending learns far more rules that focus on the relevant labels. This is because we set all labels to be irrelevant initially and then correct the wrong values. While no explicit label conditions are learned, we can observe frequently co-occurring labels and that *play* and *dontplay* seem to exclude themselves. The algorithm focuses specifically on predicting the relevancy of *play* as this seems to be the label that has the most relevant entries in the data set, whereas *dontplay* is mostly irrelevant and therefore mostly ignored in the rule set as it is set to be irrelevant initially. A reason for not learning any label-dependent rules might be that they are not required for this data set. The algorithm is perfectly capable of correcting every value without the help of label dependencies. Interestingly, the attribute windy is not used once as a condition and does not seem to be relevant for the prediction. Sadly, the algorithm also learns some redundancies in the bodies of the rules.

In order to further the understanding of prepending, we apply the learned rule set depicted in Table 14 on a test example. The process is depicted in Table 15. At first, the labels of the test example are all set to be irrelevant. Then, we check for every rule whether it is applicable. The first rule that is applicable is the second one. Hence, *play* is predicted to be relevant. Due to the attribute temperature, the fifth rule is the next applicable one. Therefore, both *play* and *playmaybe* are set to be relevant. Note that the setting of *play* does not make a difference in this case. The remaining rules do not cover the example. Thus, the classification process terminates.

outlook	temperature	humidity	windy	play	dontplay	playmaybe
overcast	64	65	TRUE	0	0	0
overcast	64	65	TRUE	1	0	0
overcast	64	65	TRUE	1	0	1

Table 15: Example of the application of a learned rule set to a test example.

8.6 Discussion

In this chapter, we have regarded an alternative approach to learning multilabel rules, which also takes advantage of the relaxed pruning. We have further presented the necessary changes to the previously introduced algorithm. The most important changes include the inclusion of all examples in the entire learning process, i.e. no examples are removed, and the loop prevention, for which we require each learned rule to have more true positives than false positives, i.e. correct more values than it distorts. For prepending, it is also important to allow the head to contain the same label as the body as long as the condition is different. As a result of including all examples, we cannot simply rely on coverage as a stopping criterion as the values might still be corrected later. This is something that will need to be fixed in future works.

9 Evaluation

By evaluating the relaxed pruning approach on multiple data sets and comparing the results to the original approach, we hope to further explore the influence of the relaxation lift and determine whether or not searching the best performing head according to the lifted heuristic value achieves the desired effects. The implementation is based on the implementation of the original algorithm by Rapp [2016], which utilizes the SeCo-framework[4] for rule learning, which has been developed at Technische Universität Darmstadt [Janssen and Fürnkranz, 2010; Janssen and Zopf, 2012]. As stated by Rapp [2016], the implementation does not focus on high-performance computations, but is rather a proof-of-concept. Similarly, this work does not focus on high-performance computations either. As a result, the algorithm is not applicable on large data sets. The data sets, which have been used for the evaluation of the relaxed pruning approach, are listed in Table 16. These multilabel data sets are provided by the developers of Mulan[5], which is a Java library for multilabel learning. Each data set listed in Table 16 comes with a pre-separated training and test set, which are used in this evaluation. The listed data sets cover several domains, nominal as well as numeric features, different number of labels and, maybe most importantly, different label cardinalities (average number of relevant labels per example). The label cardinality is important because for a data set with a label cardinality of almost one, it may not make sense to use the relaxed approach if learning only positive heads. Learning reasonable multilabel head rules, i.e. heads with two or more labels in the head, seems to be unlikely in the case that each example only has one relevant labels on average.

Name	Domain	Instances	Nominal	Numeric	Labels	Cardinality	Density	Distinct
CAL500	Music	502	0	68	174	26.044	0.150	502
EMOTIONS	Music	593	0	72	6	1.869	0.311	27
MEDICAL	Text	978	1449	0	45	1.245	0.028	94
FLAGS	Images	194	9	10	7	3.392	0.485	54
SCENE	Image	2407	0	294	6	1.074	0.179	15
BIRDS	Audio	645	2	258	19	1.014	0.053	133
YEAST	Biology	2417	0	103	14	4.237	0.303	198

Table 16: Characteristics of the data sets, which are used in the evaluation. The columns from left to right specify the name of the data sets, the domain of the instances, the number of instances, the number of nominal and numeric features, the total number of labels, the average number of labels per instance (cardinality), the average percentage of relevant labels (label density) and the number of distinct labelsets in the data. CAL500, SCENE and YEAST were discretized in order to allow the experiments to complete.

In the remainder of this chapter, we use said data sets in combination with different settings of the relaxed and normal pruning approach. Henceforth, the relaxed pruning approach is denoted by an R and the normal approach by an N. Furthermore, an important setting is whether or not to only learn only positive or also negative heads, which is indicated by $+$ or $+-$, respectively. For each approach, we use different multilabel evaluation metrics for the calculation of heuristic values of rules. Except for precision and recall, all evaluation functions presented in Section 2.3.1 are utilized. We forgo those metrics as we believe they are not as suited for the induction of good models as the other metrics. However, both metrics are used in combination by the f-measure, for which we set parameter $\beta = 0.5$, which implies a more precision-oriented measure. According to Rapp [2016], this decision has been made with the intention of preventing too many labels from being predicted as relevant. Using recall for learning rules would similarly result in too many labels being predicted, because it does not penalize wrong predictions. In the remainder of the evaluation, we use the following short notations:

[4] http://www.ke.tu-darmstadt.de/resources/SeCo

[5] The used data sets as well as others can be found at http://mulan.sourceforge.net/datasets-mlc.html.

- **HA:** Denotes hamming accuracy as defined in Section 2.3.1, i.e. the percentage of correctly predicted labels among all labels. In this work, we use the micro-averaged hamming accuracy as a base learner for the induction of the model.
- **SA:** Denotes subset accuracy as defined in Section 2.3.1, i.e. the percentage of perfectly classified examples. We use example-based averaging in combination with subset accuracy in order to be able to take advantage of anti-monotonicity.
- **F:** Denotes the f-measure with $\beta = 0.5$ as defined in Section 2.3.1. The f-measure trades off precision and recall. As a result of the parameter β, we use a more precision-oriented metric.
- **Mic:** Denotes micro-averaging as defined in Section 2.3.1. Thus, for a evaluation function using micro-averaging, a global confusion matrix is constructed before applying the evaluation metric.
- **Mac:** Denotes macro-averaging as defined in Section 2.3.1. Thus, an evaluation function is applied on each atomic confusion matrix and then averaged over both labels and examples.

Moreover, we always use a rule-dependent evaluation strategy for our experiments. This decision has been made because Rapp [2016] discovered that a rule-dependent evaluation of rules typically results in better models and is more in line with the process of applying the learned multilabel decision list on a test set. In the following sections, we first regard some characteristics of the learned models using the relaxed approach and the separate-and-conquer algorithm. Afterwards, we interpret the predictive performance of the classifiers. Then, we regard some exemplary rule sets and try to determine good relaxation lift values. Finally, the in Section 8 introduced approach prepending is evaluated on a smaller scale. All experiments have been carried out on the Lichtenberg high performance computer at Technische Universität Darmstadt.[6]

9.1 Setup

For the evaluation of the discovery of multilabel rules by relaxed pruning, we need to try different relaxation lift settings and values. As previously mentioned, due to the similarity of the KLN and root relaxation lift function, we only use the former one in the evaluation. This furthermore also significantly reduced the number of lift settings to be tested, which was a problem initially. For the same reasons, we restrict ourselves to evaluating the normal pruning approach on a single parameter setting – the default settings. Otherwise, we we would need to try all these settings for the relaxed pruning approach as well. This would result in far too many possible combinations needed to be executed. For the relaxed pruning approach, we use the same basic parameter settings. Hence, a certain comparability of the results is given. For the relaxation lift parameters, we explore the peak and the KLN relaxation lift function. For each function, we further try multiple extents of the lift and, in the case of the peak lift function, also different peak labels. However, we only use a single curvature. The effects of the curvature are still unclear and it is questionable whether they have a significant impact. A different value for the curvature might not make a difference for similar reasons why the KLN and the root lift functions are so similar: As the relaxation lift values are rather low (typically not bigger than 1.3) and we additionally only need to evaluate the functions for whole numbers, the difference in the values is low as well. Henceforth, we utilize a curvature of 2.0 for the peak relaxation lift function.

In order to be able to set the extent of the lift for the KLN relaxation lift function in a comparable way, we set the parameter k of the function in accordance to the desired relaxation lift for a specific head length. In this evaluation, we set this head length to be three. The extent of the lift is then raised by increasing the desired lift for this head length. To be precise, we start with a lift of 1.01 and increase the lift in steps of 0.02 until a lift of 1.19, after which we stop. We only increase the lift in steps of 0.02

[6] https://www.hhlr.tu-darmstadt.de/hhlr/

instead of steps of 0.01 in order to reduce the number of parameter settings. Otherwise, the experiments take too long and require too many resources. Of course, one could also express the extent of the relaxation lift differently but for the evaluation this setting seems fitting.

For the peak relaxation lift function we need to increase the peak label and for every peak label we need to try the same lift values as for the KLN lift function. Therefore we start with a peak label of two and increase this peak label by one until a maximum peak label of five. Additionally, we omit peak label four in the evaluation as it might be quite similar to utilizing peak label five. Then, for every possible peak label value we increase the lift from 1.01 to 1.19 in steps of 0.02, identical to the procedure for the KLN relaxation lift function.

Using these parameter settings, 40 combinations need to be tested for each data set. For each data set, the best parameter setting is determined on the training set. Initially, the idea was to do a ten-fold cross-validation on the training set in order to determine the best parameter setting. However, due to run time restrictions, a five-fold cross-validation was employed. After determining the best lift setting on the training set, the algorithm with the best parameter setting is applied to the test set. The best setting during the search was measured according to the base learner chosen. For instance, if using the micro-averaged f-measure as a base learner, the micro-averaged f-measure is the decisive measure for determining the best setting. In the case that these values are the same for two or more values, hamming accuracy and subset accuracy are the tie breakers. If the values are still the same, something which rather frequently occurred, we settle for the parameter setting with the lowest lift value. The idea behind this criterion is that a lower lift might lead to a more accurate model instead of a higher lift because the bias towards rules with a lower heuristic value is smaller.

9.2 Model Characteristics

The intention of the relaxed approach was to increase the number of learned multilabel rules and hence hopefully increase the number of learned label conditions as well. Both objectives are part of the model characteristics, which are covered in this section. We only regard some selected characteristics that are relevant to our context. Tables 36, 37, 38, 39, 40, 41 and 42 in the appendix show the following characteristics of learned models for different approaches and data sets:

- **#Rules:** The number of learned rules.
- **%MHR:** The percentage of multilabel (head) rules among all learned rules, i.e. the percentage of rules with two or more labels in the head among all rules.
- **#Labels/MHR:** The average number of labels contained in the head of learned multilabel rules.
- **#Conditions:** The average number of conditions contained in the body per learned rule.
- **%LabelConditions:** The percentage of label conditions among all learned conditions.

The following observations have been made about the learned models:

- The number of learned rules typically decreases when using the relaxed pruning approach. There are two noteworthy exceptions to this observation. For once, sometimes the best relaxation lift setting for a data set is the lowest possible lift value. In cases such as this, the number of learned rules does not necessarily decrease. Second, there are cases in which the normal approach achieves a considerably worse performance in terms of evaluation functions than the relaxed approach. Hence, the relaxed approach requires more rules because the normal approach learned too few rules to model the data set effectively. The decrease in the number of rules plays an important part in the run time of the algorithm. As a result, the rule learning process does not have to be executed as often, including searching the best head for each possible body or refinement. Thus, the relaxed

approach is often faster than its counterpart. The reason for requiring fewer rules is the increased coverage of each rule. Due to more labels in the head of each rule on average, the label entries in the training set can be covered by fewer rules and the algorithm terminates earlier.

- The percentage of learned multilabel (head) rules does naturally increase. This is a direct result of introducing a bias towards multilabel rules and utilizing the lifted heuristic value for searching the best performing head. Another effect of this observation is the reduced number of learned rules. Furthermore, the percentage of learned multilabel rules when also learning negative heads is usually higher than when only learning positive heads. When also learning negative heads, the learner has more label attributes to choose from. Additionally, by allowing negative heads, the restriction by the label cardinality is loosened. There exists no such restriction as label cardinality for negative heads. Moreover, they express irrelevant ones instead of relevant ones. Thus, they may be combined with positive heads more easily as they predict different dependencies. Similarly, the average number of labels per multilabel head rule for learning positive and negative heads is bigger than when only learning positive heads.

- When learning only positive heads, the average number of labels per multilabel head rule is almost always 2.00, unless the respective data set has a very high label cardinality, such as CAL500. Therefore it makes sense to utilize the peak relaxation lift function with a peak label of two in such cases. Rules with more labels seem to not get induced, so we might as well take the increased efficiency in terms of pruning.

- Similar as in the sensitivity analysis in Section 7, the average number of conditions per rule seems to decrease for the relaxed pruning approach. This could be a sign of inducing a more general model (fewer conditions per rule) by relaxing the pruning constraints (or the other way around). This assumption is supported by the reduced fit on the training set, which was observed in the sensitivity analysis.

- More often than not the percentage of label conditions decreases. While this must not necessarily be a bad thing, the percentage often drops to zero. If a label condition that only applies to a few examples is removed and another head of a rule containing a label condition grows, the latter label condition becomes more expressive. However, the percentage of label conditions decreases. For label conditions, the position in the decision list is also crucial. The earlier in the decision list (but also not too early), the more of an impact said condition has. Nevertheless, the percentage of label conditions does also regularly increase, especially for the micro-averaged f-measure. It should be noted that for mentioned reasons as well as the fact that the number of rules declines, statements about the percentage of label conditions need to be treated carefully.

As a result, we can safely assume using the relaxed approach leads to more multilabel head rules and thus fewer rules as well as often decreased run time. Furthermore, the decline in the average rule length seems to hold as well in most cases. Due to each multilabel rule having more labels on average when learning positive and negative heads, the corresponding lift must be bigger than for when only learning positive heads. This must also be regarded when defining a peak label. A slightly higher peak label is required when also learning negative heads instead of only positive ones. In terms of label conditions, no precise statements can be made. In order to gain more insight into the models, we will regard some exemplary rule sets in Section 9.4.

9.3 Predictive Quality

We have observed that using the relaxed approach does indeed result in more multilabel heads being learned. In this section, we explore the effects this has on the predictive performance of the learned models. We further investigate for which data sets it makes sense to use the relaxed approach and what can be expected by searching the best performing head according to the lifted heuristic value. Moreover,

we need to consider that applying the relaxation lift was not the only change to the algorithm. For instance, we fix the head for the refinement of the rules and require each rule to have at least as many true positives as false positives. The predictive quality of each approach can be found in Table 43, 44, 45, 46, 47, 48 and 49, respectively. Each value is further augmented with its rank among the results for the respective data set at hand in order to better compare the values. This also allows us to calculate the average ranks for different approaches and data sets. While ranks do not incorporate the magnitude of the difference in the values between approaches, they can be utilized to make statements about trends. Due to some experiments not running through for some data sets when using subset accuracy as a base learner, we first regard the average ranks for each approach excluding subset accuracy. The average ranks with the micro-averaged f-measure, macro-averaged f-measure and hamming accuracy as a base learner are listed in Table 17.

