Multi-label Rule Learning

Michael Rapp

Dissertation zur Erlangung des
akademischen Grades *Doktor-Ingenieur (Dr.-Ing.)*

Dissertationsschrift in englischer Sprache von
M.Sc. Michael Rapp
aus Dannstadt-Schauernheim, Deutschland
geb. am 27.04.1990 in Mühlacker

Erstreferent: Prof. Dr. Kristian Kersting
Korreferenten: Prof. Dr. Eyke Hüllermeier
Prof. Dr. Johannes Fürnkranz

Tag der Einreichung: 02.09.2022
Tag der Prüfung: 14.10.2022

Knowledge Engineering Group
Fachbereich Informatik
Technische Universität Darmstadt
Darmstadt, 2022

Michael Rapp: *Multi-label Rule Learning*

Darmstadt, Technische Universität Darmstadt
Jahr der Veröffentlichung auf TUprints: 2022
URN: urn:nbn:de:tuda-tuprints-220993
Tag der mündlichen Prüfung: 14.10.2022

Erklärung zur Dissertation

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

Datum und Unterschrift

Abstract

Research on multi-label classification is concerned with developing and evaluating algorithms that learn a predictive model for the automatic assignment of data points to a subset of predefined class labels. This is in contrast to traditional classification settings, where individual data points cannot be assigned to more than a single class. As many practical use cases demand a flexible categorization of data, where classes must not necessarily be mutually exclusive, multi-label classification has become an established topic of machine learning research. Nowadays, it is used for the assignment of keywords to text documents, the annotation of multimedia files, such as images, videos, or audio recordings, as well as for diverse applications in biology, chemistry, social network analysis, or marketing. During the past decade, increasing interest in the topic has resulted in a wide variety of different multi-label classification methods. Following the principles of supervised learning, they derive a model from labeled training data, which can afterward be used to obtain predictions for yet unseen data. Besides complex statistical methods, such as artificial neural networks, symbolic learning approaches have not only been shown to provide state-of-the-art performance in many applications but are also a common choice in safety-critical domains that demand human-interpretable and verifiable machine learning models. In particular, rule learning algorithms have a long history of active research in the scientific community. They are often argued to meet the requirements of interpretable machine learning due to the human-legible representation of learned knowledge in terms of logical statements.

This work presents a modular framework for implementing multi-label rule learning methods. It does not only provide a unified view of existing rule-based approaches to multi-label classification, but also facilitates the development of new learning algorithms. Two novel instantiations of the framework are investigated to demonstrate its flexibility. Whereas the first one relies on traditional rule learning techniques and focuses on interpretability, the second one is based on a generalization of the gradient boosting framework and focuses on predictive performance rather than the simplicity of models. Motivated by the increasing demand for highly scalable learning algorithms that are capable of processing large amounts of training data, this work also includes an extensive discussion of algorithmic optimizations and approximation techniques for the efficient induction of rules. As the novel multi-label classification methods that are presented in this work can be viewed as instantiations of the same framework, they can both benefit from most of these principles. Their effectiveness and efficiency are compared to existing baselines experimentally.

Zusammenfassung

Forschung im Bereich der Multi-label Klassifizierung beschäftigt sich mit der Entwicklung und Bewertung von Algorithmen, die Vorhersagemodelle für die automatische Zuweisung von Datenpunkten zu einer Untermenge vordefinierter Klassen lernen. Dies unterscheidet sich von traditionellen Problemstellungen, die es nicht erlauben, einzelne Datenpunkte mehr als einer Klasse zuzuordnen. Da viele praktische Anwendungen eine flexible Zuordnung von Daten erfordern, bei der sich Klassen nicht gegenseitig ausschließen, wurde die Multi-label Klassifizierung zu einem etablierten Thema innerhalb des machinellen Lernens. Sie wird für die Zuweisung von Schlagworten zu Textdokumenten, die Annotation von Multimedia-Dateien, wie Bildern, Videos oder Audioaufnahmen, sowie für vielseitige Anwendungen in der Biologie, Chemie oder der Analyse von sozialen Netzwerken verwendet. Während der letzten Dekade hat das zunehmende Interesse an dem Thema zu einer Vielzahl von verschiedenen Klassifizierungsmethoden geführt. Gemäß den Prinzipien des überwachten Lernens leiten diese ein Vorhersagemodell von Trainingsdaten ab, das später genutzt werden kann um Vorhersagen für noch unbekannte Daten zu ermitteln. Neben komplexen statistischen Methoden, wie künstlichen neuronalen Netzen, können auch symbolische Lernansätze in vielen Anwendungsbereichen eine hohe Vorhersagegenauigkeit erzielen. Sie werden vor allem in sicherheitskritischen Bereichen verwendet, die durch Menschen interpretier- und verifizierbare Modelle erfordern. Insbesondere Regellernalgorithmen haben eine lange Historie aktiver Foschung vorzuweisen. Aufgrund der menschenlesbaren Repräsentation von Wissen in Form von logischen Ausdrücken, wird häufig argumentiert, dass sie die Anforderungen an interpretierbares maschinelles Lernen erfüllen.

Diese Arbeit stellt ein modulares System für die Umsetzung von Multi-label Regellernmethoden vor. Es bietet nicht nur eine einheitliche Sichtweise auf existierende regelbasierte Ansätze für die Multi-label Klassifizierung, sondern vereinfacht auch die Entwicklung neuer Lernalgorithmen. Es werden zwei neuartige Umsetzungen dieses Systems untersucht um dessen Flexibilität zu demonstrieren. Während die erste auf traditionelle Regellerntechniken setzt und einen großen Wert auf Interpretierbarkeit legt, basiert die zweite auf einer Generalisierung des Gradient Boosting-Systems und ist auf Vorhersagegenauigkeit, statt der Einfachheit von Modellen, fokussiert. Motiviert durch den steigenden Bedarf nach skalierbaren Lernalgorithmen, die große Datenmengen verarbeiten können, enthält diese Arbeit außerdem eine umfangreiche Diskussion von algorithmischen Optimierungen und Approximierungstechniken für die effiziente Induktion von Regeln. Da die neuartigen Klassifizierungsmethoden, die in dieser Arbeit vorgestellt werden, auf den selben Grundprinzipien beruhen, können sie beide von diesen Techniken profitieren. Ihre Effektivität und Effizienz wird experimentell mit existierenden Verfahren verglichen.

Acknowledgments

First and foremost, I want to thank my supervisors, Johannes Fürnkranz and Eyke Hüllermeier, without whom this thesis would not have been possible. Not only have they provided a productive environment to conduct research, but they also contributed to this work significantly by sharing their extensive knowledge and ideas during numerous discussions. Special thanks also go to Eneldo Loza Mencía, who was an essential part of our team. As the supervisor of my master's thesis, he was the person who started my interest in the topic of this dissertation in the first place. More than once, he has pointed me in the right direction and was always willing to help with unexpected problems that unavoidably arise when doing research. In addition, I want to thank Gabriele Ploch, who never ran out of answers to organizational inquiries and kept our research group running smoothly.

I am also very grateful to my colleagues and co-authors, in particular Moritz Kulessa and Marcel Wever, for the effort they put into our collaborative work. I always enjoyed our vivid, though not always work-related, discussions. Unfortunately, there was a time when we could not meet each other very often due to a global pandemic. Despite, or maybe because of, these unusual circumstances, it always felt good to be connected with people who pursue the same goals and struggle with similar problems.

Moreover, I would not have been able to finish this dissertation without the support of my parents, siblings, and friends. I am lucky to have parents who, for my whole life, have unconditionally supported my decisions and always encouraged me to pursue my goals. The role they played in the successful completion of this thesis cannot be over-exaggerated. I also want to thank my uncle Ralf, who has affected my life more than he can imagine by supporting my interest in technology from a young age when not everyone saw the value of spending massive amounts of time in front of a computer. Kind regards and many thanks also go to my closest and long-lasting friends, Julian, Lukas, Matthias, Raphael, Sabrina, and Steffen, for all the good memories and fun times that helped keep my work-life balance intact. I am very happy that we managed to stay in touch during all those years. Finally, among all the people that have earned a special place in my life, my girlfriend Rebecca is, of course, the most important one. She deserves the highest order of gratitude for being the kind, selfless and supportive person she is.

Publications

Parts of this thesis have previously been published in the following publications. They are listed in chronological order.

Michael Rapp, Eneldo Loza Mencía, and Johannes Fürnkranz (2018). 'Exploiting Anti-monotonicity of Multi-label Evaluation Measures for Inducing Multi-label Rules'. In: *Proc. Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pp. 29–42

Eneldo Loza Mencía, Johannes Fürnkranz, Eyke Hüllermeier, and Michael Rapp (2018). 'Learning Interpretable Rules for Multi-Label Classification'. In: *Explainable and Interpretable Models in Computer Vision and Machine Learning*, pp. 81–113

Michael Rapp, Eneldo Loza Mencía, and Johannes Fürnkranz (2019). 'On the Trade-off Between Consistency and Coverage in Multi-label Rule Learning Heuristics'. In: *Proc. International Conference on Discovery Science*, pp. 96–111

Yannik Klein, Michael Rapp, and Eneldo Loza Mencía (2019). 'Efficient Discovery of Expressive Multi-label Rules using Relaxed Pruning'. In: *Proc. International Conference on Discovery Science*, pp. 367–382

Vu-Linh Nguyen, Eyke Hüllermeier, Michael Rapp, Eneldo Loza Mencía, and Johannes Fürnkranz (2020). 'On Aggregation in Ensembles of Multilabel Classifiers'. In: *Proc. International Conference on Discovery Science*, pp. 533–547

Michael Rapp, Eneldo Loza Mencía, Johannes Fürnkranz, Vu-Linh Nguyen, and Eyke Hüllermeier (2020). 'Learning Gradient Boosted Multi-label Classification Rules'. In: *Proc. European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pp. 124–140

Eyke Hüllermeier, Johannes Fürnkranz, Eneldo Loza Mencía, Vu-Linh Nguyen, and Michael Rapp (2020). 'Rule-based Multi-label Classification: Challenges and Opportunities'. In: *International Joint Conference on Rules and Reasoning*, pp. 3–19

Michael Rapp, Eneldo Loza Mencía, Johannes Fürnkranz, and Eyke Hüllermeier (2021). 'Gradient-based Label Binning in Multi-label Classification'. In: *Proc. European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pp. 462–477

Michael Rapp (2021). 'BOOMER – An Algorithm for Learning Gradient Boosted Multi-label Classification Rules'. In: *Software Impacts* 10, p. 100137

Michael Rapp, Moritz Kulessa, Eneldo Loza Mencía, and Johannes Fürnkranz (2021). 'Correlation-based Discovery of Disease Patterns for Syndromic Surveillance'. In: *Frontiers in Big Data* 4

Eyke Hüllermeier, Marcel Wever, Eneldo Loza Mencía, Johannes Fürnkranz, and Michael Rapp (2022). 'A flexible class of dependence-aware multi-label loss functions'. In: *Machine Learning* 111.2, pp. 713–737

Contents

List of Figures

List of Tables

List of Algorithms

Notation

Label Space

λ	A single label
k	Variable that iterates the available labels
K	The total number of available labels
$\mathcal{L}$	The set of available labels
$\mathcal{Y}$	The space of possible labelings
y	Indicates whether a label is relevant to an example according to the ground truth
$\boldsymbol{y}$	The label vector of an example according to the ground truth
Y	A matrix of labels according to the ground truth
Λ	Indicates whether a label is mostly relevant or irrelevant

Feature Space

A	A single attribute
l	Variable that iterates the available attributes
L	The total number of available attributes
$\mathcal{A}$	The set of available attributes
x	Specifies an example's value for a particular attribute
$\boldsymbol{x}$	The feature vector of an example
X	A matrix that stores the feature values of examples
n	Variable that iterates the available examples
N	The total number of available examples
$\mathcal{X}$	The space of possible feature vectors

Classification Models

$\mathcal{D}$	A (multi-label) data set that is used to train a classification model
f	A (multi-label) classification function, e.g., a single rule
t	Variable that iterates the classification functions in a model
T	The total number of classification functions in a model
F	A (multi-label) classification model consisting of several classification functions
$\mathcal{F}$	The space of possible classification models
$\hat{y}$	Indicates whether a label is predicted to be relevant to an example
$\hat{\boldsymbol{y}}$	A label vector that is predicted for an example

$\hat{Y}$	A matrix of predicted labels

Performance Evaluation

$\mathcal{M}$	A (multi-label) evaluation measure
γ	A performance score according to a (multi-label) evaluation measure
C	A confusion matrix
α	The significance level used by statistical significance tests
r	The rank of an individual classification approach
d	Variable that iterates the datasets used in an empirical study
D	The total number of datasets used in an experimental study

Rule Learning Framework

b	The body of a rule
c	A condition that is contained in the body of a rule
μ	The threshold that is used by a condition
$\hat{p}$	A prediction for an individual label that is provided by the head of a rule
$\hat{\boldsymbol{p}}$	A vector of predictions that are provided by the head of a rule
q	A numerical score that assesses the quality of a rule
w	A weight, e.g., the weight of an individual example
$\boldsymbol{w}$	A vector of weights
W	A matrix of weights
S	A matrix of label space statistics that serve as a basis for learning rules

Separate-and-Conquer Rule Learning

$\mathcal{H}$	A (multi-label) rule learning heuristic
Δ	A relaxation lift affecting the quality of a rule depending on the size of its head
ρ	A relaxation lift function
$\hat{q}$	The quality of a rule according to a relaxation lift function

Gradient Boosting

ℓ	A (surrogate) loss function to be minimized during training
$\mathcal{R}$	The global training objective of a gradient boosting algorithm
Ω	A regularization term
δ	The weight of a regularization term
R	A regularization matrix
g	A gradient corresponding to the first partial derivative of a loss function
$\boldsymbol{g}$	A vector of gradients corresponding to individual labels

h	A Hessian corresponding to the second partial derivative of a loss function
H	A matrix of Hessians corresponding to pairs of labels
η	Shrinkage parameter (also known as learning rate)

Gradient-based Label Binning

$\mathcal{B}$	Set of indices that belong to a bin
m	Variable that iterates the bins used by an approximation method
M	The number of bins used by an approximation method
ω	The width used in equal-width binning
θ	A criterion that is used to determine a mapping to bins

Introduction | 1

Artificial intelligence is a longstanding subdomain of computer science that is concerned with the development of computer programs for solving complex tasks, which are commonly assumed to require some kind of "intelligence" (McCarthy, 1968). Even though it is debatable what capabilities a computer system must have to be considered intelligent, research on the topic has resulted in algorithms that can compete with, or even outperform humans in specific problem settings. Examples that have received widespread attention from the general public include computer programs that manage to beat world-class players in games like chess (F.-H. Hsu, 2002) or Go (Gibney, 2016), advances in autonomous driving (Thrun, 2010), or the commercial success of virtual assistance technology that is capable of understanding and imitating the human voice (Hoy, 2018).

Concept Learning

The technologies mentioned above have in common that they are all tailored to a specific application. Similarly, research on *concept learning* focuses on a well-defined family of problems. Its goal is to distinguish between objects of different categories and identify features that are shared between exemplars within a particular group (Angluin, 1988). Computer systems that are aimed at solving such *classification* tasks most often rely on historical data from which they try to extract concepts that generalize beyond the given observations. The term *data mining* is most commonly used if one is interested in extracting knowledge from previously collected data and gaining valuable insights. In contrast, *machine learning* methods aim to incorporate patterns that can be found in historical data into a condensed model, which allows conclusions about yet unseen data in the future. (Mitchell, 1997). Practical use cases of the latter include spam filters that automatically detect unsought messages (Guzella and Caminhas, 2009) or algorithms for optical character recognition that convert handwritten notes into machine-encoded text (Mori, Nishida, and Yamada, 1999). As computer algorithms, unlike human analysts, can process large amounts of data in a short amount of time, machine learning methods are gaining importance in our increasingly digitalized world, where data is not only generated daily by billions of individuals that interact with electronic devices or exchange with others publicly on social media platforms, but can also be collected in factories and other business environments, where computers are increasingly used for manufacturing processes and working routines.

Multi-label Classification

In traditional classification settings, even though several categories might be available in total, each object is assumed to belong to exactly one of them. The individual categories, which are usually referred to as *classes*, are considered to be mutually exclusive in such cases. As a result, a classification system is not allowed to associate an object with multiple classes simultaneously. However, many real-world applications demand a more flexible assignment of classes to objects. For example, consider a computer system that should automatically assign newspaper articles to predefined topics to facilitate searching for documents that match a user's interests. In such a scenario, a single article is likely concerned with several related topics, such as "politics and "economics", and should be discovered when searching for either one of them. A classification approach that assumes the available classes to be mutually exclusive cannot address this problem adequately and would most likely limit the system's usefulness in practice. Problems of this kind, where individual objects may be associated with a subset of the available classes rather than a single one, are considered *multi-label classification* problems (Tsoumakas, Katakis, and Vlahavas, 2009). Because of the large number of practical use cases, such as text classification, the annotation of multimedia data, as well as applications in the field of biology, chemistry, marketing, or social network analysis, research on multi-label classification has become an established topic in the machine learning community during the past decade (Gibaja and Ventura, 2014). Due to the diverse nature of the aforementioned applications and the large number of different machine learning approaches that may be used to tackle this problem domain, it comes with various interesting questions and challenges. In particular, ongoing work on the topic is often motivated by the desire to discover hidden interdependencies between individual classes, which are more commonly referred to as *labels*. Different dependencies, such as co-occurrences or partial exclusions, are frequently encountered in many applications (Dembczyński, Waegeman, et al., 2012). This also applies to the previously mentioned scenario, where newspaper articles should be associated with topics. In this particular example, if an article is concerned with "foreign politics", it should probably be labeled with the keyword "politics" because there is a hierarchical relation between both. However, the presence of these keywords might indicate that an article is unlikely to belong to the topic "sports", as they seldomly overlap. Algorithms that are capable of discovering such relations can provide valuable insights into the data and may benefit from this ability in terms of predictive performance. However, this does not only require models that can express dependencies in a suitable form but can also come with computational challenges in cases where large amounts of data should be processed.