Base Learner		HA	SA	Mic Prec	Mic Rec	Mic F	Mac Prec	Mac Rec	Mac F	Rank
Mic F	N^+	10.14	10.79	10.14	3.43	3.71	6.00	3.00	2.71	6.84
Mic F	R^+	6.00	7.57	7.86	4.93	3.14	5.43	5.43	5.00	5.43
Mic F	N^+_-	6.93	7.14	7.29	9.64	9.21	8.07	9.79	8.64	7.98
Mic F	R^+_-	4.64	6.93	4.79	10.50	8.21	7.21	10.36	9.79	7.39
Mac F	N^+	14.29	13.07	13.29	1.00	12.57	10.57	1.00	7.86	11.95
Mac F	R^+	9.64	9.64	9.86	6.14	5.71	8.43	6.00	6.14	7.79
Mac F	N^+_-	8.93	9.00	9.86	13.36	13.21	13.36	13.29	13.36	11.13
Mac F	R^+_-	5.07	4.36	5.14	11.07	9.07	10.50	11.14	11.36	7.46
HA	N^+	11.14	8.79	10.86	2.86	4.00	6.86	3.29	2.43	6.59
HA	R^+	5.00	6.50	3.36	10.86	9.86	8.43	10.86	10.71	8.02
HA	N^+_-	4.36	6.93	4.00	10.00	8.71	2.00	10.14	8.57	7.14
HA	R^+_-	6.14	7.36	4.00	11.43	10.57	10.57	11.14	11.00	8.77

Table 17: Average ranks for each approach (excluding subset accuracy) across all sevens tested data sets. Furthermore, the combined averaged ranks calculated by combining the average ranks for hamming and subset accuracy as well as micro- and macro-averaged f-measure (in grey).

For determining a trend in the values, we average the average ranks depicted in Table 17 over the evaluation metric. Table 18 shows the average ranks of the normal and relaxed pruning approach for each evaluation function. We can observe that the ranks for hamming and subset accuracy become significantly better when using the relaxed pruning approach. Hence, we can conclude that using the relaxed approach usually results in an increase in hamming and subset accuracy. Furthermore, micro-averaged precision increases while micro-averaged recall decreases. The relaxation lift hence seems to trade-off the two values. This observation is somewhat contradictory to the expected outcome of applying the relaxation lift. By preferring rules with longer heads, we also tolerate less precise rules in theory. However, these calculations are done on the training set. Thus, reducing the fit on the training set seems to increase the precision on the test sets but lowers the recall. A reason for the decreased recall might be the condition on the lower boundary of the quality of a rule. Because of this condition, not all occurrences of relevant labels in the training data can be represented by a rule. Only rules with a minimum precision get induced, which in turn increases (micro-averaged) precision. Remember that the number of conditions per rule does also decrease by applying the relaxation lift, which is a sign or a reason for the increased generalization. As a consequence of the increased micro-averaged precision and decreased micro-averaged recall, the micro-averaged f-measure seems to increase slightly on average. Note that we use the f-measure with $\beta = 0.5$ and thus use a more precision-oriented evaluation function. Redoing the experiments with another value for β and a not as strongly precision-oriented f-measure might be interesting. The question is whether or not the observed behaviour is typical for the approach in general or a result of the precision-oriented f-measure. By regarding, the values for hamming and subset accuracy, a general trend of the approach seems likely. However, the average value does not tell the whole story. If we regard the average ranks in Table 17, we can see that the

micro-averaged f-measure increases (the ranks decrease) considerably when using the f-measure as a base learner. Just with hamming accuracy as a base learner, the micro-averaged f-measure plummets – especially when learning only positive heads. This is a trend that can be observed for the other relevant values as well. While hamming and subset accuracy become better, the micro- and macro-averaged values decrease rather significantly when using hamming accuracy as a base learner. Consequently, using the relaxed approach in combination with hamming accuracy as a base learner should mostly be avoided.

Both macro-averaged precision and recall decrease on average. Therefore, the macro-averaged f-measure suffers as well. Again, we regard the ranks for the individual base learners. By doing so, we observe that the macro-averaged f-measure does indeed rise when using the macro-averaged f-measure as a base learner. In the other cases it worsens considerably. The best macro-averaged values are achieved with hamming accuracy and the micro-averaged f-measure as a base learner when only learning positive heads (cf. Table 17). Across all base learners, we can conclude that the micro-averaged f-measure seems to increase slightly, the macro-averaged f-measure seems to suffer. Hamming and subset accuracy benefit most from the relaxed pruning approach.

Base Learner	HA	SA	Mic Prec	Mic Rec	Mic F	Mac Prec	Mac Rec	Mac F
Normal	9.3	9.3	9.2	6.7	8.6	7.8	6.8	7.3
Relaxed	6.1	7.1	5.8	9.2	7.8	8.4	9.2	9.0

Table 18: Average ranks depicted in Table 17 averaged over the evaluation function (excluding subset accuracy).

We furthermore calculate an average rank for each approach in order to determine the respective quality. In this calculation, we omit micro- and macro-averaged precision and recall so that they not distort the ranking. The results can be found in Table 17. The relaxed approach seems to benefit most when using the micro- and macro-averaged f-measure as a base learner. For those, the rank decreases significantly. Moreover, we also show that the relaxed approach does not seem to be suitable for hamming accuracy as a base learner.

Previously, we stated that the relaxed pruning approach should not be utilized for data sets with a label cardinality of almost one. In order to further investigate this statement, we calculate the average ranks for two separate data set groups. The first group consists of data sets with a label cardinality of almost one, which are only two out of the seven tested data sets – BIRDS and SCENE. The second group consists of the remaining five data sets. Average ranks for both groups are depicted in Table 19 and 20, respectively.

Base Learner		HA	SA	Mic Prec	Mic Rec	Mic F	Mac Prec	Mac Rec	Mac F
Mic F	N^+	10.50	9.00	8.00	3.00	2.00	6.00	2.50	2.00
Mic F	R^+	6.50	8.00	5.50	5.00	4.50	3.50	5.50	5.00
Mic F	N^+_-	8.25	10.50	11.25	10.25	9.25	10.25	9.75	9.25
Mic F	R^+_-	7.00	8.25	9.25	10.75	9.75	7.75	10.75	9.75
Mac F	N^+	14.00	13.00	11.50	1.00	10.0	10.50	1.00	8.50
Mac F	R^+	11.00	11.00	8.50	7.50	9.00	9.00	8.00	9.00
Mac F	N^+_-	6.25	9.50	13.25	13.25	13.25	13.25	13.25	13.25
Mac F	R^+_-	4.75	7.50	8.75	11.75	10.75	10.75	11.75	11.25
HA	N^+	12.50	9.00	10.00	3.00	4.50	7.50	3.00	3.00
HA	R^+	2.00	6.00	1.50	11.00	9.50	8.00	11.50	11.00
HA	N^+_-	4.25	6.50	3.50	9.50	9.00	1.50	9.50	8.50
HA	R^+_-	2.00	3.75	2.00	10.00	9.50	9.00	9.50	9.50

Table 19: Average ranks for each approach (excluding subset accuracy) across data sets with a label cardinality of almost one – BIRDS and SCENE.

Base Learner		HA	SA	Mic Prec	Mic Rec	Mic F	Mac Prec	Mac Rec	Mac F
Mic F	N^+	10.00	11.50	11.00	3.60	4.40	6.00	3.20	3.00
Mic F	R^+	5.80	7.40	8.80	4.90	2.60	6.20	5.40	5.00
Mic F	N^+_-	6.40	5.80	5.70	9.40	9.20	7.20	9.80	8.40
Mic F	R^+_-	3.70	6.40	3.00	10.40	7.60	7.00	10.20	9.80
Mac F	N^+	14.40	13.10	14.00	1.00	13.60	10.60	1.00	7.60
Mac F	R^+	9.10	9.10	10.40	5.60	4.40	8.20	5.20	5.00
Mac F	N^+_-	10.00	8.80	8.50	13.40	13.20	13.40	13.30	13.40
Mac F	R^+_-	5.20	3.10	3.70	10.80	8.40	10.40	10.90	11.40
HA	N^+	10.60	8.70	11.20	2.80	3.80	6.60	3.40	2.20
HA	R^+	6.20	6.70	4.10	10.80	10.00	8.60	10.60	10.60
HA	N^+_-	4.40	7.10	4.20	10.20	8.60	2.20	10.40	8.60
HA	R^+_-	7.80	8.80	4.80	12.00	11.00	11.20	11.80	11.60

Table 20: Average ranks for each approach (excluding subset accuracy) across data sets with a higher cardinality than the BIRDS and SCENE data set.

	Mic F				Mac F				HA			
Cardinality	N^+	R^+	N^+_-	R^+_-	N^+	R^+	N^+_-	R^+_-	N^+	R^+	N^+_-	R^+_-
Low	5.88	6.00	9.31	8.69	11.38	10.00	10.56	8.56	7.25	7.13	7.06	6.19
Higher	7.23	5.20	7.45	6.88	12.18	6.90	11.35	7.03	6.33	8.38	7.18	9.80

Table 21: Average ranks for each approach and label cardinality (excluding subset accuracy), calculated by combining the average ranks for hamming and subset accuracy as well as micro- and macro-averaged f-measure.

In order to better compare the values for data sets of a label cardinality of almost one and higher, we again calculate average ranks for each approach (combining hamming and subset accuracy as well as micro- and macro-averaged f-measure). The average ranks can be found in Table 21. For data sets with a cardinality of almost one, the relaxed approach does not seem to perform significantly better. Instead, they seem to perform almost equally good or the relaxed approach is slightly better. Contrary to the average rank across all data sets, the performance with hamming accuracy as a base learner increases minimally. In contrast, hamming accuracy for the relaxed approach is ranked significantly worse than the normal approach for data sets with a higher label cardinality. Comparing both data set groups, we can observe that the data sets with a higher label cardinality benefits considerably more from the relaxed approach. Summing up, utilizing the relaxed approach for data sets with a very low label cardinality should be treated carefully. We cannot definitely decide whether or not we should always advise against using it based on average ranks for two data sets. However, we can see that the performance increase is only very slight and that the data sets with higher label cardinality benefit considerably more from the bias towards multilabel rules. Again, relaxed pruning and hamming accuracy as a base learner seems to be a bad combination.

9.3.1 Performance According To Best Lift Setting

In this section, we regard the predictive performance of the normal and relaxed approach for different lift settings. For that, we calculate a table in which all values are taken from base learners with a lift of one or three percent (cf. Table 22). Each such relaxed pruning approach is then compared to the same base learner and the normal pruning. The second table contains the averaged values for base learners with a lift greater than three percent (cf. Table 23). By separating according to the best lift setting, we can determine whether or not a lower or higher lift value is to be preferred and we might get an insight into the effects of fixing the head and requiring each rule to have at least as many true positives as false positives. The best lift settings for the evaluation are listed in Table 30.

Base Learner		HA	SA	Mic Prec	Mic Rec	Mic F	Mac Prec	Mac Rec	Mac F	Rank
Mic F	N^+	10.33	8.00	9.67	4.00	3.33	5.33	3.33	3.67	6.33
Mic F	R^+	4.83	5.33	6.00	4.67	3.67	3.67	5.00	4.67	4.63
Mic F	N^+_-	6.63	9.00	7.00	10.63	10.13	10.88	11.38	11.13	9.22
Mic F	R^+_-	5.50	8.25	5.38	10.88	9.63	9.88	10.38	9.88	8.31
Mac F	N^+	16.00	14.75	13.50	1.00	12.50	13.50	1.00	10.50	13.44
Mac F	R^+	13.00	12.50	10.50	6.00	8.00	11.50	6.00	7.50	10.25
Mac F	N^+_-	6.90	8.00	9.00	12.50	12.10	12.30	12.20	12.30	9.83
Mac F	R^+_-	5.90	5.30	5.20	11.50	10.30	10.90	11.20	11.10	8.15
HA	N^+	11.40	9.10	11.20	2.80	4.20	8.20	3.40	2.60	6.83
HA	R^+	5.80	6.70	3.70	11.20	10.00	7.60	10.80	10.80	8.33
HA[7]	N^+_-	7.00	8.00	7.00	6.00	5.00	5.00	6.00	6.00	6.50
HA	R^+_-	6.00	8.00	6.00	7.00	6.00	7.00	7.00	7.00	6.75

Table 22: Average ranks for each approach across base learners (evaluation function in combination with positive/negative heads) with a low lift value, i.e. a lift of one or three percent. Furthermore, the combined averaged ranks calculated by combining the average ranks for hamming and subset accuracy as well as micro- and macro-averaged f-measure (in grey).

Base Learner		HA	SA	Mic Prec	Mic Rec	Mic F	Mac Prec	Mac Rec	Mac F	Rank
Mic F	N^+	10.00	12.88	10.50	3.00	4.00	6.50	2.75	2.00	7.22
Mic F	R^+	6.88	9.25	9.25	5.13	2.75	6.75	5.75	5.25	6.03
Mic F	N^+_-	7.33	4.67	7.67	8.33	8.00	4.33	7.67	5.33	6.33
Mic F	R^+_-	3.50	5.17	4.00	10.00	6.33	3.67	10.33	9.67	6.17
Mac F	N^+	13.60	12.40	13.20	1.00	12.60	9.40	1.00	6.80	11.35
Mac F	R^+	8.30	8.50	9.60	6.20	4.80	7.20	6.00	5.60	6.80
Mac F	N^+_-	14.00	11.50	12.00	15.50	16.00	16.00	16.00	16.00	14.38
Mac F	R^+_-	3.00	2.00	5.00	10.00	6.00	9.50	11.00	12.00	6.00
HA	N^+	10.50	8.00	10.00	3.00	3.50	3.50	3.00	2.00	6.00
HA	R^+	3.00	6.00	2.50	10.00	9.50	10.50	11.00	10.50	7.25
HA	N^+_-	3.92	6.75	3.50	10.67	9.33	1.50	10.83	9.00	7.25
HA	R^+_-	6.17	7.25	3.67	12.17	11.33	11.17	11.83	11.67	9.10

Table 23: Average ranks for each approach across base learners (evaluation function in combination with positive/negative heads) with a higher lift value, i.e. a lift of higher than three percent. Furthermore, the combined averaged ranks calculated by combining the average ranks for hamming and subset accuracy as well as micro- and macro-averaged f-measure (in grey).

The combined average ranks (cf. the marked ranks in Table 22 and 23) follow the previously seen trend. When using the f-measure as a base learner, the relaxed approach performs considerably better on average than with hamming accuracy as a base learner. We can further observe that the relaxed approach seems to benefit most when using the f-measure as a base learner and if the best setting is a higher lift value. Especially for the macro-averaged f-measure, the relaxed approach seems to be a good fit. Not only does the rank improve, but also the average ranks are rather good, except for the macro-averaged f-measure. However, the macro-averaged f-measure for learning positive as well as negative heads and using normal pruning always predicts all labels to be irrelevant for data sets for which the best lift setting is high. Nevertheless, the relaxed approach improves upon this and makes better predictions. Again, the macro-averaged values seem to mostly suffer from utilizing the relaxed pruning approach.

The average ranks for the relaxation lift settings with a very low lift value (cf. Table 22) need to be regarded in more detail. While the average combined rank increases with the f-measure as a base learner, the micro- and macro-averaged f-measure value employ a slightly different behaviour. For instance, with the micro-averaged f-measure as a base learner, the micro-averaged f-measure worsens a bit or increases

only minimally. The same observations can be made about the macro-averaged f-measure for this base learner. Hence, the increased subset and hamming accuracy do somewhat distort the ranking in this case. Thus we can say that the f-measure values with the f-measure as a base learner do not differ significantly, but hamming and subset accuracy do. As a consequence, we can conclude that the increase in hamming and subset accuracy might be the result of requiring at least as many true positives as false positives. However, we cannot really be entirely sure about this as the lift still plays a role. Furthermore, precision typically increases and recall typically decreases for both groups.

9.3.2 Subset Accuracy

So far we have left out subset accuracy because some of the experiments did not complete.[8] In this section, we regard the ranks for the approaches with subset accuracy as a base learner. The average and combined average ranks are depicted in Table 24. Note that we only use the ranks for data sets for which both the normal and relaxed approach completed.

Base Learner		HA	SA	Mic Prec	Mic Rec	Mic F	Mac Prec	Mac Rec	Mac F	Rank
SA	N^+	11.50	8.63	10.50	6.00	4.50	6.25	5.75	4.75	7.34
SA	R^+	13.75	8.50	13.00	4.50	7.00	8.25	4.25	3.00	8.06
SA	N^+_-	4.88	4.63	6.50	9.00	7.50	5.75	8.75	8.25	6.31
SA	R^+_-	2.88	4.13	5.25	8.38	4.00	3.75	8.75	7.75	4.69

Table 24: Average ranks for subset accuracy as a base learner across data sets for which the experiments ran through. Furthermore, the combined averaged ranks calculated by combining the average ranks for hamming and subset accuracy as well as micro- and macro-averaged f-measure (in grey).

We can observe that the relaxed approach does not perform well when only learning positive heads. Interestingly, micro- and macro-averaged precision decreases while recall increases. This is in contrast to the other base learners, for which precision and recall behave the opposite way. However, hamming accuracy and the micro-averaged f-measure both worsen, while subset accuracy basically stays the same and the macro-averaged f-measure increases. Summing up, the performance of the relaxed approach when only learning positive heads and using subset accuracy as a base learner seems to drop.

Subset accuracy in combination with also learning negative heads shows a different picture. All values increase, which results in considerably improved ranks. Thus, the relaxed approach seems to be a good fit for this setting, although only tested on four data sets. Furthermore, the combined average rank of 4.69 is the best among all tested approaches (cf. Table 17), followed by the micro-averaged f-measure as a base learner and only learning positive heads. Only the macro-averaged f-measure values for the subset accuracy approach and negative heads does not rank among the best, but is still improved in comparison to the normal pruning approach.

9.4 Rule Set Characteristics

In order to get more insight into the effects of the relaxation lift, we regard several rule sets in this section. At first, we regard two examples in which the application of the relaxation lift resulted in a worse performing model. Then, we regard three examples in which the relaxed pruning approached improved the learned model. For both version, we analyze as to why the relaxed pruning has a positive or negative effect.

[8] The data sets for which the experiments did not complete are BIRDS, MEDICAL, CAL500.

9.4.1 Worsened by Relaxed Pruning

The first rule set is learned by hamming accuracy as a base learner and positive as well as negative heads on the data set YEAST. The rule sets for the normal and relaxed pruning approach are given in Table 25. For the relaxed pruning, the best settings from the evaluation are used. The relaxed approach performs worse in terms of hamming and subset accuracy as well as micro- and macro-averaged f-measure. This base learner combination is additionally the only setting in which a peak label of five and a relaxation lift of greater than 1.01 was learned. The main difference between the data sets seems to be the predictions for Class2 and Class5. While the normal approach sets Class5 to be relevant if Att61 has a certain value, followed by setting the remaining unset label of said class to be irrelevant, the relaxed approach sets Class5 to be irrelevant initially and proceeds to focus on predicting the relevancy of Class5. However, the subsequent rules are useless as they cannot overwrite the irrelevant label for Class5, because they are already set. Nevertheless, the algorithm continues learning similar rules. The reason is most likely the re-inclusion of training examples. Another main difference can be observed by regarding Class2. While the normal pruning approach focuses heavily on this class, Class2 is set to be irrelevant initially as well. As a result, the relaxed approach performs worse. Reducing the peak label to two or three does not lessen this problem.

A likely reason is the refinement process in combination with a rule with several labels in the head. The normal pruning approach learns only single-label heads and is therefore capable of refining these single-label heads more. An example for this is Att61 in the condition of the rule that sets Class5 to be relevant. This is also true for the other classes. On the other hand, the relaxed pruning approach initially learns a rule with three labels in the head. In the refinement process, each possible refinement is tried. As we fix the head, the head may not change. Therefore, refinements are likely to be worse performing because the algorithm cannot find a good refinement for the head with three labels, because no conditions are reasonably applicable to the combination of the three labels. Hence, the rule is not refined and the initial rule with no conditions is learned. Additionally, if a fitting condition exists, it would have likely been learned in the initial search for the next rule, in which all bodies of length one as well as the empty body is tried. As a result, the rule set is too general. The remaining rules with heads of length five can be explained similarly. Due to the bias towards rules with five labels in the head, the learned rules might be the only ones for which a refinement is possible (each rule has at least two refinements) and thus increases the quality of the rule, in comparison to other possible rules. The algorithm continues to learn 354 rules, which later on focus on different labels. Hence, the algorithm terminates. It should be further mentioned that due to the complication with refinements, no label conditions are learned. Furthermore, this problem seems solvable by not fixing the head anymore. Additionally, this may be fixed by applying prepending on the data set.

The second rule set is learned by the micro-averaged f-measure as a base learner and only positive heads on the data set SCENE. The rule set for the normal and relaxed pruning approach are given in Table 26. For the relaxed pruning, the best settings from the evaluation are used. The relaxed approach performs better in terms of hamming and subset accuracy, but worse in terms of micro- and macro-averaged f-measure. By regarding both rule sets, the reason becomes apparent quickly. The relaxed approach stops inducing rules too early because none with as many true positives as false positives are left. In this case, the extra condition on the lower bound of the quality of the rule is not beneficial. The increased hamming accuracy can be accounted to predicting most irrelevant labels correctly, as the not predicted labels are assumed to be irrelevant. Additionally, subset accuracy increases from 7.53% to 19.31%. This result from allowing only the best rules for each label to be learned.

Class14 = 0 (1481, 19)	Class10 = 0, Class11 = 0, Class14 = 0, Class9 = 0 (5538, 462)
Class9 = 0 (1391, 109)	Class12 = 1, Class13 = 1, Class6 = 0, Class7 = 0, Class8 = 0 (5842, 1658)
Class10 = 0 (1341, 159)	Class4 = 0, Class5 = 0, Class1 = 0 (3041, 1459)
Class11 = 0 (1325, 175)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← $Att61_1$ (698, 112)
Class7 = 0 (1241, 259)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att45, Att80 (90, 5)
Class8 = 0 (1211, 289)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att66, Att59 (91 24)
Class6 = 0 (1140, 360)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att45, Att95 (57, 13)
Class12 = 1 (1129, 371)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att79, Att81 (69, 11)
Class13 = 1 (1121, 379)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att62, Att31 (47, 13)
Class5 = 1 ← Att61 (112, 50)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att24, $Att94_1$ (36, 4)
Class5 = 0 (992, 346)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att62, Att44 (17, 3)
Class1 = 1 ← Att88, Class5 = 0 (99, 38)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att80, $Att71_1$ (53, 2)
Class1 = 0 (993, 370)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att66, $Att65_1$ (58, 12)
Class4 = 1 ← Att61 (118, 44)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att62, Att58, Att34 (15, 0)
Class4 = 0 (924, 414)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att79, $Att71_2$ (13, 2)
Class3 = 1 ← $Att35_1$ (104, 49)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att24, Att82 (23, 2)
Class3 = 0 (827, 520)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att62, Att13, $Att94_2$ (15, 0)
Class2 = 1 ← Att9, Class4 = 0 (91, 53)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att79, Att35 (9, 1)
Class2 = 1 ← Att51, Class3 = 0 (78, 50)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att52, $Att61_2$, Att36 (45, 15)
Class2 = 1 ← Att79, Class3 = 0 (66, 46)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att79, Att89 (8, 2)
Class2 = 1 ← $Att35_2$ (69, 54)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att62, Att5 (17, 3)
Class2 = 1 ← $Att33_1$, Att46 (8, 0)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← $Att65_2$', Att51 (33, 7)
Class2 = 1 ← $Att33_2$, Att1 (11, 2)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att79, Att70(5, 0)
Class2 = 1 ← Att34, Att93 (8, 2)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att24, Att101, Att28 (15, 0)
Class2 = 1 ← Att27, Att2 (2, 1)	Class2 = 0, Class10 = 0, Class3 = 0, Class5 = 1, Class14 = 0 ← Att79, Att53 (5, 0)
Class2 = 0 (636, 313)	...

Table 25: Learned models for the data set YEAST, hamming accuracy as a base learner and learning positive as well as negative heads using the normal (left) and the relaxed (right) pruning approach (peak relaxation lift function with peak label five and a lift of 1.13). For the relaxed approach, only a part of the rule set is depicted (continues very similarly). Each rule is annotated with its number of true and false positives (TP, FP). The exact attribute values are abstracted.

Field = 1 ← Att237, Att238, $Att236_1$ (73, 10)	Field = 1 ← Att237, Att238, $Att236_1$ (73, 10)
Sunset = 1 ← Att112, $Att106_1$ (70, 12)	Sunset = 1 ← Att112, Att106 (70, 12)
FallFoliage = 1 ← Att148, Att155 (52, 19)	FallFoliage = 1 ← Att148, Att155 (52, 19)
Beach = 1 ← $Att38_1$ (78, 47)	Beach = 1 ← Att38 (78, 47)
Sunset = 1 ← $Att48_1$, Att43, Att49 (36, 11)	Field = 1 ← $Att236_2$ (56, 52)
Field = 1 ← $Att236_2$ (56, 52)	Mountain = 1 ← Att221 (66, 61)
FallFoliage = 1 ← Att210 (49, 51)	
Mountain = 1 ← Att221 (66, 61)	
Urban = 1 ← Att97 (56, 66)	
FallFoliage = 1 ← Att167 (33, 43)	
Mountain = 1 ← Att99 (47, 53)	
Beach = 1 ← $Att48_2$ (30, 33)	
Sunset = 1 ← $Att106_2$, Att142 (18, 9)	
Urban = 1 ← Att240 (43, 77)	
Mountain = 1 ← Att164 (37, 51)	
Field = 1 ← Att243 (14, 16)	
Beach = 1 ← $Att38_2$ (35, 62)	
FallFoliage = 1 ← Att207 (20, 31)	
Urban = 1 ← Att219 (31, 63)	
...	

Table 26: Learned models for the data set SCENE, micro-averaged f-measure as a base learner and learning positive as well as negative heads using the normal (left) and the relaxed (right) pruning approach (peak relaxation lift function with peak label three and a lift of 1.01). For the normal approach, only a part of the rule set is depicted. Each rule is annotated with its number of true and false positives (TP, FP). The exact attribute values are abstracted.

9.4.2 Improved by Relaxed Pruning

Despite the setbacks mentioned in the previous section, relaxed pruning does improve the learned models relatively often. In this section, we regard three different such rule sets. The first rule set is learned by the macro-averaged f-measure as a base learner and positive as well as negative heads on the data set FLAGS. The rule set for the normal and relaxed pruning approach are given in Table 27. For the relaxed pruning, the best settings from the evaluation are used. The relaxed approach performs better across all evaluation metrics except for recall. By regarding Table 27 we can observe that the normal pruning approach learns a rule with no condition for every label. Each rule predicts the majority class of the label, i.e. if more than 50% of the labels are relevant or irrelevant, it predicts relevant or irrelevant, respectively. Afterwards, the algorithm stops. This rule set does neither reasonably generalize, nor does it help to gain information about the data set. The rule set learned by the relaxed approach performs very well in terms of micro- and macro-averaged f-measure. Additionally, subset and hamming accuracy are improved. Due to a peak label of two and a rather high lift of 1.19 at the peak label, rules with a head length of two are learned exclusively. We can further observe co-occurrences of labels. For instance, *red* and *white* seem to be present in most cases if there are a certain number of colours. This is something that was not evident in the rule set of the normal pruning approach. Similarly, *green* and *yellow* seem to also co-occur. Three rules with the combination of both labels in the head are learned, predicting the same values for both labels. Despite no label conditions being learned, we can gain considerably more information about the data set than with the original algorithm. A possible reason why the relaxed approach does also usually increase subset accuracy is the fact that frequently co-occurring labels are learned and thus set together. If each label is learned individually, both rules might predict something different in different conditions. However, if the combination of the labels is learned, they are set simultaneously and the rule models the occurrence of both labels. Therefore, the chances of setting both labels correctly increases in comparison to when setting each label individually.