Rule Learning Algorithms

Rule learning methods are among the most commonly used and well-understood approaches to concept learning and have a longstanding tradition in research on data mining and machine learning problems (Fürnkranz, Gamberger, and Lavrač, 2012). They belong to the family of symbolic learning approaches, i.e., they rely on symbolical representations to incorporate domain knowledge into a model. A rule-based model typically consists of several logical rules that describe different parts of the data. To identify the objects that are described by a rule, conditional clauses, which refer to the objects' individual features, are used. As symbolic rules can be presented in a human-legible form, they can be analyzed and verified by domain experts. For this reason, they are not only well-suited for data mining tasks, where the goal is to identify interesting patterns in the data, but can also be used in machine learning models for solving classification tasks. The use of rule-based approaches to multi-label classification appears to be promising and has previously been advocated for in the literature. Existing work on the topic suggests that rules allow for modeling correlations between labels in a natural form and help to make them explicit (Loza Mencía, Fürnkranz, et al., 2018). Depending on the methodology that is used for the construction of rule-based models, their characteristics and complexity may vastly differ. Especially in the multi-label setting, where different types of rules are conceivable for capturing information about the interactions between labels, the flexibility that comes with different rule learning techniques can be considered an advantage and makes them a versatile tool for the construction of classification models that are specifically tailored to different use cases and requirements.

1.1 Research Challenges

The present thesis is concerned with the development and evaluation of rule learning methods that are specifically tailored to the particularities of multi-label classification. Even though existing rule learning algorithms can be applied to this particular type of classification task by relying on well-known and model-agnostic transformation methods, which break down a multi-label classification problem into smaller single-label problems, machine learning approaches that can tackle this problem domain directly may provide advantages in terms of the following aspects.

- **Interpretability.** Rule learning methods are a common choice in domains that require interpretable classification models, which can be verified and analyzed by human experts (M. Du, N. Liu, and X. Hu, 2019). Due to the growing number of potential use cases for machine-guided decision-making, research on interpretable machine learning has recently gained relevance and has received increasing attention in the literature. In the context of multi-label classification, rule-based models are particularly interesting, as they allow to express different types of interactions between labels in a natural and human-legible form.

- **Exploitation of Label Dependencies.** As argued above, the idea of exploiting label dependencies to improve predictive performance is a major motivation of many multi-label classification methods and may help to improve the quality of predictions that are provided by a multi-label rule learning method. Even though existing rule-based methods are already able to capture certain types of label dependencies, alternative representations of such interactions, which demand different algorithmic solutions, are possible and worth investigating (Loza Mencía, Fürnkranz, et al., 2018).

- **Adaptation to Varying Target Measures.** Compared to traditional classification settings, a wide variety of evaluation measures is commonly used in multi-label classification to assess the quality of predictions for several labels. Prior research has shown that these measures vastly differ in their respective characteristics and may even conflict. Whereas the optimization of some measures requires taking label dependencies into account, other measures are more adequately adressed by simpler approaches that neglect the existence of such dependencies (Dembczyński, Waegeman, et al., 2012). Unfortunately, it often remains unclear what measure existing multi-label classification methods aim to optimize. Ideally, a single approach should be flexible enough to be tailored to varying measures, depending on the use case. To meet this requirement, a well-justified theoretical framework for constructing rule-based models, which can be adapted to different target measures, is needed.

- **Computational Efficiency.** With the increasing access to large amounts of data, there is an emerging need for highly scalable machine learning methods that are capable of processing vast amounts of historical data efficiently. This trend is also reflected by recent literature on multi-label classification, where great emphasis is put on the ability to deal with high-dimensional problems (Prajapati and Thakkar, 2019). Unfortunately, only a few implementations of rule learning algorithms are publicly available, and long-established implementations often lack the computational efficiency of more recent competitors. This work aims to make the presented algorithms available to a broader audience. To ensure that they are useful in practice, all algorithms in this work are published under an open-source license. Furthermore, parts of this thesis are devoted to algorithmic aspects that help to improve their computational efficiency.

Despite the long history of active research on rule learning algorithms and the availability of a wide variety of different methods for single-label classification, only a few rule-based approaches that are specifically tailored to multi-label classification are available today. This work contributes to this growing field of research by investigating how existing rule learning methodologies can be generalized to the multi-label setting. Furthermore, it presents novel algorithms that are designed with the aforementioned research challenges in mind.

1.2 Outline and Contributions

In the following, we outline the structure of this work and provide a brief overview of the topics that are discussed in each of the following chapters.

- **Chapter 2.** We start by providing an introduction to existing classification methods that are relevant to the remainder of this work. Following the topic of this thesis, strong emphasis is put on rule learning algorithms, but the discussion does also include decision tree learners, as they are closely related to rule-based classifiers. In addition, we motivate the use of ensemble techniques that may provide advantages for both types of learning approaches and serve as a foundation for the methodology in Chapter 6.
- **Chapter 3.** This chapter aims to give a basic understanding of multi-label classification problems. It includes a formal definition of the problem domain and introduces notations that are used in the remainder of this work. Besides a discussion of applications and commonly used evaluation measures, we also elaborate on established algorithms from the field of multi-label classification that may be used together with the rule-based approaches presented

in this work, such as transformation methods or techniques for dimensionality reduction.

- **Chapter 4.** In this chapter, a flexible and modular framework for the implementation of rule-based approaches to multi-label classification is presented. We do not only discuss the aspects that such rule learning methods have in common and highlight different possibilities to implementing individual components in varying ways, but also revisit existing publications on the topic and review them in the context of the proposed framework.
- **Chapter 5.** Based on the framework that the previous chapter focuses on, in Chapter 5, we investigate several aspects that are important to successfully apply the separate-and-conquer paradigm, many traditional rule learners rely on, to the multi-label classification setting. In particular, we present empirical results regarding the choice of the rule learning heuristic, which guides the construction of such models, and propose a methodology for the induction of rules that are able to model label dependencies, which is a major focus of research on multi-label classification.
- **Chapter 6.** In this chapter, another instantiation of the previously presented framework, which relies on the principles of gradient boosting, is proposed. Unlike the approach in Chapter 5, this particular method for learning rule-based ensembles focuses on predictive performance rather than the interpretability of the resulting models. Furthermore, it can deliberately be tailored to different multi-label evaluation measures and therefore allows to be flexibly be adjusted to different use cases and requirements.
- **Chapter 7.** This chapter is devoted to algorithmic details that allow for the efficient induction of rules. Due to the shared foundation, they are based on, the approaches in Chapter 5 and Chapter 6 have many aspects in common. As a consequence, they can both benefit from the optimizations that are discussed in this chapter. Besides a technique that enables dealing with sparse data efficiently and the possibility of using multi-threading, the descriptions and experimental results that are provided in this chapter are also concerned with an approximation method that helps speed up training on large datasets that come with many numerical features.
- **Chapter 8.** In addition to the algorithmic optimizations that are considered in the previous chapter, an approximation technique, which is deliberately targeted at the computational bottleneck of the boosting algorithm in Chapter 6, is proposed in this chapter. It revolves around the idea of assigning labels to a limited number of bins, which helps reduce the computational complexity in some particularly challenging use cases.

Finally, in Chapter 9, we summarize the content of the previous chapters and discuss the extent to which the present thesis has contributed to the research topic. Moreover, we provide an overview of various ideas that might be addressed in the future to improve the methodologies that are proposed in this work.

2 Overview of Relevant Classification Methods

Due to its long history, its diverse applications, and the wide variety of existing approaches, machine learning can be considered a multidisciplinary field that is based on findings from various disciplines, such as statistics, probability theory, information theory, psychology, and many more (see, e.g., Mitchell, 1997, for an overview on the topic). The goal of this chapter is to provide a basic insight into this large field of research by discussing a small selection of well-known machine learning methods that are relevant to this work.

Binary and Multi-class Classification

In adherence to the topic of this thesis, we restrict ourselves to methods that aim to solve classification problems. Research on this particular problem domain is concerned with the development of classification systems, also referred to as *classifiers*, that can provide categorical predictions for given data examples. Most prominently, this includes *binary classification* problems, where each example belongs to one out of two predefined classes. Based on the properties of an example, which we refer to as *attributes* or *features*, a classifier should be able to identify the class to which an example belongs. For example, the dataset "Mushroom" that is provided by the UCI machine learning repository[1] requires a classifier to predict whether different species of mushrooms are edible or poisonous, based on attributes that describe their appearance, odor, habitat, etc. Problems that come with more than two classes are referred to as *multi-class classification* problems. For example, there is also a variant of the aforementioned "Mushroom" dataset that demands a more fine-grained categorization of examples in terms of the classes "edible", "poisonous", "unknown" and "not recommended". For reasons of simplicity, we focus on binary classification problems in the following.