The second rule set is learned by the micro-averaged f-measure as a base learner and only positive heads on the data set YEAST. The rule set for the normal and relaxed pruning approach are given in Table 28. For the relaxed pruning, the best settings from the evaluation are used. The relaxed approach performs significantly better in terms of hamming and subset accuracy and achieves the highest micro-averaged f-measure value for this data set. The macro-averaged f-measure values decline a bit, however. While the normal approach learns 148 rules, the relaxed approach only learns 17. It appears that the normal approach learns far too many rules that often have considerably more false positives than true positives. The result is low precision but a very high recall. Due to the condition of requiring at least as many true positives as false positives, the relaxed pruning approach does not learn as many rules. Thus, a more compact model is induced. Additionally, the algorithm learns better rules. They are more precise but do not capture every single relevant label. This is typical for the relaxed pruning approach in combination with the lower boundary for the quality of the rule. We learn more precise rules but do not learn many of the rare occurrences of relevant labels. Hence, recall decreases while precision improves. For the relaxed approach, the KLN relaxation function with a lift of 1.09 at label three is the best performing lift setting in the evaluation. Despite, only four multilabel heads are learned. All have no more than two labels in the head. Hence, although the lift increases for even longer heads, they have too big of a decline in the heuristic value in order to be learned by the algorithm. In such cases, setting the relaxation lift a bit too high does no harm. Note that the rule sets of the relaxed pruning approach do only focus on a few labels. Labels such as Class6 are not considered in the rule set, because the lower boundary for the quality of the rule is not met. Furthermore, the difference to the earlier shown rule set for the data set YEAST should be regarded (cf. Table 25). While the relaxed approach was a bad choice for the data set in combination with hamming accuracy as a base learner and positive as well as negative heads, for the current setting, the relaxed pruning approach yielded far better results and a more compact model.

orange = 0 (109, 20)	black = 0, orange = 0 ← colours ≤ 3.0 (151, 11.0)
red = 1 (104, 25)	red = 1, white = 1 ← colours ≥ 2.5. (166, 38)
black = 0 (98, 31)	green = 1, yellow = 1 ← colours ≥ 3.5 (72, 24)
white = 1 (95, 34)	green = 0, yellow = 0 ← animate = 0 (104, 46)
yellow = 0 (68, 61)	blue = 1, orange = 0 ← colours ≥ 2.5 (90, 60)
blue = 1 (66, 63)	black = 0, blue = 0 ← stripes ≤ 1.0 (34, 8)
green = 0 (65, 64)	black = 1, white = 1 ← population ≤ 252.5 (34, 15)
	blue = 1, red = 1 ← colours ≥ 1.5 (21, 13)
	green = 1, yellow = 1 ←sunstars ≤ 7.0 (7, 3)
	white = 0, red = 0 (3, 0)
	black = 0, yellow = 0 (2, 0)

Table 27: Learned models for the data set FLAGS, macro-averaged f-measure as a base learner and learning positive as well as negative heads using the normal (left) and the relaxed (right) pruning approach (peak relaxation lift function with peak label two and a lift of 1.19). Each rule is annotated with its number of true and false positives (TP, FP).

Class12 = 1 (1129, 371)	Class12 = 1, Class13 = 1 (2250, 750)
Class13 = 1 (1121, 379)	Class4 = 1, Class5 = 1 ← Att61 (230, 94)
Class5 = 1 ← Att61 (112, 50)	Class1 = 1 ← Att88 (104, 45)
Class4 = 1 ← Att61 (118, 44)	Class3 = 1 ← Att35 (104, 49)
Class2 = 1 (656, 844)	Class2 = 1, Class3 = 1 ← Att50_1 (174, 111)
Class1 = 1 ← Att88 (104, 45)	Class4 = 1 ← Att34 (76, 56)
Class3 = 1 (624, 876)	Class5 = 1 ← Att45 (77, 77)
Class4 = 1 ← Att34_1 (76, 56)	Class1 = 1 ← Att94 (69, 61)
Class5 = 1 ← Att45 (77, 77)	Class3 = 1 ← Att64 (78, 61)
Class1 = 1 ← Att94 (69, 61)	Class2 = 1 ← Att52 (83, 68)
Class4 = 1 (338, 868)	Class2 = 1, Class3 = 1 ← Att50_2, Class12 = 1 (163, 135)
Class6 = 1 ← Att56 (58, 74)	Class2 = 1 ← Att66 (61, 57)
Class7 = 1 ← Att96, Att84 (28, 21)	Class2 = 1 ← Att9 (50, 36)
Class8 = 1 ← Att103 (52, 99)	Class3 = 1 ← Att1 (43, 42)
Class6 = 1 ← Att96 (53, 87)	Class2 = 1 ← Att59 (44, 42)
Class1 = 1 (296, 925)	Class4 = 1 ← Att23 (25, 20)
Class7 = 1 ← Att85, Att89_1 (28, 20)	Class3 = 1 ← Att33 (35, 35)
Class8 = 1 ← Class7 = 1 (35, 44)	
Class11 = 1 ← Att60, Att98 (17, 11)	
Class5 = 1 (269, 915)	
Class6 = 1 ← Att25 (40, 71)	
Class8 = 1 ← Att89_2 (42, 102)	
Class7 = 1 ← Class8 = 1 (70, 207)	
Class10 = 1 ← Att35, Att38 (14, 15)	
Class6 = 1 ← Class7 = 1 (61, 191)	
Class11 = 1 ← Att35, Att34_2 (23, 60)	
Class6 = 1 (148, 717)	
...	

Table 28: Learned models for the data set YEAST, micro-averaged f-measure as a base learner and learning only positive heads using the normal (left) and the relaxed (right) pruning approach (KLN relaxation lift function with a lift of 1.09 at label three). For the normal approach, only a part of the rule set is depicted. Each rule is annotated with its number of true and false positives (TP, FP). The exact attribute values are abstracted.

Sunset = 1 ← Att112, Att106 (70, 12)	FallFoliage = 0, Sunset = 1 ← Att112_1 (172, 64)
Field = 1 ← Att237 (90, 28)	Field = 1, Sunset = 0 ← Att237 (200, 36)
Sunset = 1 ← Att116_1, Att111_1 (1, 0)	FallFoliage = 0, Field = 1 ← Sunset = 0 (180, 40)
Sunset = 0 ← Att116_1 (144, 1.0)	Field = 0, Sunset = 1 ← Att112_1 (189, 31)
Sunset = 0 ← Att116_2 (134, 0.0)	FallFoliage = 1, Sunset = 0 ← Att197 (96, 28)
Sunset = 0 ← Att116_3 (131, 0.0)	Sunset = 0 ← Att218_1 (108, 10)
Sunset = 1 ← Att116_4, Att119 (1, 0)	Urban = 0, Sunset = 1 ← Att112_1 (201, 35)
Sunset = 0 ← Att116_4 (123, 0)	Sunset = 0 ← Att218_2, Att212 (42, 1)
Sunset = 1 ← Att116_5, Att132 (1, 0)	Sunset = 0 ← Att218_3, Att224 (51, 1)
Sunset = 0 ← Att116_5 (121, 0)	FallFoliage = 1, Sunset = 0 ← Att117 (77, 51)
Sunset = 1 ← Att116_6, Att17, Att109 (2, 0)	Beach = 0, Sunset = 1 ← Att112_1 (203, 33)
Sunset = 0 ← Att116_6 (100, 3)	Sunset = 0 ← Att48, Att10 (13, 1)
Sunset = 1 ← Att116_7, Att44 (8, 2)	Sunset = 0 ← Att218_4, Att183 (23, 0)
Sunset = 1 ← Att116_8, Att18 (6, 1)	Sunset = 0 ← Att218_5, Att112_2 (21, 1)
Sunset = 1 ← Att116_8, Att69_1, Att214 (4, 0)	Sunset = 0 ← Att218_6, Att261 (20, 1)
Sunset = 1 ← Att116_8, Att48, Att193 (2, 0)	FallFoliage = 1, Field = 0 ← FallFoliage = 1 (203, 49)
Sunset = 1 ← Att116_8, Att115_1, Att185 (1, 0)	Sunset = 0 ← Att219, Att223 (17, 2)
Sunset = 0 ← Att116_8, Att115_1 (34, 0)	Sunset = 0 ← Att49, Att192 (8, 0)
Sunset = 1 ← Att116_7, Att111_2, Att69_2 (3, 0)	FallFoliage = 1, Sunset = 0 ← Att125 (35, 21)
Sunset = 1 ← Att116_7, Att117 (1, 0)	FallFoliage = 1, Sunset = 0 ← Att201 (22, 2)
Sunset = 1 ← Att116_7, Att130 (1, 0)	Sunset = 0 ← Att45, Att194 (6, 0)
Sunset = 1 ← Att116_7, Att115_2, Att111_2 (1, 0)	Field = 1, Sunset = 0 ← Att236 (110, 28)
Sunset = 0 ← Att116_7, Att115_2 (42, 0)	Field = 1, Urban = 0 ← Field = 1 (312, 46)
Sunset = 1 ← Att116_9, Att30 (14, 1)	Field = 1, Beach = 0 ← Field = 1 (310, 48)
Sunset = 1 ← Att116_9, Att260 (7, 0)	FallFoliage = 0, Sunset = 1 ← Att109 (15, 5)
Sunset = 1 ← Att116_9, Att47 (8, 3)	Sunset = 0 ← Att41, Att277 (6, 0)
FallFoliage = 1 ← Att148, Att155 (52, 19)	Sunset = 0 ← Att99, Att102_1 (3, 0)
Sunset = 1 ← Att116_9, Att175 (2, 0)	Beach = 1, Sunset = 0 ← Att38 (176, 32)
Sunset = 1 ← Att116_9, Att221 (1, 0)	Field = 0 ← Att214, Att215 (56, 4)
Sunset = 0 ← Att116_9, Att107 (31, 0)	Sunset = 0 ← Att45, Att184 (4, 2)
...	...

Table 29: Learned models for the data set SCENE, example-based subset accuracy as a base learner and learning positive as well as negative heads using the normal (left) and the relaxed (right) pruning approach (peak relaxation lift function with peak label two and a lift of 1.19). For both approaches, only a part of the rule set is depicted. Each rule is annotated with its number of true and false positives (TP, FP). The exact attribute values are abstracted.

The third and last rule set is learned by example-based subset accuracy as a base learner and positive as well as negative heads on the data set SCENE. The rule sets for the normal and relaxed pruning approach are given in Table 29. For the relaxed pruning, the best settings from the evaluation are used. The relaxed approach performs considerably better in terms of subset accuracy and slightly better for the remaining evaluation metrics. As the normal approach learns 982 rules and the relaxed approach learns 551 rules, only the first parts of the rule sets are depicted. Something that catches the eye immediately is the early focus on the *Sunset* label by the normal approach. In contrast to that, the relaxed approach seems to learn a more diverse rule set initially. We do also observe that for the depicted part, the normal approach does not learn any label conditions. In this category, the relaxed approach does far better. Additionally, we learn a good amount of multilabel rules, even for some global label dependencies. For this data set, we can observe that just regarding the percentage of label conditions does not tell the whole story. For the complete rule set, the normal approach learns 3.18% label conditions, while the relaxed approach learns only 2.40% label conditions. Albeit the decrease in the percentage of label conditions, the relaxed approach seems to learn more effective label conditions. Implying, the label-dependent rules are learned earlier in the decision list and thus have more of an impact. Already the third rule is a global label dependency in combination with a multilabel head. This is enabled by the increased coverage of previous rules. For the normal approach, fewer labels are set that can be used for the induction of label-dependent rules. Due to the label dependencies and the multilabel head rules, we

can gather more information about the data set using the relaxed pruning approach. Let us just consider the label-dependent rules, which are

$$\text{FallFoliage} = 0, \text{Field} = 1 \leftarrow \text{Sunset} = 0$$
$$\text{FallFoliage} = 1, \text{Field} = 0 \leftarrow \text{FallFoliage} = 1$$
$$\text{Field} = 1, \text{Urban} = 0 \leftarrow \text{Field} = 1$$
$$\text{Field} = 1, \text{Beach} = 0 \leftarrow \text{Field} = 1$$

Just by regarding these rules, we can conclude that if a scene does not contain as sunset, it does not contain fall foliage, but it likely depicts a field. Furthermore, the second label-dependent rule states that if a scene contains fall foliage, it is unlikely to be a field. Hence, both labels seem to exclude each other. Furthermore, if a scene contains a field, it is unlikely to contain an urban area or a beach. These statements all make sense if thinking about such pictures. Typically, beach pictures are not adjacent to fields and urban areas are not typically contained in pictures of fields. Furthermore, field pictures do typically not contain fall foliage and are often taken in the summer. Additionally, fall foliage pictures are often taken in the afternoon or evening, when the sun is setting. Thus, we can conclude that if no sunset is present, the picture is unlikely to contain fall foliage. We can see that despite only four regarded rules, we can gain a lot of information about the data set SCENE and can use it in the prediction of the labels. Note that we must consider that most previous covered examples are removed from the training set, i.e. that the learned label dependencies are not applicable globally but only after the application of all previous rules. Although we do not regard the remaining rules that are not depicted Table 29, we can expect similar expressiveness for the respective data set.

9.5 Discussion

In the previous sections we have evaluated the relaxed approach and compared it to the original algorithm. We have further regarded exemplary rule sets for cases in which the relaxed pruning lead to a worse and a better rule set. Thus, we have covered the advantages and disadvantages of the approach. In most cases, using the relaxed approach seems to be an improvement. Additionally, relaxed pruning seems to especially benefit from the application on data sets for which the label cardinality is not too low. Furthermore, the relaxed pruning approach seems to handle data sets, for which the normal pruning predicts the majority classes, better. The results obtained in the evaluation are typically in line with what we have observed in the sensitivity analysis.

9.6 Determining Good Lift Values

In this section, we take a look at what good lift settings and values are. Recall that we restricted ourselves to only evaluating the peak and the KLN relaxation lift function. We further had to restrict the possible relaxation lift values and omitted a peak label of four in the evaluation. The relaxation lift settings that reach the best performance for each approach and data sets are listed in Table 30. Note that, as previously described, we choose the lower lift in case of a tie performance-wise, which occurred rather often.

Base Learner		CAL500	EMOTIONS	MEDICAL	FLAGS	SCENE	BIRDS	YEAST
Mic F	+	(PK, 5.0, 1.01)	(PK, 3.0, 1.13)	(PK, 5.0, 1.01)	(LN, 3.0, 1.13)	(PK, 3.0, 1.01)	(PK, 3.0, 1.07)	(LN, 3.0, 1.09)
Mac F	+	(PK, 3.0, 1.13)	(PK, 3.0, 1.01)	(LN, 3.0, 1.07)	(LN, 3.0, 1.15)	(PK, 5.0, 1.01)	(PK, 5.0, 1.19)	(LN, 3.0, 1.19)
HA	+	(PK, 2.0, 1.17)	(PK, 2.0, 1.03)	(PK, 5.0, 1.01)	(PK, 3.0, 1.03)	(LN, 3.0, 1.03)	(LN, 3.0, 1.05)	(PK, 5.0, 1.01)
SA	+	-	(PK, 2.0, 1.19)	-	(PK, 2.0, 1.11)	(PK, 5.0, 1.01)	-	(LN, 3.0, 1.03)
Mic F	+-	(PK, 5.0, 1.01)	(PK, 2.0, 1.13)	(PK, 2.0, 1.03)	(PK, 3.0, 1.19)	(PK, 2.0, 1.01)	(PK, 2.0, 1.17)	(PK, 3.0, 1.01)
Mac F	+-	(PK, 5.0, 1.01)	(PK, 3.0, 1.15)	(PK, 2.0, 1.03)	(PK, 2.0, 1.19)	(PK, 2.0, 1.01)	(PK, 3.0, 1.03)	(PK, 5.0, 1.01)
HA	+-	(LN, 3.0, 1.01)	(PK, 3.0, 1.09)	(LN, 3.0, 1.19)	(PK, 2.0, 1.19)	(LN, 3.0, 1.19)	(LN, 3.0, 1.07)	(PK, 5.0, 1.13)
SA	+-	-	(PK, 2.0, 1.19)	-	(PK, 2.0, 1.03)	(PK, 2.0, 1.19)	-	(PK, 3.0, 1.01)

Table 30: The best lift values for each relaxed pruning approach, attained in the evaluation. Each lift settings is denoted by the relaxation lift function (PK for peak and LN for KLN), a label (in the case of the peak lift function the peak label) and the relaxation lift at said (head length).