1: `https://archive.ics.uci.edu`

Supervised vs. Unsupervised Learning

Approaches to classification are often characterized as either *supervised* or *unsupervised learning* approaches (Kotsiantis, 2007). The former type of method aim at deriving a *model* from given *training examples* for which the true classes are known. A model that has been trained on labeled data incorporates information about the problem domain and can afterward be used to obtain predictions for unseen examples that have not yet been provided to the learning method as part of the training procedure. In contrast, unsupervised classification methods do not rely on a representation of the problem domain in terms of a pre-trained model, but try to identify structures that are hidden in unlabeled data by creating groups of similar examples, referred to as *clusters*. In addition to supervised and unsupervised approaches, some methods make use of *semi-supervised* or *reinforcement learning*. The former refers to supervised learning problems, where the classes of some training examples are missing. The latter type of approach relies on feedback that is provided to the learning method to improve its predictions successively over time. All methods that are proposed in this work rely on the principles of supervised learning, i.e., they require labeled training data to be available. Hence, we restrict ourselves to this learning method in the following.

Symbolic vs. Statistical Methods

Among the supervised learning approaches that are commonly used to tackle classification problems, one may distinguish between *statistical* and *symbolic* methods. The former uses statistical optimization techniques to determine the parameters of a predictive function. Examples include *artificial neural networks*, *support vector machines*, or *logistic regression*. Textbooks that focus on this line of research include those by Bishop (1995), Duda, Hart, and Stork (2000), Ripley (2007), Hastie, Tibshirani, and J. H. Friedman (2009), or Goodfellow, Bengio, and Courville (2016). Symbolic learning methods rely on symbolic descriptions to represent learned concepts and capture knowledge about a problem domain. For example, in *decision tree learning* or *rule learning*, models are typically represented in terms of logical "if"-"then"-clauses that test for the properties of given examples to determine a prediction. Books that are concerned with this area have for example been published by Mitchell (1997), Langley (1996), or Fürnkranz, Gamberger, and Lavrač (2012). The algorithms and techniques that are discussed in this thesis are mostly related to symbolic learning methods. Consequently, in the remainder of this chapter, we focus on two well-established representatives of this particular concept type, namely decision trees and rule learning algorithms.

2.1 Decision Trees

Structure of a Decision Tree

Decision trees are symbolic classification models organized in a hierarchical, flowchart-like structure. They consist of *nodes* that are connected to two or more successors via *edges*, forming a directed, non-cyclic graph. The top-most node of a decision tree, which does not have any predecessors, is referred to as the *root node*. Nodes that are located at the lowest level of the tree and therefore do not have any successors are called *leaves*. Each of the internal nodes is concerned with a single attribute present in the training data, whereas the leaves are labeled with one of the available classes. Given an example to be classified, a decision tree is traversed from top to bottom, starting at the root node. At each node, the example's value for the respective attribute is compared to several constants that correspond to the node's outgoing edges. Depending on which edge is satisfied by the value at hand, the example is propagated to one of the node's successors. The traversal of the tree is continued until a leaf is finally reached and the example is assigned its class label. Figure 2.1 depicts the typical structure of a decision tree. It has been learned by applying the *J48* decision tree learner, which is provided as part of the WEKA (Hall et al., 2009) machine learning library, to a variant of the aforementioned "Mushroom" dataset with binary classes.

Growing a Decision Tree

Decision trees are usually constructed in a top-down fashion, starting with a tree that consists of a single node. The tree is recursively refined by following a breadth-first or depth-first strategy, where existing leaf nodes are replaced with new internal nodes that split up the training examples that belong to the original leaves. Introducing such a split into an existing tree structure requires selecting an attribute the new node should be concerned with and deciding on the conditional tests corresponding to its outgoing edges. In the case of numerical attributes, which come with arbitrary numerical values, relational operators like

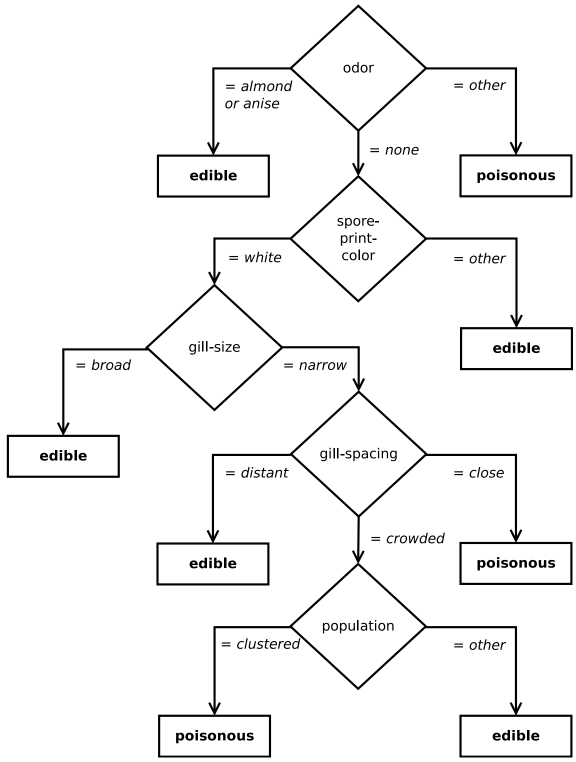


Figure 2.1: Exemplary decision tree that results from applying the J48 algorithm to the dataset "Mushroom". Internal nodes are represented by trapezoids, whereas rectangular shapes are used for leaves. For simplicity, edges that lead to leaf nodes with identical class labels have been merged and given the label "other".

$<$, $\leq$, $>$ or $\geq$ can be used. Nominal attributes, which are restricted to a predefined set of discrete values, are typically dealt with using operators like $=$ or $\neq$. Among all nodes that can potentially be added to a decision tree, the best one is chosen according to a *split criterion* that takes into account the changes in predictive accuracy that result from modifying the tree structure accordingly. Existing algorithms for constructing decision trees differ in the split criterion they use. Among the most popular approaches are the algorithms *ID3* (Quinlan, 1986), as well as its extension *C4.5* (Quinlan, 1993). Their split criterion, which is commonly known as *information gain*, is based on information-theoretic entropy. An implementation of C4.5, referred to as J48, was used to learn the decision tree in Figure 2.1. Another popular decision tree learner is *CART* (Breiman et al., 1984), which assesses the quality of potential splits in terms of the *Gini index*.

Over- and Underfitting

On the one hand, a simple decision tree with very few nodes is often too general because a broad separation of examples does not allow to provide accurate predictions. Such a tree is said to *underfit* the data. On the other hand, introducing too many branches into the tree structure, which in the extreme case leads to a scenario where a leaf is created for each one of the training examples, may result in *overfitting*. In such a case, even though it delivers accurate predictions for the training examples, the resulting decision tree must be expected to be prone to noise in the data. Consequently, it likely performs badly when provided with unseen examples. In order to perform well on unseen data, it is crucial to strive for a good balance between the simplicity and complexity of a tree-based model (Mitchell, 1997). For this reason, successful decision tree learners like C4.5 or CART employ *pruning* techniques that eliminate nodes from a previously constructed and potentially too specific tree if they turn out to be harmful to the predictive accuracy according to an evaluation of the tree's performance on a previously unused portion of the training data. In addition, many learning algorithms allow imposing restrictions on the growth of decision trees by limiting their maximum depth or specifying the minimum number of training examples that must belong to a single leaf.

2.2 Rule Learning Algorithms

Structure of Rule-based Models

In addition to decision trees, rule-based classification models are another popular approach in symbolic learning with a long history of active research. For example, Fürnkranz, Gamberger, and Lavrač (2012) provide a broad overview of the topic. Both types of models are closely related to each other, as they express domain knowledge in terms of conditional clauses that refer to attributes present in the data. In fact, each decision tree can be transformed into an equivalent rule-based model by viewing the paths in a tree, from the root node to each one of its leaves, as individual *rules*. Each one of these rules consists of a *body* and a *head*. Whereas the former consists of a set of *conditions* that correspond to the nodes in a decision tree, the latter provides a prediction in the form of a class label, similar to the leaves in a tree-based model. In accordance with existing work on the topic, we use the notation

$$Head \leftarrow Body$$

for the representation of rules in this work. Conjunctive rules, where the body is given as a conjunction of several conditions, are used most commonly in the rule learning literature. Nevertheless, there are also approaches that make use of both, logical OR ($\vee$) and AND ($\wedge$) operators, for the concatenation of conditions in a rule's body (Michalski, 1980; Theron and Cloete, 1996). Table 2.1 illustrates the typical structure of a rule-based classification model that consists of several conjunctive rules. The depicted model has been obtained by applying the rule learning algorithm *JRip*, a Java implementation of RIPPER (Cohen, 1995) that is provided by the WEKA (Hall et al., 2009) project, to the "Mushroom" dataset from the UCI machine learning repository.

Decision Trees vs. Rule Models

If an example satisfies the conditions in a rule's body, it is said to be *covered* by the rule. In such a case, the prediction that is provided by the rule's head applies to the example. Depending on its conditions, a rule covers an axis-aligned, hyper-rectangular region of the *feature space*, i.e., the space of all possible examples. In contrast to decision trees, the individual

Table 2.1: Exemplary rule model, consisting of several conjunctive rules, as well as a default rule, that results from applying the JRip algorithm to the dataset "Mushroom". If an example is covered by one the conjunctive rules, it assigned to the class "poisonous". If none of these rule cover an example, the default rule predicts the class "edible" instead.

poisonous	$\leftarrow$	odor = *foul*
poisonous	$\leftarrow$	gill-size = *narrow* ∧ gill-color = *buff*
poisonous	$\leftarrow$	gill-size = *narrow* ∧ odor = *pungent*
poisonous	$\leftarrow$	odor = *creosote*
poisonous	$\leftarrow$	spore-print-color = *green*
poisonous	$\leftarrow$	stalk-surface-above-ring = *silky* ∧ gill-spacing = *close*
poisonous	$\leftarrow$	habitat = *leaves* ∧ cap-color = *white*
poisonous	$\leftarrow$	stalk-color-above-ring = *yellow*
edible	$\leftarrow$	$\emptyset$

rules in a rule-based model must not necessarily be non-overlapping. Moreover, many rule learning approaches use a *default rule* that does not contain any conditions in its body and therefore applies to all examples. It is meant to provide a default prediction, usually corresponding to the majority class, i.e., the class that occurs most often in the training data, for examples not covered by another rule. Due to these differences, even though decision trees can always be transformed into a set of equivalent rules, not all rule-based models can be encoded as a tree-like structure. As a consequence, despite their many similarities and even though many techniques from decision tree learning can be transferred to rule learning approaches and vice versa, rules can be considered a more general concept class than the commonly used decision trees and provide additional flexibility when it comes to the selection of rules to be included in a model.

Descriptive vs. Predictive Rules

When it comes to techniques and algorithms for rule induction, one needs to distinguish between *descriptive* and *predictive rule learning*. The former is used to describe learning techniques aimed at discovering interesting patterns in unlabeled data. Prominent examples include the algorithms *Apriori* (Rakesh Agrawal, Imieliński, and Swami, 1993), FP-Growth (Han, Pei, and Yin, 2000) and their numerous variants that allow extracting frequent patterns of co-occurring feature values, also referred to as *association rules*, from a given dataset. In contrast, predictive rule learning methods aim to derive a model from labeled training data, which can be used for prediction afterward. As previously mentioned, we are primarily concerned with classification in this work. Hence, we elaborate on the most common techniques for the induction of predictive rules in the following.

Covering Algorithms

One of the most commonly used strategies for the induction of predictive rules is the *separate-and-conquer* (SeCo) paradigm as described by Fürnkranz (1999). Methods that are based on this particular covering algorithm learn a set of rules by following an iterative procedure. At first, a new rule that covers a subset of the given training examples is induced. Afterward, the examples it covers are removed from the training set, and the algorithm proceeds by learning the next rule. Ultimately, the training procedure finishes as soon as no examples are left or if a certain *stopping criterion* is met, as elaborated on below. The SeCo strategy results in an ordered list of rules, commonly refered to as a *decision list*. Because it was learned on a subset of the training data, where examples covered by previously induced rules have already been removed, each rule in the list depends on its predecessors. Given an example to predict for, the dependence between rules is taken into account by processing the rules in a decision list in the order of their induction. The first rule that covers an example is responsible for its prediction. If an example is not covered by any rule, the default rule takes effect instead. A generalization of the SeCo paradigm, referred to as *weighted covering*, was independently proposed by Weiss and Indurkhya (2000) and Gamberger and Lavrač (2000). Instead of removing examples entirely once they have been covered, weighted covering reduces the weights of individual examples whenever an additional rule covers them. As a consequence, examples that one or more rules have already covered have a smaller impact on the evaluation of new rules. Uncovered examples have a greater chance to be covered in subsequent iterations of the algorithm instead. An advantage of weighted covering is the reduced impact of rules learned during the early stages of the covering algorithm. In addition, it usually results in more rules being learned because each example is allowed to be covered several times. Similar to ensemble methods, which are discussed in Section 2.3 below, the combination of predictions provided by many rules tends to produce better classification results. However, as an example may be covered by multiple rules with varying class labels in their heads, weighted covering demands for a prediction scheme that allows resolving conflicts between contrary rules, e.g., by employing a voting mechanism.

Rule Learning Heuristics

Similar to the construction of decision trees, individual rules are usually built in a top-down fashion, where the rule is successively refined by adding new conditions to its initially empty body. As a result of adding a new condition, the rule covers fewer examples and becomes more specific. Principally, bottom-up approaches, where a specific rule with predetermined conditions is successively generalized, are also feasible. However, they are less effective in practice, according to Fürnkranz (2002). As discussed in Section 2.1, in the context of decision trees, it is essential to strive for a balance between too general and overly specific rules to avoid the problem of under- or overfitting the data (Janssen and Fürnkranz, 2008). It is crucial to use a suitable evaluation measure to compare the quality of potential refinements to achieve a good trade-off between the *coverage* and *consistency* of a rule. In the rule learning literature, measures that guide the rule refinement process are often referred to as *heuristics*. Traditionally, the development, analysis, and empirical evaluation of reliable heuristics have played an important role in research on predictive rule learning (see, e.g., Fürnkranz and Flach, 2005; Fürnkranz, Gamberger, and Lavrač, 2012; Janssen and Fürnkranz, 2010), and a large number of different heuristics have consequently been proposed in the past. On the one hand, this includes heuristics like the *Laplace* metric used by *CN2* (Clark and Boswell, 1991), which assesses the quality of a rule in terms of how many examples it covers and how accurately models their true classes. On the other hand, heuristics like FOIL's information gain (Quinlan, 1990) evaluate the possible refinements of a rule relative to its predecessor. In addition, there are also parameterizable heuristics, such as the F-measure (Rijsbergen, 1979) or the M-estimate (Cestnik, 1990). They allow for an explicit trade-off between coverage and consistency by adjusting a user-configurable parameter.

Pruning techniques

Despite sophisticated rule learning heuristics, rule-based learning approaches are prone to overfitting when applied to domains with a significant amount of noise in the data. Successful rule learning methods often employ pruning techniques to counteract this problem (Fürnkranz, 1997). On the one hand, this includes *pre-pruning* methods that are most often implemented in the form of stopping criteria. Their goal is to terminate the training process as soon as adding new rules to a model likely results in overfitting. For example, the stopping criterion of CN2 (Clark and Boswell, 1991) employs a statistical significance test to prevent the induction of unreliable rules. In contrast, FOIL (Quinlan, 1990) and RIPPER (Cohen, 1995) rely on the information-theoretic *minimum description length* (MDL) principle. On the other hand, pruning techniques prevent the construction of overly specific rules by removing too restrictive conditions from their bodies. In contrast to decision tree learning, where *post-pruning* techniques are most commonly used to improve the reliability of a model once its construction has been completed, the use of *incremental reduced error pruning* (IREP) has proved to be an effective tool for overfitting avoidance in rule-based learning (Fürnkranz and Widmer, 1994). Its basic idea is to learn an overly specific rule on a random fraction of the available training examples and then generalize it afterward by removing trailing conditions that are considered harmful to the rule's predictive accuracy according to an evaluation on previously unused examples. Most notably, the principles of IREP are employed by RIPPER (Cohen, 1995), which is one of the most competitive rule learning algorithms available today.

Interpretability

As argued in the introduction of this work, many safety-critical application domains, where unexpected behavior may lead to life-threatening situations or economic loss, demand human-interpretable machine learning models that can be inspected and verified by domain experts (see, e.g., M. Du, N. Liu, and X. Hu, 2019; Molnar, Casalicchio, and Bischl, 2020; Murdoch et al., 2019, for surveys on interpretable machine learning). Symbolic classification methods, including decision trees and rule-based models, are often considered to meet these requirements due to the representation of domain knowledge in terms of simple conditional statements humans can easily understand. However, the latter are often preferred over the former in applications that demand verification by domain experts because even pruned decision trees, due to their restriction to non-overlapping branches, tend to become quite complex in noisy domains (Boström, 1995). Moreover, identical subtrees often occur within a single decision tree due to the fragmentation resulting from a separation into non-overlapping regions (Pagallo and Haussler, 1990). As the simplicity of models is generally considered an important requirement for their interpretability, such practical issues that may result in unnecessarily complicated models are often considered disadvantageous. However, even though large models are arguably harder to comprehend by humans, other factors that may affect the plausibility and justifiability of a model have recently been debated (Fürnkranz, Kliegr, and Paulheim, 2020). As another trend in the literature on interpretable machine learning, rule-based models have increasingly received attention for their ability to make the decision-making process of complex statistical methods transparent. Existing approaches are either based on converting entire black-box models into rule-based representations that can be analyzed by humans (e.g., Zilke, Loza Mencía, and Janssen, 2016) or utilize rules to provide local explanations of their predictions (e.g., Ribeiro, S. Singh, and Guestrin, 2016). This illustrates another attractive property of rule models. Whereas a collection of rules provides a global model, individual rules can be viewed as local explanations of the examples they cover (Fürnkranz, 2005). However, when using a learning approach like the SeCo strategy mentioned above, which results in an ordered list of rules, each rule must be interpreted in the context of its predecessors. This is also highlighted by Lakkaraju, Bach, and Leskovec (2016), who argue that unordered sets of rules are more interpretable than decision lists, as they can be decomposed into individual local patterns. In conclusion, even though rule-based models are generally accepted as one of the most interpretable model classes, their suitability for a manual analysis depends on the particularities of a specific learning approach. Moreover, as discussed in Section 2.3 below, models that are restricted to few rules are often outperformed by larger models that result from ensemble methods. Due to these limitations and the increasing demand for interpretable machine learning methods, the rule learning discipline, after its advent in the 1980s and 90s, has again grown in relevance and scientific interest in recent years.