Something that leaps to the eye immediately is the frequency of the peak relaxation lift function. In most cases, the peak relaxation lift functions seems to result in the best performance. In contrast, the KLN relaxation function is the best performing function in far fewer cases. For the data set SCENE we can observe that the lift is almost zero, except with hamming and subset accuracy as a base learner and learning negative heads. This is most likely a result of the low label cardinality 1.074 of the data set. Contrary to that, for the data set BIRDS this is not the case, although the label cardinality is with 1.014 even lower. Moreover, peak label five does only occur once with a relaxation lift of greater than one percent (for the YEAST data set). In the other cases, peak label five in combination with a lift of one percent is the minimum possible relaxation lift. The lift is lower than peak label three and a lift of one percent for instance, because the same lift is normalized over a wider span. However, the difference in the lift is so little in such cases that it does not matter. Surprisingly, similar lift values are rather frequent, even for data sets with a higher label cardinality. In cases such as this it is likely that a higher lift value achieved identical results. The exception is CAL500, which does not need much lifting as the label cardinality is very high and thus enough multilabel rules are already predicted. In this case it is more important to not predict too many labels with a single rule. A relaxation lift of 19 percent also occurs frequently, which is the maximum possible lift value. In these cases, it might be interesting whether or not a higher lift would have been chosen. However, a higher lift for subsequent head lengths is possible with the KLN relaxation lift function, which was not chosen in such cases. Often, the best lift value seems to be between 10 and 20 percent. We can also conclude that data sets with a very high label cardinality do not need to be lifted as much as others. Furthermore, it seems likely that data sets with a very low cardinality also require less of a lift than others. For the remaining data sets, we can advise using the peak relaxation lift function and a lift between 10 and 20 percent. Although, for the data set MEDICAL, which has a low label cardinality of 1.245, the lift values are also decreased in comparison to the others. However, the lift is still considerable. For such data sets, a lift of between three and seven percent appears to be working well. Whether or not to use peak label two or three remains unclear. For the data sets EMOTIONS and FLAGS for instance, both peak labels appear in 50% of the cases. Nevertheless, both seem to be working well and which to choose can also be dependent upon the preferences of the user.

9.7 Prepending

The evaluation of the prepending approach presented in Section 8 is done on a smaller scale and is only supposed to show the functionality of the approach. Hence, we only execute the experiments for a single base learner and a single relaxation lift setting. Nevertheless, models for all seven data sets are learned. For the relaxation lift we utilize the peak relaxation lift function with a peak label of four and a relaxation lift of 1.10 at the peak label. This setting was chosen as it only lifts the label search space a bit and can be used for all data sets. Setting the peak label to four was done in order to not prevent heads of longer length from being learned. Still, if they result in too big of a decline in the heuristic value, such rules are not learned. Furthermore, the peak label was chosen higher than usual as we aim at learning positive and negative heads. While prepending is also capable of utilizing only positive heads, it makes more sense to allow negative heads so that we can correct labels that were mistakenly believed to be relevant. For the remaining parameters, we use the same settings as in the evaluation of the relaxed pruning approach in Section 9.1. Moreover, we restrict the evaluation to only using the micro-averaged f-measure as a base learner in order to show the functionality of prepending.

9.7.1 Performance and Model Characteristics

Table 31 lists the performance of the prepending approach on the training set for all seven data sets and the described setup. We can observe that the approach overfits the training set in most cases. Due to prepending neither having a stopping criterion, nor being capable of inducing stopping rules, the algorithm only terminates if no more rules can be learned. This is the case if there are no more rules

that satisfy the lower boundary on the quality of the rule, as described in Section 8.4. Recall that for prepending, the condition requires a strictly greater number of true positives in order to prevent loops. Thus, the approach typically induces far too many rules (cf. Table 33) and corrects the values too often.

Data Set	HA	SA	Mic Prec	Mic Rec	Mic F	Mac Prec	Mac Rec	Mac F
CAL500	85.89	0.00	52.44	65.97	58.43	70.09	79.24	71.04
EMOTIONS	86.49	46.80	75.52	81.81	78.54	88.45	81.70	80.43
MEDICAL	99.86	93.99	100	94.98	97.42	95.56	91.03	92.53
FLAGS	89.04	58.91	88.34	89.34	88.84	88.52	84.16	85.00
SCENE	87.90	34.35	89.06	36.08	51.36	73.56	39.06	49.33
BIRDS	99.53	92.55	100	91.50	95.56	100	88.60	93.38
YEAST	79.29	11.73	70.90	53.61	61.05	41.79	29.17	30.32

Table 31: Evaluation results on the training set for each data set while using Prepending.

Despite the overfitting, the performance on the test data is relatively good for most data sets. Table 32 shows the performance of the learned models on the test sets. In order to better compare the values, hamming and subset accuracy as well as the f-measure values are annotated with the rank they would achieve in the evaluation of the relaxed approach in Section 9.3. We can see that prepending yields rather good results, except for the data set EMOTIONS. Especially considering the parameters were not optimized (including the relaxation lift). Thus, we can conclude that despite still requiring a stopping criterion, prepending seems to be a promising approach in terms of predictive performance in comparison to the other approaches.

Data Set	HA		SA		Mic Prec	Mic Rec	Mic F		Mac Prec	Mac Rec	Mac F	
CAL500	75.18	11	0.00	8	25.91	36.11	30.17	12	14.73	20.88	15.52	4
EMOTIONS	69.31	14	13.86	12	53.27	55.14	54.19	12	54.16	54.36	52.12	11
MEDICAL	98.52	7	52.25	8	74.18	70.75	72.42	6	40.74	41.05	40.59	7
FLAGS	73.41	5	23.08	1	72.22	71.89	72.07	7	64.62	63.99	61.79	9
SCENE	84.95	1	23.91	4	72.30	27.33	39.66	7	57.99	27.39	36.14	7
BIRDS	94.12	7	41.80	7	41.55	37.70	39.53	4	34.63	30.84	30.92	2
YEAST	77.52	5	7.20	7	66.86	50.88	57.78	6	32.40	27.22	27.47	10

Table 32: Evaluation results on the test set for each data set while using Prepending.

Data Set	Normal	Relaxed	Prepending
CAL500	0.00	100	51.66
EMOTIONS	3.08	0.00	0.52
MEDICAL	9.04	10.84	0.00
FLAGS	3.41	0.00	4.08
SCENE	0.00	0.00	18.18
BIRDS	0.50	1.52	1.23
YEAST	0.00	0.00	16.67

Data Set	#Rules	%MHR
CAL500	2486	10.50%
EMOTIONS	73	12.33%
MEDICAL	116	71.55%
FLAGS	41	9.76%
SCENE	41	18.15%
BIRDS	137	44.53%
YEAST	27	25.93%

Table 33: The percentage of label conditions for the learned models and approaches (left) as well as the number of learned rules and percentage of multilabel (head) rules (right).

Table 33 depicts the percentage of label conditions for each data set and approach as well as the number of rules and percentage of multilabel rules. In most cases, prepending seems to improve the percentage of label conditions learned in comparison to the relaxed approach. Note that the 100% for CAL500 and the relaxed pruning approach comes from only a single condition being a label condition and must therefore be treated carefully. For the data set MEDICAL no label conditions are learned with prepending, contrary to the relaxed pruning approach. Similarly, for the data set EMOTIONS the percentage of label conditions could only be improved minimally. All models contain a considerable number of

multilabel rules, despite setting the relaxation lift not too high. Again, we can observe the overfitting and the need for a stopping criterion by regarding the number of rules learned for the data set CAL500.

Summing up, prepending seems to have a positive effect on the percentage of label conditions learned. Furthermore, the predictive quality is already acceptable, but might improve further by implementing a stopping criterion and the induction of stopping rules. Additionally, a good number of multilabel rules seem to be learned by the approach.

9.7.2 Rule Set Characteristics

Although the predictive quality is acceptable, regarding the learned rule sets is crucial in understanding the proposed approach. Only then can we see how well prepending learns rules and exploits label dependencies. Hence, we look at two different rule sets that were learned in the evaluation.

The first rule set was learned on the data set FLAGS and is depicted in Table 34. Not too many multilabel rules are learned, which is a cause of the rather low relaxation lift applied. The first few rules set the labels to be relevant. This happens as we set all labels to be irrelevant initially. After some time, the prediction of negative heads begins. The prediction of negative heads becomes possible as we always make some mistakes when predicting labels to be relevant. The false positives indicate the number of predictions made that distorted correct values. At some point, the algorithm starts to correct those values. The further down in the decision list, the less values get corrected. Sometimes, a label condition is learned. However, only one global dependency yellow = 1 ← white = 0 is learned. At first, the fact that such label dependencies only correct very few examples seems problematic. Despite, the label dependency is useful. The important point is that while it may correct only a few values, it does not distort many values either. If the rule or label dependency would not be valid, it would distort many values and hence not get learned. Thus, as we will see in the other rule set as well, label dependencies tend to get learned in the later stages of the decision list. This is something that must also be considered in the implementation of a stopping criterion and applies to other rules without labels in the body as well. The learned rules depicted in Table 34 might also be easier to understand as they directly depend on the prediction of all previous rules and mostly model exceptions to previous rules.

The second rule set was learned on the data set SCENE. For this rule set we can observe the same characteristic in regard to the label conditions as previously mentioned. The last twelve rules all model label dependencies. This stands in contrast to the rule set for SCENE that we regarded earlier (cf. Table 29). Using prepending, the rules become more influential the later in the decision list they are learned as they may overwrite predictions of previous rules. They have the last word so to speak. Furthermore, exceptions to label dependencies can be learned far better. For instance, a rule Sunset = 0 ← FallFoliage = 1 is learned. The next rule is a rule that models the exceptions to this label dependency – Sunset = 1 ← FallFoliage = 1, Att50. Implying that if *FallFoliage* is predicted and Att50 has a certain value, the previous prediction is overwritten and the scene actually contains a sunset. Interestingly, the former rule does not distort any values. Hence, the latter rule corrects a value that was set by another rule. Note that such label-dependent rules are only learned if they still correct a value. In the case that a label dependency exists, but cannot be used anymore to correct some of the earlier predictions, the corresponding rules are not learned. By applying prepending, a lot of such dependencies between labels are learned and may be used to improve the understanding of the data set at hand.

Furthermore, applying too high of a lift to the label search space is not as problematic as for the original relaxed pruning approach as values can be corrected later on. Thus, the chances of choosing a bad relaxation lift setting that results in a bad model reduces.

red = 1 ← colours ≥ 2.5, area ≤ 2641.5, bars ≤ 3.0, crescent = 0 (85, 9)
white = 1 ← area ≤ 245.5, population ≤ 56.0, sunstars ≤ 5.0, bars ≤ 0.0 (58, 7)
green = 1, yellow = 1 ← colours ≥ 3.5, population ≤ 38.5, area ≤ 1607.0 (66, 18)
green = 1 ← landmass = 4, stripes ≤ 3.0, circles ≤ 0.0 (17, 2)
orange = 1 ← colours ≥ 6.5 (6, 0)
blue = 1, white = 1 ← colours ≥ 2.5, bars ≤ 0.0 (72, 40)
red = 1 ← stripes ≤ 2.0, colours ≤ 3.0, crescent = 0, bars ≤ 2.0, colours ≥ 2.0 (14, 2)
orange = 1 ← sunstars ≤ 0.0, colours ≥ 2.0, circles ≤ 0.0, animate = 1, text = 0, colours ≥ 4.0 (5, 0)
white = 1 ← sunstars ≤ 0.0, colours ≥ 2.0, circles ≤ 0.0. (15, 5)
orange = 1 ← religion = 4. (2, 0)
black = 1 ← religion = 2, stripes ≥ 3.0, colours ≥ 4.0 (8, 0)
blue = 0, yellow = 0 ← religion = 2 (16, 2)
white = 0 ← landmass = 4, area ≥ 245.5, stripes ≤ 3.0, colours ≥ 3.0, area ≤ 1938.5 (9, 2)
yellow = 1 ← white = 0 (7, 2)
blue = 0 ← landmass = 4, population ≥ 2.0 (8, 2)
yellow = 1 ← religion = 6, sunstars ≥ 1.0, stripes ≤ 0.0 (5, 0)
green = 1 ← language = 6, population ≥ 28.0 (5, 0)
green = 0 ← landmass = 6, colours ≤ 5.0 (3, 0)
yellow = 1 ← landmass = 2, blue = 1, quarters ≤ 0.0 (4, 0)
blue = 1 ← language = 2, population ≤ 36.5 (4, 1)
yellow = 1, white = 0 ← language = 4, population ≥ 8.0 (4, 0)
black = 1 ← colours ≥ 6.5, stripes ≥ 2.0 (3, 0)
black = 1 ← language = 4, population ≥ 8.5 (2, 0)
orange = 1 ← circles ≥ 0.5, white = 0 (1, 0)
orange = 1 ← circles ≥ 0.5, population ≤ 8.5, language = 7 (1, 0)
red = 1 ← circles ≥ 0.5, population ≤ 28.0 (2, 0)
red = 1 ← area ≥ 8937.5 (1, 0)
blue = 0 ← language = 4, population ≥ 5.0 (2, 0)
green = 0 ← landmass = 3, colours ≤ 4.0 (2, 0)
green = 1 ← language = 9, circles ≤ 0.0 (1, 0)
white = 1 ← language = 9 (1, 0)
blue = 0 ← language = 9 (1, 0)
blue = 1 ← bars ≥ 4.0 (1, 0)
blue = 1 ← stripes ≥ 8.0 (1, 0)
orange = 1 ← landmass = 3, bars ≥ 3.0, area ≤ 268.0 (1, 0)
green = 1 ← orange = 1, bars ≥ 3.0. (2, 0)
blue = 0 ← landmass = 3, area ≤ 32.5, crosses ≤ 0.0, stripes ≤ 2.0 (3, 0)
yellow = 1 ← landmass = 3, icon = 1, bars ≤ 0.0 (2, 0)
black = 1 ← landmass = 3, area ≤ 0.0, stripes ≤ 0.0 (2, 0)
red = 1 ← landmass = 3, population ≥ 18.0 (1, 0)
blue = 1 ← circles ≥ 1.5 (1, 0)

Table 34: Learned model for the data set FLAGS while using prepending. Each rule is annotated with its number of true positives and false positives (TP, FP), which have a different semantics in the case of prepending (cf. Section 8.2).