2.3 Ensemble Methods

Bias-Variance Trade-off

Decision trees and rule learners, especially when used without any pruning techniques, are generally considered as learning approaches with high *variance*. This means that they are very sensitive to small changes in the training data, which may result in drastic changes in the constructed model. On the one hand, this flexibility in model structure causes the mentioned classification methods to be unstable, as their predictions can easily be affected by minor perturbations of the training inputs. On the other hand, it results in low *bias*, i.e., it ensures that the learner can flexibly adjust to different data distributions and can model complex decision boundaries. The bias and variance of a learning method, as defined by Breiman (1996), are closely related to the risk of under- or overfitting the data. Methods that balance these aspects well are expected to deliver more reliable yet accurate predictions. A well-known technique that helps to reduce the variance of a classifier while at the same time increasing its bias is the use of *ensemble methods* that combine the predictions of several unstable classifiers.

Random Forests

The *random forest* (Breiman, 2001) method is an effective approach to classification problems that illustrates the basic principles of ensemble methods very well. It aims at learning ensembles of unpruned decision trees, usually consisting of 100 or more trees, that we refer to as *ensemble members*. The individual trees are constructed on different subsets of the available training examples drawn with replacement, i.e., a single example may occur multiple times within a single subset. In addition, randomization is used to construct the individual trees by restricting the possible spits at each of their internal nodes to random subsets of the available attributes. This does not only allow for a much faster construction of ensemble members, but also leads to a more diverse collection of trees. Because each tree is forced to use different attributes for modelling the provided data, the trees in a random forest can be viewed as alternative solutions to a given classification problem. Due to the absence of pruning techniques, the individual decision trees do not generalize well to unseen data. However, the combination of their predictions via majority voting results in accurate predictions that typically outperform those of single decision trees. Moreover, because the overall predictions of a random forest are less susceptible to changes in the training input, this particular classification method usually provides convincing results out-of-the-box, without the need for excessive parameter tuning.

Boosting Methods

Another popular classification approach, which is based on the idea to incorporate several unstable classifiers in an ensemble, is commonly referred to as *boosting*. Its success in the machine learning community originates from the highly influential *AdaBoost* (Freund and Schapire, 1997) algorithm. As ensemble members, AdaBoost traditionally uses *decision stumps*, i.e., decision trees that are restricted to a single level. They are built by following a sequential procedure, where the predictions of previously learned members affect the construction of their predecessors. The basic idea is to adjust the weights of the training examples depending on whether the existing ensemble members accurately deal with them. Examples for which the existing decision stumps already provide accurate predictions are assigned smaller weights, whereas more emphasis is placed on mispredicted examples. The decision stump that is constructed at a particular iteration of the training algorithm is obliged to focus on examples that are insufficiently handled by its predecessors and therefore is meant to rectify their predictions. There is a close connection between the basic principles of boosting and the adjustment of weights as performed by rule learning algorithms based on weighted covering (c.f., Section 2.2). However, rather than reducing the weights of examples, depending on how often they are covered, boosting algorithms follow a statistically well-justified framework for computing the updates of weights. They employ a *loss function* to measure the overall quality of an ensemble's predictions at each training iteration and weigh the individual training examples according to their impact on the overall performance. This enables to guide the construction of new ensemble members such that the resulting model ultimately optimizes the given loss function. An early attempt to utilize boosting techniques for the induction of rules by extending the methodology of RIPPER (Cohen, 1995) is *SLIPPER* (Cohen and Singer, 1999). Later, a more general framework that incorporates different kinds of boosting-based rule learners, including SLIPPER as a special case, was proposed in the form of ENDER (Dembczyński, Kotłowski, and Słowiński, 2010). Even though boosting techniques have received early attention in the rule learning community, they are nowadays more commonly used to construct tree-based models. Implementations of *gradient boosted decision trees*, such as XGBoost (T. Chen and Guestrin, 2016) or LightGBM (Ke et al., 2017), use unpruned decision trees as ensemble members. They are among the strongest classification methods available today and are used successfully in numerous application domains.

3 Foundations of Multi-label Classification

In this chapter, we discuss the fundamentals of multi-label classification. In addition to other classification tasks, such as binary and multi-class classification, this particular kind of classification problem has become an established topic in the machine learning community. With regard to the large number of publications this field of research has received in recent years, we focus on existing work that is closely related to the concepts discussed in the remainder of this thesis. We start by providing a formal definition of multi-label classification problems in Section 3.1, which is accompanied by a discussion of real-world applications and datasets in Section 3.2. In Section 3.3, we introduce multi-label evaluation measures that are commonly used in the literature to assess the performance of predictive models. The varying characteristics of these measures are an important aspect of research on multi-label classification. Hence, in Section 3.5, we discuss to what extent they benefit from the ability of learning algorithms to take dependencies between labels into account. Finally, we conclude this chapter by providing an overview of the capabilities and limitations of commonly used approaches to multi-label classification in Section 3.6, Section 3.7, and Section 3.8.

3.1 Problem Definition

Single- vs. Multi-label Classification

In machine learning, classification refers to tasks that require the assignment of objects to classes. Among the machine learning methods that deal with such problems, binary and multi-class classification have the longest history of active research. In the remainder of this work, we refer to these traditional classification settings as *single-label problems*. Algorithms that are aimed at these particular types of classification problems should be capable of assigning objects, usually referred to as examples or instances, to one out of two or several mutually exclusive classes (cf. Chapter 2). E.g., the examples could be text documents, which a classifier should automatically assign to one out of several predefined topics. Classification approaches based on supervised learning aim to deduce a predictive model from a limited set of labeled training examples for which the true classes are known. A good model should generalize well beyond the provided observations, such that it can be used to make predictions for unseen examples. This is contrast to unsupervised and semi-supervised learning, where no labeled examples are provided to the classifier beforehand.

Label Space

Unlike in binary or multi-class classification, where an example always corresponds to a single class, in multi-label classification (MLC), a single example may be associated with several class labels at the same time. For example, in the field of text classification, a text document can often not be assigned to a single category unambiguously but may belong to multiple topics, such as "Politics" and "Economy", simultaneously.

Hence, the goal of a multi-label classifier is to assign a particular example to an arbitrary number of labels λ_k out of a predefined and finite labelset $\mathcal{L} = \{\lambda_1, \ldots, \lambda_K\}$.[1] We specify the labels that are associated with an example in the form of a binary label vector

$$\boldsymbol{y} = (y_1, \ldots, y_K) \in \mathcal{Y}, \tag{3.1}$$

where each element $y_k \in \{0, 1\}$ indicates whether the k-th label is irrelevant ($y_k = 0$) or relevant ($y_k = 1$) to the example.[2] The label space $\mathcal{Y} = \{0, 1\}^K$ denotes all possible labelings. The methods that are discussed in the remainder of this work require the label vectors of individual examples to be fully specified. We do not consider approaches that deal with partially labeled data (see, e.g., Xie and S.-J. Huang, 2018).

1: Binary classification can be considered as a special case of multi-label classification where only a single label is available, i.e., where $|\mathcal{L}| = 1$.

2: If $\sum_k y_k = 1$, a multi-label classification task simplifies to a multi-class problem with K mutually exclusive classes.

Feature Space

In this work, we deal with structured tabular data, where each example can be represented by a feature vector

$$\boldsymbol{x} = (x_1, \ldots, x_L) \in \mathcal{X} \tag{3.2}$$

that assigns constant values to *numerical*, *ordinal*, or *nominal* attributes or features A_l out of a predefined set $\mathcal{A} = \{A_1, \ldots, A_L\}$ inherent to the application domain at hand. $\mathcal{X} \in \mathcal{R}^L$ denotes the feature space that consists of all possible feature vectors. A feature value that corresponds to a numerical attribute may be any positive or negative real number, e.g., a temperature. In the case of ordinal attributes, the values are restricted to a predefined finite set. Such categorical values, e.g., temperatures specified as either "cold", "warm" or "hot", are usually encoded by enumerating the available categories in a meaningful order. In contrast, the categorical values of nominal attributes are not subject to any order. E.g., no meaningful order can be imposed on boolean values like "true" and "false". In the course of this work, we also discuss means to deal with missing feature values, i.e., datasets where individual elements in an example's feature vector are unspecified.

Multi-label Classifiers

The methods discussed in this work treat multi-label classification as a supervised learning problem, i.e., a model is fit to the examples in a labeled training data set

$$\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) \mid 1 \leq n \leq N\} \subset \mathcal{X} \times \mathcal{Y} \tag{3.3}$$

for which the true labelings, referred to as the *ground truth*, are known. The goal is to learn a model $f : \mathcal{X} \rightarrow \mathcal{Y}$ that maps from the feature space to the label space. A model of this kind can be considered as a classification function that provides a prediction $\hat{\boldsymbol{y}} = f(\boldsymbol{x})$ for any given example. In the following, we will denote a binary label vector that is predicted by a multi-label classifier as

$$\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_K) \in \mathcal{Y}. \tag{3.4}$$

Rankings and Probability Estimates

Instead of providing a prediction in the form of a binary label vector, MLC classifiers may also deliver a ranking of the available labels, where the labels are ordered by their relevance for a particular example. Alternatively, they may assess the relevance or irrelevance of individual labels in terms of probabilities. Such representations can subsequently be turned into a binary prediction, e.g., by applying a threshold that separates relevant labels from irrelevant ones. However, if not stated otherwise, we assume a classifier's predictions to be given in the binary form.

3.2 Applications and Datasets

Dateset Repositories

Experiments are usually conducted on a series of benchmark datasets to empirically evaluate an MLC method's predictive performance and compare the results to those of competing approaches. Several collections of datasets from different real-world application domains are publicly available, e.g., the datasets provided by the MEKA (Read, Reutemann, et al., 2016) project[3] , the MULAN (Tsoumakas, Spyromitros-Xioufis, et al., 2011) project[4] , the LIBSVM (C.-C. Chang and C.-J. Lin, 2011) project[5] or those included in the MLDA (Moyano, Gibaja, and Ventura, 2017) tool for analyzing multi-label datasets.[6] A selection of datasets from the mentioned sources, which are used for experiments throughout the remainder of this work, is shown in Table 3.1. All of these datasets can also be retrieved from a publicly available repository.[7]

3: `https://waikato.github.io/meka/datasets`

4: `http://mulan.sourceforge.net/datasets-mlc.html`

5: `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`

6: `https://www.uco.es/kdis/mllresources`

7: `https://github.com/mrapp-ke/Boomer-Datasets`

Text Classification

Early approaches to multi-label classification mostly evolved around the problem of text classification (see, e.g., Sebastiani, 2002, for an early publication on the topic). Due to the large number of benchmark datasets that belong to this particular application domain, the automated annotation of text documents with topics, authors, emotions, or the like remains one of the most relevant MLC settings. The types of documents to be dealt with by a text classification algorithm may vary. In the past, such algorithms have been applied to news articles (Lang, 2008; Lewis et al., 2004; Read, 2010), web documents (Read, 2010; Tsoumakas, Katakis, and Vlahavas, 2008; Ueda and Saito, 2003), e-mail conversations (Read, Pfahringer, and Holmes, 2008), legal texts (Loza Mencía and Fürnkranz, 2008), scientific papers (Joachims, 1998; Katakis, Tsoumakas, and Vlahavas, 2008), as well as medical and technical reports (Pestian et al., 2007; Srivastava and Zane-Ulman, 2005).

Multimedia

Nowadays, MLC methods are also used to deal with diverse multimedia resources, such as images (Boutell et al., 2004; M.-L. Zhang and Z.-H. Zhou, 2007), videos (Snoek et al., 2006), or audio recordings (Briggs et al., 2013; Tsoumakas, Katakis, and Vlahavas, 2008; Turnbull et al., 2008). However, as many classification methods are restricted to relational data, rather than being capable of dealing with highly unstructured data, such as raw images, the data must eventually be converted into a suitable format. For example, Boutell et al. (2004) use numerical features to represent the colors that are predominant in different regions of images.

Biology

Approaches to multi-label classification have also received attention in the field of biology. Prominent examples include the assignment of genes to functional categories (Elisseeff and Weston, 2001) and the prediction of sub-cellular locations of proteins according to their sequences (Xu et al., 2016).

Other Applications

Besides the applications of MLC methods that are mentioned above and are relevant to this work, as we use datasets from the respective domains for experiments, many more use cases exist. For example, this includes chemical data analysis, social network mining, or applications in marketing. For a more comprehensive list of real-world applications, we refer to survey articles on the topic, such as those published by Gibaja and Ventura (2014) or M.-L. Zhang and Z.-H. Zhou (2014).

Data Dimensionality

In binary and multi-class classification, the amount of data that must be processed by a supervised learning approach mainly depends on the number of training examples and attributes. In multi-label classification, where each example may be associated with several labels rather than a single class, the dimensionality of the input data is also heavily affected by the number of available labels. Table 3.1 provides an overview of the dimensionality of selected benchmark datasets that are relevant to this work. Whereas some of the datasets consist of thousands of examples or features, we restrict ourselves to datasets with up to a few hundred labels. This is a deliberate choice even tough research on MLC has increasingly shifted towards *extreme multi-label classification* (XMLC), where enormous amounts of data with thousands or even millions of labels should be handled (see, e.g., Rahul Agrawal et al., 2013; Deng et al., 2010; Denton et al., 2015; Prabhu, Kag, et al., 2018; Prabhu and Varma, 2014; Prajapati and Thakkar, 2019, for applications and datasets).[8] Approaches to XMLC are usually based on projections from a high-dimensional problem domain to a domain with reduced complexity, which requires strong assumptions about the data. Rather than restricting ourselves to a particular scenario with unique requirements, we aim at a more general investigation of the capabilities and limitations of rule-based approaches considering a broad range of applications and datasets with varying characteristics. For completeness, we provide an overview of projection methods suitable for XMLC in Section 3.8. As they can often be used with arbitrary black-box classifiers, they can potentially be combined with the rule learning approaches discussed in the following chapters. However, we consider this particular direction of research to be out of the scope of this work.

8: For example, benchmark datasets for extreme multi-label classification can be obtained from `http://manikvarma.org/downloads/XC/XMLRepository.html`.

Feature and Label Sparsity

To gain a better impression of the benchmark datasets used in this work, Table 3.2 provides an overview of metrics that are commonly used to characterize multi-label data. First of all, we consider the *feature* and *label sparsity*, i.e., the percentage of feature values and ground truth labels that are equal to zero. These two metrics calculate as

$$\mathcal{S}_X(\mathcal{D}) := \frac{\sum_{n=1}^{N} \sum_{l=1}^{L} [\![x_{nl} = 0]\!]}{NL} \text{ and } \mathcal{S}_Y(\mathcal{D}) := \frac{\sum_{n=1}^{N} \sum_{k=1}^{K} [\![y_{nk} = 0]\!]}{NK}, \tag{3.5}$$

where $[\![x]\!]$ evaluates to 1 or 0 if the predicate x is true or false, respectively. In many cases, especially when dealing with text classification datasets, the feature values of the individual examples are often equal to zero.[9] A similar observation can be made regarding the label sparsity, which is typically high, regardless of the application domain. This illustrates the tremendous potential of learning algorithms that are capable of dealing with sparsity in the feature or label space in an efficient way. In Chapter 7, we discuss ways to exploit such sparsity to reduce the time that is needed to train rule-based MLC models.

9: Text classification datasets often come with binary features that indicate the presence (1) or absence (0) of keywords in individual text documents. As most documents contain only a small fraction of the available keywords, most feature values are equal to zero.

Label Imbalance

The sparsity in ground truth labels that is characteristic of many multi-label datasets also imposes challenges on learning algorithms. Besides labels that occur more frequently, a multi-label dataset typically includes many rare labels that are only relevant to a small fraction of the available

Table 3.1: Dimensionality of selected benchmark datasets from different domains (sorted alphabetically by their names), including the number of examples, numerical and nominal attributes and labels.

Dataset	Examples	Attributes		Labels
		Numerical	Nominal	
20NG (Lang, 2008)	19,300	1,006	0	20
Bibtex (Katakis, Tsoumakas, and Vlahavas, 2008)	7,395	0	1,836	159
Birds (Briggs et al., 2013)	645	258	2	19
Bookmarks (Katakis, Tsoumakas, and Vlahavas, 2008)	87,860	0	2,150	208
CAL500 (Turnbull et al., 2008)	502	68	0	174
Delicious (Tsoumakas, Katakis, and Vlahavas, 2008)	16,110	0	500	983
Emotions (Tsoumakas, Katakis, and Vlahavas, 2008)	593	72	0	6
Flags (Gonçalves, Plastino, and Freitas, 2013)	194	10	9	7
Genbase (Diplaris et al., 2005)	662	0	112	27
Enron (Read, Pfahringer, and Holmes, 2008)	1,702	0	1,001	53
EukaryoteGO (Xu et al., 2016)	7,766	12,690	0	22
EukaryotePseAAC (Xu et al., 2016)	7,766	440	0	22
EUR-Lex-SM (Loza Mencía and Fürnkranz, 2008)	19350	5,000	0	201
Image (M.-L. Zhang and Z.-H. Zhou, 2007)	2,000	294	0	5
IMDB (Read, 2010)	120,900	1,001	0	28
Langlog (Read, 2010)	1,460	1,004	0	75
Mediamill (Snoek et al., 2006)	43,910	120	0	101
Medical (Pestian et al., 2007)	1,954	1,909	0	45
Nus-Wide cVLADplus (Chua et al., 2009)	269,648	128	1	81
Ohsumed (Joachims, 1998)	13,930	1,002	0	23
Reuters-K500 (Lewis et al., 2004)	6,000	500	0	103
Scene (Boutell et al., 2004)	2,407	294	0	6
Slashdot (Read, 2010)	3,782	3,125	0	22
TMC2007 (Srivastava and Zane-Ulman, 2005)	28,600	0	49,060	22
Yahoo-Computers (Ueda and Saito, 2003)	12,444	34,096	0	33
Yahoo-Reference (Ueda and Saito, 2003)	8,027	39,679	0	33
Yahoo-Science (Ueda and Saito, 2003)	6,428	37,187	0	40
Yahoo-Social (Ueda and Saito, 2003)	12,111	52,350	0	39
Yeast (Elisseeff and Weston, 2001)	2,417	103	0	14

examples. We use the *label imbalance ratio* (Charte et al., 2013) to measure the degree of imbalance across the labels in a dataset as

$$\mathcal{IR}(\mathcal{D}) = \frac{1}{K} \sum_{k=1}^{K} \frac{\arg\max_{i \in [1,K]} \left(\sum_{n=1}^{N} y_{ni} \right)}{\sum_{n=1}^{N} y_{nk}}. \tag{3.6}$$