Field = 1 ← Att237, Att238, Att236_1' (73, 10)	Field = 1 ← Att118, Att244 (9, 3)
Sunset = 1 ← Att112, Att106 (70, 12)	Sunset = 0 ← Att118 (3, 0)
FallFoliage = 1 ← Att148, Att155 (52, 19)	Field = 1 ← Att125, Att229, Att2 (6, 1)
Beach = 1 ← Att38 (78, 47)	Sunset = 0 ← Att125 (3, 0)
Field = 1 ← Att236_2, Att235 (41, 25)	FallFoliage = 1 ← Att233_2, Att167 (12, 9)
Field = 0 ← Att200 (16, 2)	Field = 0 ← Att233_2, Att124 (4, 0)
Mountain = 1 ← Att221 (66, 61)	FallFoliage = 0 ← Sunset = 1, Att136 (7, 0)
Mountain = 0 ← Att207 (35, 12)	Beach = 1 ← Att8, Att48_2 (10, 2)
Field = 1 ← Att233_1, Att241, Att240 (13, 4)	FallFoliage = 0 ← FallFoliage = 1, Att110 (5, 1)
Field = 0 ← Att147 (11, 3)	Sunset = 0 ← FallFoliage = 1 (1, 0)
Beach = 0 ← Att233_1 (18, 0)	Sunset = 1 ← FallFoliage = 1, Att50 (1, 0)
Sunset = 0 ← FallFoliage = 1, Att154 (9, 4)	FallFoliage = 0 ← FallFoliage = 1, Att250 (3, 0)
Sunset = 1 ← Att48_1, Att43, Att49, Att47 (37, 10)	Field = 0 ← FallFoliage = 1, Att68 (1, 0)
Sunset = 1 ← Att12, Att111 (6, 1)	Sunset = 0 ← Sunset = 1, Att56, Att210 (2, 0)
Sunset = 1 ← Att31, Att107, Att4 (6, 1)	Sunset = 0 ← Sunset = 1, Att44, Att90 (1, 0)
Sunset = 0 ← Att201, Att6 (5, 0)	Beach = 0 ← Sunset = 1 (2, 0)
Sunset = 1 ← Att1, Att136 (6, 1)	FallFoliage = 0 ← FallFoliage = 1, Att212 (2, 0)
Sunset = 0 ← Att64 (4, 1)	Beach = 0 ← FallFoliage = 1 (2, 0)
FallFoliage = 0 ← Att1, Att149 (10, 2)	FallFoliage = 0 ← FallFoliage = 1, Att60 (1, 0)
Field = 0 ← Att83, Att234 (6, 1)	Mountain = 0 ← FallFoliage = 1 (1, 0)
Sunset = 1 ← Att8, Att219 (4, 3)	

Table 35: Learned model for the data set SCENE while using prepending (first part left, second part on the right). Each rule is annotated with its number of true positives and false positives (TP, FP), which have a different semantics in the case of prepending (cf. Section 8.2).

The learned rules for the data set CAL500 seem to especially exploit hierarchical dependencies in the labels, such as

$$\text{Genre-Rock} = 1 \leftarrow \text{Genre-Best-Rock} = 1$$
$$\text{Genre-Folk} = 1 \leftarrow \text{Genre-Best-Folk} = 1$$

Interestingly, such rules are not only learned once, but the same rule is also learned several times. Due to some other rules being applied after the initial learning of such a rule, some values get distorted. Hence, as the label dependency is globally applicable, the algorithm learns the same rule again to correct the distorted values again. Furthermore, several labels are found that exclude each other, such as

$$\text{NOT-Song-Very-Danceable} = 0 \leftarrow \text{Song-Very-Danceable} = 1$$
$$\text{NOT-Emotion-Sad} = 0 \leftarrow \text{Emotion-Sad} = 1$$

While this relationship is obvious due to the name, the algorithm cannot make the same connection. Additionally, if the labels were named differently, the algorithm would still reveal the relationship and thus exploit it. Other selected global label dependencies include

$$\text{Genre-RandB} = 1, \text{Genre-Hip-Hop-Rap} = 0 \leftarrow \text{Genre-Best-RandB} = 1$$
$$\text{Genre-Folk} = 1, \text{Genre-Singer-Songwriter} = 1 \leftarrow \text{Genre-Alternative-Folk} = 1$$
$$\text{Genre-Contemporary-RandB} = 1 \leftarrow \text{Genre-Best-RandB} = 1$$
$$\text{Angry-Agressive} = 1 \leftarrow \text{Vocals-Aggressive} = 1$$

Local dependencies are omitted on purpose because they are not easily interpretable. Nevertheless, we can conclude that a lot of effective label dependencies are learned for the shown data sets. For the data set EMOTIONS for instance, no label dependencies are learned. The reason seems to be that the learned rules that set the labels to be relevant are already good enough and rarely make mistakes. Hence, the values do not need much correcting and label dependencies are not required.

10 Conclusion

In this final chapter, a conclusion in the form a summary of the previous chapters and possible future improvements of the presented algorithms is given.

10.1 Summary

The algorithms in this work are based on the algorithm proposed by Rapp [2016], which is capable of learning multilabel rules by searching for the best performing label combination for the head of a given body. Hence, most parts regarding the general learning process, such as the multilabel decision list and re-inclusion of covered examples, stay the same. What changes is the refinement of rules and the search for the best performing head. For those, we opt for fixing the head during the rule refinement process and searching for the head with the highest lifted heuristic value. Furthermore, we limit the quality of a rule by requiring at least as many true positives as false positives. In order to enable the relaxation of pruning, we introduce three different relaxation lift functions – the peak relaxation lift function, the KLN relaxation lift function and the root relaxation lift function. All three enable an easier handling of the definition of the lift of the label search space. As a result of introducing this bias towards longer heads, the original algorithms for exploiting anti-monotonicity and decomposability of multilabel evaluation metrics, proposed by Rapp et al. [2018], cannot be used to guarantee the finding of the best performing lifted head, which was illustrated by giving counter-examples. Therefore, both algorithms needed to be adapted. In the sensitivity analysis, we studied the effects of the made changes. Thus, we executed the algorithm for different relaxation lift settings and compared the results for the KLN and peak relaxation lift function. We further took a look at the fit on the training set of the relaxed pruning algorithm and observed that a higher lift results in a more general model. This seems to be the result of fixing the head in combination with longer heads. In addition to the relaxed pruning, we also suggest an alternative algorithm that uses the relaxed pruning approach – prepending. For prepending, several impactful changes to the previously shown algorithm had to be made. Most notably, we changed the semantics of true positives, false positives, false negatives and true negatives so that we can correct previous predictions. Furthermore, we allow the overwriting of values and initially treat missing values as zeroes. Additionally, due to the fact that fully covered examples may still be corrected, we cannot rely on coverage as a stopping criterion.

We evaluated the relaxed pruning approach by comparing it to the original algorithm by Rapp [2016]. We observe that the relaxed pruning approach seems to be an improvement when using the micro- or macro-averaged f-measure or subset accuracy as a base learner. For hamming accuracy as a base learner, the algorithm does not seem to work well. In general, we can say that (micro-averaged) precision mostly increases and recall decreases in most cases. The reason most likely being the imposed limit on the quality of a rule. Moreover, run time seems to increase in most cases as the rule learning process does not have to be executed as often. Often, relaxed pruning seems to induce a more expressive model. We furthermore identify the peak relaxation lift function as the most promising relaxation lift function. Depending on the label cardinality of the data set and whether or not learning only positive or positive and negative head rules, the suggestion for the best lift setting differs. In most cases, the peak relaxation lift function with peak label two or three with a lift between 1.10 and 1.20 seems to work well. For data sets with a low label cardinality, a lift between 1.03 and 1.07 is more promising. Nevertheless, in some cases the percentage of label conditions increases but often it does not. Moreover, we have presented positive and negative examples for the made changes, i.e. the fixing of the head and the limit on the quality of the rule. Finally, we briefly evaluate prepending by executing it on the seven tested data sets. We mostly observe an overfit on the training set. Despite the overfit, the predictive performance seems promising. Especially when considering that parameters were not optimized and the algorithm does not have a stopping criterion, nor stopping rules. Additionally, more label dependencies seem to get learned.

10.2 Future Work

Despite the studied effects and the, in most cases, improved performance of the relaxed pruning approach and the other changes to the algorithm, in some cases it remains unclear as to what changes specifically imply which effects. Therefore, it might be beneficial to evaluate the relaxed pruning approach individually without the made changes. Similarly, a more extensive evaluation might yield more precise results. Due to restrictions in run time and resources, only a few settings could be evaluated. For that, it might also make sense to improve the general efficiency of the algorithm. Furthermore, a different strategy for fixing the head could be employed. As a result of the fixing, rules are difficult to refine. Therefore, changes to the refinement process might be beneficial. For instance, we could only allow the number of labels in the head to increase. Another approach could be to only allow fewer labels in the head if the performance significantly increases. These adjustments could solve the problems that occurred by fixing the head. Similarly, the way in which conditions are learned sometimes seems problematic. For instance, exceptions or generally good refinements might not be found because we only try bodies of length one initially. Additionally, as already stated by Rapp [2016], the usage of a beam search should be investigated in that regard. When performing a beam search, we keep track of the best β rules instead of only the best. Note that in general, this has a negative impact on the computational complexity. However, if fixing the head during the refinement, the negative impact is not as significant.

Another aspect for future work is the pruning. While we have given counter-examples for pruning the lifted heuristic value with anti-monotonicity and decomposability and stated that this suboptimal pruning can also occur for longer depths, i.e. the lifted heuristic value decreases for a certain number of times before reaching its peak, we have not formally proven this. Therefore, more investigation in this regard might allow slightly heavier pruning, especially for anti-monotonicity. For this, it might also make sense to exploit certain constraints or characteristics of the relaxation lift functions. For instance, that the additional lift decreases with more labels, i.e. the function flattens.

While some of these changes also apply to prepending, prepending still requires more work. Most notably, the approach is in need of a stopping criterion. Otherwise, the algorithm corrects values too often. However, the fact that label dependencies seem to be learned towards the end of the decision list needs to be considered. Additionally, prepending is currently not capable of inducing stopping rules, which might also prove useful. For the stopping criterion, different approaches should be considered. An example for a criterion could be to only allow each value to be correct a certain number of times or that rules must cover enough examples. Note that this does not refer to the true positives, which denote the number of corrections a rule performs. We have also observed that prepending only learns label dependencies if they can be used to correct some of the values. If that is not the case, no such dependencies are learned. Therefore, it might be possible to systematically or purposefully make same mistakes during the learning process. However, it is likely that the value is simply corrected by a rule with a normal condition. While not specifically tested for prepending, a lifting of rules with label conditions could be performed. For the relaxed pruning algorithm, this did not seem to be beneficial, however, as only such rules were learned. Last but not least, a more thorough evaluation of the prepending approach is required. In this regard, other averaging strategies may also be explored as in this work we only use micro-averaging. Additionally, prepending was only used in combination with a decomposable evaluation metric. Therefore, employing the algorithm with anti-monotonous evaluation metrics should be studied.

Base Learner		#Rules	%MHR	#Labels/MHR	#Conditions	%LabelConditions
Mic F	N^+	1342	0.89%	2.00	2.06	38.57%
Mic F	R^+	110	4.55%	2.00	0.98	25.00%
Mic F	N^+_-	86	48.83%	3.10	0.00	0.00%
Mic F	R^+_-	48	89.58%	4.02	0.02	100%
Mac F	N^+	87	45.98%	3.15	0.00	0.00%
Mac F	R^+	46	100%	2.39	0.93	6.98%
Mac F	N^+_-	86	48.84%	3.10	0.00	0.00%
Mac F	R^+_-	46	100%	3.85	0.00	0.00%
HA	N^+	3234	0.87%	2.00	1.80	57.20%
HA	R^+	2	100%	4.00	0.00	0.00%
HA	N^+_-	149	22.15%	3.36	0.85	62.70%
HA	R^+_-	334	97.60%	161.70	1.41	0.00%
SA	N^+	-	-	-	-	-
SA	R^+	-	-	-	-	-
SA	N^+_-	-	-	-	-	-
SA	R^+_-	-	-	-	-	-

Table 36: Model characteristics of the normal and the relaxed pruning approach on the data set CAL500. From left to right the number of learned rules (#Rules), the percentage of multilabel head rules (%MHR), the average number of labels per learned multilabel head rule (#Labels/MHR), the average number of conditions per rule (#Conditions) and the percentage of label conditions among all conditions (%LabelConditions). When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

Base Learner		#Rules	%MHR	#Labels/MHR	#Conditions	%LabelConditions
Mic F	N^+	44	0.00%	0.00	6.32	0.36%
Mic F	R^+	93	2.15%	2.00	2.37	0.45%
Mic F	N^+_-	46	0.00%	0.00	5.65	3.08%
Mic F	R^+_-	16	81.25%	2.00	1.31	0.00%
Mac F	N^+	6	0.00%	0.00	0.00	0.00%
Mac F	R^+	83	100%	2.01	1.00	0.00%
Mac F	N^+_-	6	0.00%	0.00	0.00	0.00%
Mac F	R^+_-	7	100%	2.57	1.00	14.29%
HA	N^+	185	0.00%	0.00	4.46	0.85%
HA	R^+	11	36.36%	2.00	5.81	0.00%
HA	N^+_-	21	0.00%	0.00	3.14	1.52%
HA	R^+_-	115	99.13%	2.96	1.98	0.44%
SA	N^+	197	0.00%	0.00	2.16	0.23%
SA	R^+	195	19.49%	2.00	1.93	0.27%
SA	N^+_-	108	0.00%	0.00	2.02	0.46%
SA	R^+_-	418	43.78%	2.00	1.50	1.28%

Table 37: Model characteristics of the normal and the relaxed pruning approach on the data set EMOTIONS. From left to right the number of learned rules (#Rules), the percentage of multilabel head rules (%MHR), the average number of labels per learned multilabel head rule (#Labels/MHR), the average number of conditions per rule (#Conditions) and the percentage of label conditions among all conditions (%LabelConditions). When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

Base Learner		#Rules	%MHR	#Labels/MHR	#Conditions	%LabelConditions
Mic F	N^+	101	0.00%	0.00	1.62	1.22%
Mic F	R^+	94	0.00%	0.00	1.67	1.27%
Mic F	N^+_-	149	2.01%	3.67	1.19	9.04%
Mic F	R^+_-	102	56.86%	2.00	1.35	10.87%
Mac F	N^+	18	33.33%	4.33	0.00	0.00%
Mac F	R^+	90	98.89%	2.01	1.00	0.00%
Mac F	N^+_-	19	36.84%	4.71	0.00	0.00%
Mac F	R^+_-	47	100%	2.13	0.94	4.55%
HA	N^+	82	0.00%	0.00	1.93	0.63%
HA	R^+	12	0.00%	0.00	4.83	0.00%
HA	N^+_-	83	2.41%	6.00	1.10	3.30%
HA	R^+_-	5	100%	43.80	0.80	0.00%
SA	N^+	82	3.66%	2.00	1.84	1.97%
SA	R^+	-	-	-	-	-
SA	N^+_-	131	30.53%	2.13	1.36	25.28
SA	R^+_-	-	-	-	-	-

Table 38: Model characteristics of the normal and the relaxed pruning approach on the data set MEDICAL. From left to right the number of learned rules (#Rules), the percentage of multilabel head rules (%MHR), the average number of labels per learned multilabel head rule (#Labels/MHR), the average number of conditions per rule (#Conditions) and the percentage of label conditions among all conditions (%LabelConditions). When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

Base Learner		#Rules	%MHR	#Labels/MHR	#Conditions	%LabelConditions
Mic F	N^+	36	0.00%	0.00	3.42	1.63%
Mic F	R^+	14	42.86%	2.00	2.00	3.57%
Mic F	N^+_-	40	0.00%	0.00	2.20	3.41%
Mic F	R^+_-	4	100%	3.50	3.00	0.00%
Mac F	N^+	7	0.00%	0.00	0.00	0.00%
Mac F	R^+	13	100%	2.08	1.00	0.00%
Mac F	N^+_-	7	0.00%	0.00	0.00	0.00%
Mac F	R^+_-	11	100%	2.00	0.82	0.00%
HA	N^+	57	0.00%	0.00	2.51	6.99%
HA	R^+	6	100%	2.33	1.33	0.00%
HA	N^+_-	22	0.00%	0.00	1.23	0.00%
HA	R^+_-	28	85.71%	2.92	1.36	2.63%
SA	N^+	71	0.00%	0.00	2.14	1.32%
SA	R^+	61	8.20%	2.00	1.92	1.71%
SA	N^+_-	105	4.76%	2.00	2.00	4.29%
SA	R^+_-	95	6.32%	2.00	1.85	3.41%

Table 39: Model characteristics of the normal and the relaxed pruning approach on the data set FLAGS. From left to right the number of learned rules (#Rules), the percentage of multilabel head rules (%MHR), the average number of labels per learned multilabel head rule (#Labels/MHR), the average number of conditions per rule (#Conditions) and the percentage of label conditions among all conditions (%LabelConditions). When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

Base Learner		#Rules	%MHR	#Labels/MHR	#Conditions	%LabelConditions
Mic F	N^+	71	0.00%	0.00	1.69	2.50%
Mic F	R^+	6	0.00%	0.00	1.67	0.00%
Mic F	N^+_-	6	0.00%	0.00	0.00	0.00%
Mic F	R^+_-	4	75.00%	2.00	0.00	0.00%
Mac F	N^+	6	0.00%	0.00	0.00	0.00%
Mac F	R^+	4	100%	2.00	1.00	0.00%
Mac F	N^+_-	6	0.00%	0.00	0.00	0.00%
Mac F	R^+_-	3	100%	2.00	0.00	0.00%
HA	N^+	328	0.00%	0.00	1.73	3.34%
HA	R^+	6	0.00%	0.00	1.67	0.00%
HA	N^+_-	11	0.00%	0.00	1.18	38.46%
HA	R^+_-	7	100%	5.43	0.71	0.00%
SA	N^+	239	0.00%	0.00	2.11	0.00%
SA	R^+	233	0.00%	0.00	2.12	0.00%
SA	N^+_-	982	0.00%	0.00	2.43	3.18%
SA	R^+_-	551	94.56%	2.00	1.06	2.40%

Table 40: Model characteristics of the normal and the relaxed pruning approach on the data set SCENE. From left to right the number of learned rules (#Rules), the percentage of multilabel head rules (%MHR), the average number of labels per learned multilabel head rule (#Labels/MHR), the average number of conditions per rule (#Conditions) and the percentage of label conditions among all conditions (%LabelConditions). When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

Base Learner		#Rules	%MHR	#Labels/MHR	#Conditions	%LabelConditions
Mic F	N^+	134	0.00%	0.00	1.71	1.75%
Mic F	R^+	138	2.90%	2.00	1.53	2.37%
Mic F	N^+_-	135	0.00%	0.00	1.48	0.50%
Mic F	R^+_-	66	89.39%	2.00	1.00	1.52%
Mac F	N^+	14	35.71%	2.00	0.00	0.00%
Mac F	R^+	87	100%	2.07	1.00	0.00%
Mac F	N^+_-	14	35.71%	2.00	0.00	0.00%
Mac F	R^+_-	38	100%	2.47	1.00	5.26%
HA	N^+	161	0.00%	0.00	1.66	1.12%
HA	R^+	1	0.00%	0.00	9.00	0.00%
HA	N^+_-	58	0.00%	0.00	1.00	0.00%
HA	R^+_-	3	100%	17.67	0.33	0.00%
SA	N^+	158	0.00	0.00	1.66	1.53%
SA	R^+	-	-	-	-	-
SA	N^+_-	130	0.00%	0.00	1.16	0.66%
SA	R^+_-	-	-	-	-	-

Table 41: Model characteristics of the normal and the relaxed pruning approach on the data set BIRDS. From left to right the number of learned rules (#Rules), the percentage of multilabel head rules (%MHR), the average number of labels per learned multilabel head rule (#Labels/MHR), the average number of conditions per rule (#Conditions) and the percentage of label conditions among all conditions (%LabelConditions). When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

Base Learner		#Rules	%MHR	#Labels/MHR	#Conditions	%LabelConditions
Mic F	N^+	148	0.00%	0.00	1.99	14.97%
Mic F	R^+	17	23.53%	2.00	1.00	5.88%
Mic F	N^+_-	14	0.0%	0.00	0.00	0.00%
Mic F	R^+_-	11	36.36%	2.25	0.00	0.00%
Mac F	N^+	14	0.00%	0.00	0.00	0.00%
Mac F	R^+	1	100%	2.00	0.00	0.00%
Mac F	N^+_-	14	0.00%	0.00	0.00	0.00%
Mac F	R^+_-	8	87.5%	2.43	0.00	0.00%
HA	N^+	1202	0.00%	0.00	2.06	16.60%
HA	R^+	1	100%	2.00	0.00	0.00%
HA	N^+_-	26	0.00%	0.00	0.77	20.00%
HA	R^+_-	354	100%	4.99	1.97	0.00%
SA	N^+	1188	0.25%	2.00	2.63	10.04%
SA	R^+	1075	1.12%	2.00	2.59	6.01%
SA	N^+_-	1654	40.87%	2.00	3.04	15.97%
SA	R^+_-	845	59.53%	2.00	2.32	1.83%

Table 42: Model characteristics of the normal and the relaxed pruning approach on the data set YEAST. From left to right the number of learned rules (#Rules), the percentage of multilabel head rules (%MHR), the average number of labels per learned multilabel head rule (#Labels/MHR), the average number of conditions per rule (#Conditions) and the percentage of label conditions among all conditions (%LabelConditions). When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

B Results of Performance Evaluations

Base Learner		HA		SA		Mic Prec		Mic Rec		Mic F		Mac Prec		Mac Rec		Mac F	
All 0s		85.15	-	0.0	-	0.0	-	0.0	-	0.0	-	1.15	-	1.15	-	1.15	-
All 1s		14.85	-	0.0	-	14.85	-	100	-	25.86	-	14.85	-	98.85	-	22.83	-
Mic F	N^+	77.35	10	0.0	8	34.77	10	59.93	3	**44.01**	1	**15.52**	1	26.08	3	18.69	3
Mic F	R^+	85.02	9	0.0	8	49.39	9	34.48	4	40.61	3	11.58	6	11.98	4	11.31	5
Mic F	N^+_-	**86.55**	3	0.0	8	**65.19**	3	20.19	10	30.83	9	4.15	10	5.74	11	4.75	10
Mic F	R^+_-	**86.55**	3	0.0	8	**65.19**	3	20.19	10	30.83	9	4.15	10	5.74	11	4.75	10
Mac F	N^+	15.40	12	0.0	8	14.93	12	**99.93**	1	25.97	12	14.84	2	**98.28**	1	**22.81**	1
Mac F	R^+	85.28	8	0.0	8	50.68	8	33.90	5	40.63	2	12.87	4	11.87	5	11.70	4
Mac F	N^+_-	**86.55**	3	0.0	8	**65.19**	3	20.19	10	30.83	9	4.15	10	5.75	8.5	4.75	10
Mac F	R^+_-	**86.55**	3	0.0	8	**65.19**	3	20.19	10	30.83	9	4.15	10	5.75	8.5	4.75	10
HA	N^+	71.08	11	0.0	8	29.01	11	65.52	2	40.22	4	14.43	3	32.49	2	19.49	2
HA	R^+	**86.55**	3	0.0	8	**65.19**	3	20.19	10	30.83	9	4.15	10	5.74	11	4.75	10
HA	N^+_-	86.25	7	0.0	8	60.26	7	21.78	6	32.00	5	12.61	5	6.79	6	7.12	6
HA	R^+_-	86.54	6	0.0	8	65.12	6	20.21	7	30.85	6	4.46	7	5.77	7	4.94	7
SA	N^+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SA	R^+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SA	N^+_-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SA	R^+_-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 43: Predictive performance of the normal and the relaxed pruning approach on the data set CAL500. Both approaches were tested with different settings, mainly different base learners and whether or not to also learn negative heads. When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

Base Learner		HA		SA		Mic Prec		Mic Rec		Mic F		Mac Prec		Mac Rec		Mac F	
All 0s		67.08	-	0.0	-	0.0	-	0.0	-	0.0	-	0	-	0	-	0	-
All 1s		32.92	-	0.0	-	32.92	-	100	-	49.53	-	32.92	-	100	-	49.12	-
Mic F	N^+	73.60	7	12.38	12	59.08	9	64.41	5	61.63	3	58.17	6	64.22	5	60.75	2
Mic F	R^+	72.11	12	15.35	9	57.14	11	61.15	6.5	59.08	6	55.39	9	62.03	6	57.18	6
Mic F	N^+_-	73.27	8	19.31	3	60.62	8	53.63	9	56.91	10	60.33	4	53.61	9	56.56	8
Mic F	R^+_-	75.00	3	16.83	5.5	64.55	6	53.38	10	58.44	8	62.45	3	50.46	11	53.61	10
Mac F	N^+	32.92	16	0.0	15.5	32.92	15	**100**	1	49.53	14	32.92	15	**100**	1	49.12	13
Mac F	R^+	64.11	15	6.93	14	47.35	14	80.45	2	59.61	4	47.83	13	80.10	2	59.42	3
Mac F	N^+_-	67.08	14	0.0	15.5	0.0	16	0.0	16	0.0	16	0.0	16	0.0	16	0.0	16
Mac F	R^+_-	74.84	5	20.30	2	65.77	5	49.12	12	56.24	11	53.47	11	47.38	12	49.43	12
HA	N^+	72.19	11	15.84	7.5	55.87	12	73.93	3	**63.65**	1	54.99	10	73.96	3	**62.86**	1
HA	R^+	74.34	6	16.83	5.5	67.46	3	42.61	13	52.23	12	56.28	8	42.37	13	47.46	14
HA	N^+_-	74.92	4	14.85	10.5	70.56	2	40.85	14	51.75	13	**69.20**	1	40.86	14	50.48	11
HA	R^+_-	72.44	9	7.92	13	**74.44**	1	24.81	15	37.22	15	35.78	14	25.79	15	28.68	15
SA	N^+	72.28	10	14.85	10.5	57.63	10	59.65	8	58.62	7	56.71	7	59.79	7	58.03	5
SA	R^+	69.55	13	15.84	7.5	52.94	13	67.67	4	59.41	5	51.84	12	68.51	4	58.40	4
SA	N^+_-	**75.66**	1.5	18.81	4	67.33	4	50.63	11	57.80	9	65.23	2	50.71	10	55.91	9
SA	R^+_-	**75.66**	1.5	**22.77**	1	63.54	7	61.15	6.5	62.32	2	60.18	5	59.08	8	57.12	7

Table 44: Predictive performance of the normal and the relaxed pruning approach on the data set EMOTIONS. Both approaches were tested with different settings, mainly different base learners and whether or not to also learn negative heads. When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

Base Learner		HA		SA		Mic Prec		Mic Rec		Mic F		Mac Prec		Mac Rec		Mac F	
All 0s		97.24	-	0.0	-	0.0	-	0.0	-	0.0	-	13.33	-	13.33	-	13.33	-
All 1s		2.76	-	0.0	-	2.76	-	100	-	5.36	-	2.76	-	86.67	-	4.99	-
Mic F	N^+	98.45	7	53.80	7	71.15	9	73.38	6	72.25	6	41.63	6	43.76	4	41.39	5
Mic F	R^+	**98.74**	1.5	60.93	2	76.59	5	78.13	2	**77.35**	1	44.68	3	45.13	3	44.73	2
Mic F	N^+_-	98.58	6	57.21	4	73.87	7	75.25	4	74.55	4	41.99	5	41.68	6	41.13	6
Mic F	R^+_-	**98.74**	1.5	**63.10**	1	79.12	3	73.88	5	76.41	2	**48.24**	1	45.22	2	**45.97**	1
Mac F	N^+	18.24	12	0.0	11.5	3.22	11	**98.63**	1	6.23	11	2.72	12	**71.11**	1	4.92	12
Mac F	R^+	96.14	11	17.21	10	37.85	10	62.50	9	47.15	9	33.85	8	41.65	7	36.27	7
Mac F	N^+_-	97.24	10	0.0	11.5	0.0	12	0.0	12	0.0	12	13.33	11	13.33	12	13.33	11
Mac F	R^+_-	98.42	8	43.88	8	81.84	2	54.63	10	65.52	8	31.88	9	32.78	10	32.20	9
HA	N^+	98.59	5	58.75	3	73.93	6	75.50	3	74.71	3	42.20	4	41.94	5	41.78	4
HA	R^+	98.63	3	54.73	6	**81.98**	1	64.25	8	72.04	7	36.77	7	35.32	9	35.75	8
HA	N^+_-	98.60	4	55.81	5	78.42	4	68.12	7	72.91	5	47.47	2	40.99	8	42.67	3
HA	R^+_-	97.68	9	19.38	9	73.16	8	24.88	11	37.13	10	18.27	10	18.97	11	18.56	10
SA	N^+	98.57	-	58.91	-	73.88	-	74.25	-	74.06	-	42.21	-	41.81	-	41.72	-
SA	R^+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SA	N^+_-	98.65	-	57.83	-	76.95	-	72.63	-	74.73	-	44.58	-	43.26	-	43.28	-
SA	R^+_-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 45: Predictive performance of the normal and the relaxed pruning approach on the data set MEDICAL. Both approaches were tested with different settings, mainly different base learners and whether or not to also learn negative heads. When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

Base Learner		HA		SA		Mic Prec		Mic Rec		Mic F		Mac Prec		Mac Rec		Mac F	
All 0s		52.31	-	0.0	-	0.0	-	0.0	-	0.0	-	0.0	-	0.0	-	0.0	-
All 1s		47.69	-	0.0	-	47.69	-	100	-	64.58	-	47.69	-	100	-	61.26	-
Mic F	N^+	69.01	11	0.0	15.5	61.66	13	92.63	2	74.03	4	57.06	9	85.88	2	**67.89**	1
Mic F	R^+	72.97	3	6.15	12	67.80	9	82.49	6	74.43	2	60.52	6	70.09	8	63.59	6
Mic F	N^+_-	71.43	6	10.77	4	68.35	7	74.65	10	71.37	10	62.98	3	68.24	9	63.85	4
Mic F	R^+_-	73.85	2	9.23	7.5	72.90	2	71.89	13	72.39	6	59.85	7	57.43	13	56.73	14
Mac F	N^+	47.69	16	0.0	15.5	47.69	16	**100**	1	64.58	15	47.69	15	**100**	1	61.26	9
Mac F	R^+	70.99	8	7.69	10.5	64.21	11	88.48	4	74.42	3	56.44	10	76.40	4	64.08	3
Mac F	N^+_-	67.91	14	9.23	7.5	68.21	8	61.29	15	64.56	16	29.23	16	42.86	16	34.46	16
Mac F	R^+_-	**74.07**	1	15.38	2	69.72	5	80.65	8	**74.79**	1	58.65	8	65.16	10	59.76	12
HA	N^+	68.35	13	4.62	13	61.51	14	89.86	3	73.03	5	55.91	12	83.56	3	66.49	2
HA	R^+	69.45	10	10.77	4	64.44	10	80.18	9	71.46	9	62.61	4	74.52	6	63.38	7
HA	N^+_-	72.31	5	10.77	4	**73.10**	1	66.36	14	69.57	13	**66.83**	1	54.72	14	57.78	13
HA	R^+_-	69.89	9	**16.92**	1	71.98	3	60.37	16	65.66	14	53.12	14	45.27	15	45.33	15
SA	N^+	68.79	12	3.08	14	63.35	12	82.03	7	71.49	8	55.53	13	72.16	7	62.56	8
SA	R^+	65.93	15	9.23	7.5	60.13	15	84.79	5	70.36	12	55.97	11	75.51	5	63.75	5
SA	N^+_-	71.21	7	9.23	7.5	68.38	6	73.73	11	70.95	11	60.98	5	62.57	11	60.29	11
SA	R^+_-	72.53	4	7.69	10.5	70.54	4	72.81	12	71.66	7	63.12	2	61.61	12	60.75	10

Table 46: Predictive performance of the normal and the relaxed pruning approach on the data set FLAGS. Both approaches were tested with different settings, mainly different base learners and whether or not to also learn negative heads. When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

Base Learner		HA		SA		Mic Prec		Mic Rec		Mic F		Mac Prec		Mac Rec		Mac F	
All 0s		81.90	-	0.0	-	0.0	-	0.0	-	0.0	-	0.0	-	0.0	-	0.0	-
All 1s		18.10	-	0.0	-	18.10	-	100	-	30.65	-	18.10	-	100	-	30.61	-
Mic F	N^+	68.09	14	7.53	9	33.81	10	79.68	3	47.48	3	38.07	9	79.61	3	50.02	3
Mic F	R^+	83.24	4	19.31	6	58.03	4	26.71	8	36.58	7	49.09	2	26.38	8	33.07	7
Mic F	N^+_-	81.90	8.5	0.0	14	0.0	14.5	0.0	14.5	0.0	14.5	0.0	14.5	0.0	14.5	0.0	14.5
Mic F	R^+_-	81.90	8.5	0.0	14	0.0	14.5	0.0	14.5	0.0	14.5	0.0	14.5	0.0	14.5	0.0	14.5
Mac F	N^+	18.10	16	0.0	14	18.10	12	**100**	1	30.42	11	18.10	12	**100**	1	30.62	8
Mac F	R^+	81.70	11	1.09	11	48.71	7	20.32	10	28.68	12	32.38	10	20.49	10	24.81	12
Mac F	N^+_-	81.90	8.5	0.0	14	0.0	14.5	0.0	14.5	0.0	14.5	0.0	14.5	0.0	14.5	0.0	14.5
Mac F	R^+_-	81.90	8.5	0.0	14	0.0	14.5	0.0	14.5	0.0	14.5	0.0	14.5	0.0	14.5	0.0	14.5
HA	N^+	63.10	15	6.77	10	30.67	11	82.37	2	44.70	6	31.09	11	83.67	2	44.78	4
HA	R^+	**83.85**	1	15.72	8	**68.92**	1	19.63	12	30.56	9	46.16	5	19.86	12	27.06	11
HA	N^+_-	83.65	3	17.73	7	66.24	2	19.78	11	30.47	10	**51.12**	1	20.21	11	28.44	10
HA	R^+_-	83.77	2	21.82	5	65.02	3	22.32	9	33.24	8	43.19	8	23.13	9	30.09	9
SA	N^+	79.49	12	25.59	3	45.25	8	63.43	4	**52.82**	1	47.37	4	63.81	4	**54.01**	1
SA	R^+	78.09	13	23.91	4	42.67	9	61.12	5	50.25	2	44.50	7	61.47	5	51.16	2
SA	N^+_-	82.94	6	27.93	2	53.97	5	39.26	7	45.45	5	44.78	6	40.22	7	40.31	6
SA	R^+_-	82.97	5	**32.19**	1	53.91	6	40.88	6	46.50	4	47.72	3	41.47	6	42.80	5

Table 47: Predictive performance of the normal and the relaxed pruning approach on the data set SCENE. Both approaches were tested with different settings, mainly different base learners and whether or not to also learn negative heads. When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

Base Learner		HA		SA		Mic Prec		Mic Rec		Mic F		Mac Prec		Mac Rec		Mac F	
All 0s		94.90	-	46.75	-	0.0	-	0.0	-	0.0	-	0.0	-	0.0	-	0.0	-
All 1s		5.10	-	0.0	-	5.10	-	100	-	9.71	-	5.10	-	100	-	9.46	-
Mic F	N^+	93.87	7	39.32	9	40.60	6	43.45	3	**41.98**	1	32.47	3	35.74	2	**33.00**	1
Mic F	R^+	93.58	9	37.77	10	38.66	7	44.09	2	41.19	2	29.00	5	32.99	3	29.65	3
Mic F	N^+_-	93.68	8	41.49	7	38.24	8	38.98	6	38.61	4	27.91	6	28.62	5	27.39	4
Mic F	R^+_-	94.82	5.5	47.37	2.5	48.45	4	24.92	7	32.91	5	**39.53**	1	19.13	7	25.10	5
Mac F	N^+	5.10	12	0.0	12	5.10	11	**100**	1	9.71	9	5.10	9	**100**	1	9.46	9
Mac F	R^+	90.83	11	36.22	11	24.90	10	39.62	5	30.58	6	19.20	8	28.19	6	22.14	6
Mac F	N^+_-	94.90	4	46.75	5	0.0	12	0.0	12	0.0	12	0.0	12	0.0	12	0.0	12
Mac F	R^+_-	**95.27**	1	**50.46**	1	62.11	3	18.85	9	28.92	7	22.68	7	9.27	9	12.53	8
HA	N^+	93.48	10	40.56	8	37.54	9	41.85	4	39.58	3	30.03	4	32.72	4	30.31	2
HA	R^+	95.05	3	47.06	4	69.57	2	5.11	10	9.52	10	3.66	11	2.16	11	2.72	11
HA	N^+_-	94.82	5.5	45.82	6	48.06	5	19.81	8	28.05	8	37.53	2	13.62	8	18.05	7
HA	R^+_-	95.11	2	47.37	2.5	**88.24**	1	4.79	11	9.09	11	4.64	10	2.47	10	3.22	10
SA	N^+	93.30	-	38.39	-	35.24	-	37.38	-	36.28	-	25.78	-	26.79	-	25.07	-
SA	R^+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SA	N^+_-	94.18	-	44.27	-	41.20	-	32.91	-	36.59	-	28.10	-	22.93	-	24.03	-
SA	R^+_-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 48: Predictive performance of the normal and the relaxed pruning approach on the data set BIRDS. Both approaches were tested with different settings, mainly different base learners and whether or not to also learn negative heads. When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

Base Learner		HA		SA		Mic Prec		Mic Rec		Mic F		Mac Prec		Mac Rec		Mac F	
All 0s		69.76	-	0.0	-	0.0	-	0.0	-	0.0	-	0.0	-	0.0	-	0.0	-
All 1s		30.24	-	0.0	-	30.24	-	100	-	46.44	-	30.24	-	100	-	42.60	-
Mic F	N^+	51.95	15	0.0	15	38.03	14	93.56	2	54.08	8	30.62	8	76.21	2	42.20	4
Mic F	R^+	77.54	3.5	7.31	6	65.60	10	54.10	6	**59.30**	1	30.80	7	30.00	6	29.39	6
Mic F	N^+_-	76.74	9	1.20	10	74.43	3.5	35.16	14	47.76	13	10.63	14	14.29	14	12.19	14
Mic F	R^+_-	76.74	9	1.20	10	**74.73**	1	35.16	14	47.76	13	10.63	14	14.29	14	12.19	14
Mac F	N^+	30.24	16	0.0	15	30.24	16	**100**	1	46.44	16	30.24	9	**100**	1	42.60	3
Mac F	R^+	77.54	3.5	10.03	3	66.50	9	51.85	8	58.27	4	31.68	6	27.99	8	27.94	8
Mac F	N^+_-	76.74	9	**12.00**	1.5	74.43	3.5	35.16	14	47.76	13	10.63	14	14.29	14	12.19	14
Mac F	R^+_-	76.74	9	**12.00**	1.5	74.43	3.5	35.16	14	47.76	13	10.63	14	14.29	14	12.19	14
HA	N^+	61.77	13	1.09	12	43.29	13	85.29	3	57.43	6	33.16	4	65.84	4	43.03	2
HA	R^+	76.74	9	1.20	10	74.43	3.5	35.16	14	47.76	13	10.63	14	14.29	14	12.19	14
HA	N^+_-	77.99	2	6.11	8	71.22	7	45.65	10	55.64	7	34.95	2	22.99	10	24.38	10
HA	R^+_-	76.75	6	0.76	13	71.33	6	38.64	11	50.13	10	18.26	11	16.82	11	15.43	11
SA	N^+	72.96	12	6.22	7	54.42	12	65.07	5	59.27	2	**37.19**	1	43.83	5	40.04	5
SA	R^+	52.27	14	0.0	15	36.97	15	82.07	4	50.98	9	34.58	3	75.98	3	**44.17**	1
SA	N^+_-	76.90	5	8.40	5	64.36	11	52.89	7	58.06	5	28.77	10	29.59	7	28.70	7
SA	R^+_-	**78.16**	1	9.38	4	68.86	8	50.70	9	58.40	3	32.71	5	27.13	9	27.93	9

Table 49: Predictive performance of the normal and the relaxed pruning approach on the data set YEAST. Both approaches were tested with different settings, mainly different base learners and whether or not to also learn negative heads. When using subset accuracy, some experiments did not finish. Missing values are indicated by "-" in the respective entry.

References

Bosc, G., Golebiowski, J., Bensafi, M., Robardet, C., Plantevit, M., Boulicaut, J.-F., and Kaytoue, M. (2016). Local subgroup discovery for eliciting and understanding new structure-odor relationships. In Calders, T., Ceci, M., and Malerba, D., editors, *Discovery Science: 19th International Conference, DS 2016*, volume 9956 of *Discovery Science: 19th International Conference, DS 2016, Bari, Italy, October 19–21, 2016, Proceedings*, pages 19–34, Bari, Italy. Springer International Publishing.

Chawla, N. V. (2005). Data mining for imbalanced datasets: An overview. In Maimon, O. and Rokach, L., editors, *Data Mining and Knowledge Discovery Handbook*, volume 2, chapter 45. Springer.

Dembczyński, K., Waegeman, W., Cheng, W., and Hüllermeier, E. (2012). On label dependence and loss minimization in multi-label classification. *Machine Learning*, 88(1):5–45.

Friedman, J. H. (1997). On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1:55–77.

Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54.

Janssen, F. (2012). *Heuristic Rule Learning*. PhD thesis, TU Darmstadt, Knowledge Engineering Group.

Janssen, F. and Fürnkranz, J. (2010). The seco-framework for rule learning. Technical report, TU Darmstadt, Knowledge Engineering Group.

Janssen, F. and Zopf, M. (2012). The seco-framework for rule learning. In *Proceedings of the German Workshop on Lernen, Wissen, Adaptivität - LWA2012*.

Koyejo, O. O., Natarajan, N., Ravikumar, P. K., and Dhillon, I. S. (2015). Consistent multilabel classification. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 3321–3329. Curran Associates, Inc.

Loza Mencía, E. (2012). *Efficient Pairwise Multilabel Classification*. Dissertation, Technische Universität Darmstadt, Knowledge Engineering Group.

Loza Mencía, E., Fürnkranz, J., Hüllermeier, E., and Rapp, M. (2018). Learning interpretable rules for multi-label classification. In Escalante, H. J., Escalera, S., Guyon, I., Baró, X., Yağmur, Güçlütürk, Güçlü, U., and van Gerven, M. A. J., editors, *Explainable and Interpretable Models in Computer Vision and Machine Learning*, volume 1. Springer.

Loza Mencía, E. and Janssen, F. (2013). Towards multilabel rule learning. In Henrich, A. and Sperker, H.-C., editors, *Proceedings of the German Workshop on Lernen, Wissen, Adaptivität - LWA2013*, pages 155–158, Bamberg, Germany.

Loza Mencía, E. and Janssen, F. (2016). Learning rules for multi-label classification: a stacking and a separate-and-conquer approach. *Machine Learning*, 105(1):77–126.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

Newlands, D. and Webb, G. I. (2004). Alternative strategies for decision list construction. In *Proceedings of the Fourth Data Mining Conference (DM IV 03)*, pages 265–273.

Ng, R. T., Lakshmanan, L. V. S., Han, J., and Pang, A. (1998). Exploratory mining and pruning optimizations of constrained associations rules. *SIGMOD Rec.*, 27(2):13–24.

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

Rapp, M., Loza Mencía, E., and Fürnkranz, J. (2018). Exploiting anti-monotonicity of multi-label evaluation measures for inducing multi-label rules. In Phung, D. Q., Tseng, V. S., Webb, G. I., Ho, B., Ganji, M., and Rashidi, L., editors, *Proceedings of the 22nd Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-18)*, pages 29–42, Melbourne, Australia. Springer-Verlag.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2011). Classifier chains for multi-label classification. *Machine Learning*, 85(3):333.

Sasaki, Y. (2007). The truth of the f-measure. School of Computer Science, University of Manchester.

Thabtah, F., Cowling, P., and Peng, Y. (2006). Multiple labels associative classification. *Knowledge and Information Systems*, 9:109–129.

Tsoumakas, G. and Katakis, I. (2007). Multi-label classification: An overview. *Int J Data Warehousing and Mining*, 2007:1–13.

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2005). Mining multi-label data. In Maimon, O. and Rokach, L., editors, *Data Mining and Knowledge Discovery Handbook*, volume 2, chapter 34. Springer.

Webb, G. (1993). Learning decision lists by prepending inferred rules. In *In Proceedings of the Australian Workshop on Machine Learning and Hybrid Systems*, pages 6–10.

Zhang, M.-L., Li, Y.-K., Liu, X.-Y., and Geng, X. (2018). Binary relevance for multi-label learning: an overview. *Frontiers of Computer Science*, 12(2):191–202.

Zhu, S., Ji, X., Xu, W., and Gong, Y. (2005). Multi-labelled classification using maximum entropy method. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 274–281, New York, NY, USA. ACM.