Figure 3.1 shows the frequencies of labels in a typical multi-label dataset and the imbalance ratio for each label. It can be seen that the vast majority of labels occur far less often than the most frequent ones. In order to perform well, a classifier must be able to accurately identify the examples to which these rare labels belong without considering them in cases where they are irrelevant. Rare labels also require special treatment when sampling from multi-label data, which is an essential aspect of many machine learning techniques.

Distinct Labelsets

We also characterize multi-label datasets regarding the number of *distinct labelsets* (Tsoumakas, Katakis, and Vlahavas, 2009) they entail. This metric, which corresponds to the number of unique label vectors $\boldsymbol{y}_n$ in a dataset, is especially relevant when using classification methods that do not consider each label independently but aim to model the relevance and

Table 3.2: Characteristics of selected benchmark datasets (sorted alphabetically by their names), including the percentage of sparse feature values and ground truth labels, the average label imbalance ratio, the average label cardinality and the number of distinct labelsets.

Dataset	Feature Sparsity	Label Sparsity	Avg. Label Imbalance Ratio	Label Cardinality	Distinct Labelsets
20NG	96.81	94.86	1.01	1.03	55
Bibtex	96.26	98.49	12.50	2.40	2,856
Birds	38.64	94.66	5.41	1.01	133
Bookmarks	94.16	99.02	12.31	2.03	18,716
CAL500	0.15	85.03	20.58	26.04	502
Delicious	96.34	98.07	71.13	19.02	15,806
Emotions	0.33	68.86	1.48	1.87	27
Flags	59.20	51.55	2.26	3.39	54
Genbase	22.63	95.36	37.32	1.25	32
Enron	91.60	93.63	73.95	3.38	753
EukaryoteGO	99.86	94.79	45.01	1.15	112
EukaryotePseAAC	43.37	94.79	45.01	1.15	112
EUR-Lex-SM	95.26	98.90	536.98	2.21	2,504
Image	0.22	75.28	1.19	1.24	20
IMDB	98.06	92.86	25.12	2.00	4,503
Langlog	81.38	98.43	39.27	1.18	304
Mediamill	0.00	95.67	256.40	4.38	6,555
Medical	99.32	97.24	89.99	1.24	94
Nus-Wide cVLADplus	0.00	97.69	95.12	1.87	18,430
Ohsumed	96.03	92.77	7.87	1.66	1,147
Reuters-K500	98.41	98.58	54.08	1.46	811
Scene	1.15	82.10	1.25	1.07	15
Slashdot	99.46	94.10	19.46	1.18	156
TMC2007	99.79	90.19	15.16	2.16	1,341
Yahoo-Computers	99.62	95.43	176.70	1.51	428
Yahoo-Reference	99.59	96.44	461.86	1.17	275
Yahoo-Science	99.53	96.38	52.63	1.45	457
Yahoo-Social	99.71	96.72	257.70	1.28	361
Yeast	0.00	69.74	7.20	4.24	361

irrelevance of entire label subsets. In theory, the number of possible label combinations 2^K grows exponentially with the number of labels K, making such an approach increasingly costly. However, as can be seen in Table 3.2, only a small fraction of the possible labelsets is typically observed in practice and must therefore be taken into account by a classifier.

Label Cardinality

As another result of the sparsity in the ground truth labels that is inherent to many multi-label datasets, the *label cardinality* (Tsoumakas, Katakis, and Vlahavas, 2009) is often relatively small. It corresponds to the average number of labels that are relevant to an example and computes as

$$\mathcal{C}(\mathcal{D}) := \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk}. \tag{3.7}$$

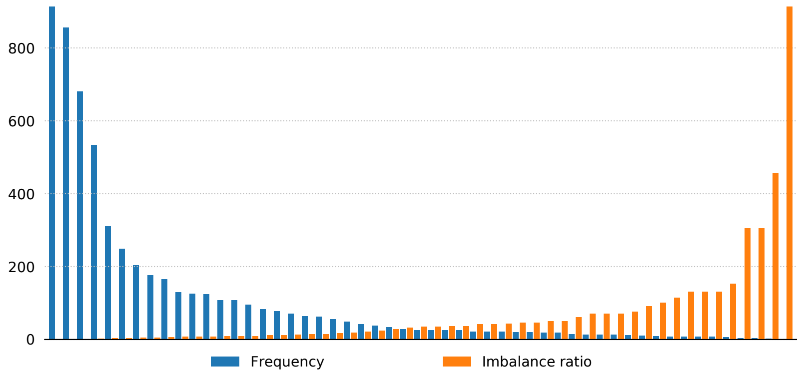


Figure 3.1: Frequencies of the labels in the dataset "Enron", sorted in decreasing order from left to right, as well as the corresponding imbalance ratio for each label.

3.3 Evaluation Measures

Training-Test Split

To compare the predictive performance of different classifiers with each other, evaluation measures are used to assess the quality of predictions in terms of a single score. To obtain an unbiased estimate of a model's quality, labeled examples that have not been provided to the learning algorithm as part of the training data should be used for evaluation. To make it easier to conduct experiments, dataset repositories, such as those mentioned in Section 3.2, often provide datasets in a preprocessed form, where the available examples are split into disjoint training and test sets. Alternatively, *cross validation* (CV) may result in a more reliable estimate of a model's performance. It is based on splitting up a dataset into several non-overlapping subsets of approximately the same size. Each of these sets is then used to evaluate a model trained on the examples from the remaining subsets. The performance estimates for the different models are finally averaged to obtain a single score. In this work, we primarily use 10-fold CV, as depicted in Figure 3.2. To further maintain the distribution of labels in a dataset, Sechidis, Tsoumakas, and Vlahavas (2011) advocate the use of *stratification* to create representative training and test sets from a multi-label dataset. According to their studies, the use of stratification reduces the variance across different CV folds and helps prevent the creation of subsets where no examples are associated with rare labels.

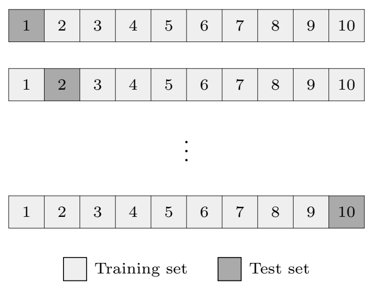


Figure 3.2: Illustration of a 10-fold cross validation, where the training and test sets that are used for the individual folds result from a partition of the dataset into ten equally sized subsets.

Evaluation Functions

In multi-label classification, where the labels that may be associated with an example are not mutually exclusive, many ways to compare a classifier's predictions to the ground truth labels exist. This has lead to the proposal of many evaluation measures with varying characteristics. In this work, we use *bipartition evaluation measures* that compare the set of labels that are predicted to be relevant to the set of relevant labels according to the ground truth. All measures of this kind can be considered functions

$$\mathcal{M}(Y, \hat{Y}) \in \mathbb{R} \text{ with } Y, \hat{Y} \in \{0,1\}^{N \times K} \tag{3.8}$$

that assess the quality of predictions for N examples and K labels in terms of a single score $\gamma \in \mathbb{R}$. By Y, we denote the true labels according to the ground truth, whereas $\hat{Y}$ corresponds to the predicted labels provided by a classifier.

Hamming Loss

Among the most commonly reported measures in empirical studies are adaptations of the 0/1 loss known from binary classification. This includes the *Hamming loss* that measures the fraction of incorrectly predicted labels among all labels. It calculates as

$$\mathcal{M}_{Hamm.}(Y, \hat{Y}) \coloneqq \frac{1}{NK} \sum_{n=1}^{N} \sum_{k=1}^{K} [\![y_{nk} \neq \hat{y}_{nk}]\!], \tag{3.9}$$

where the term $[\![x]\!]$ evaluates to 1 or 0, depending on whether the predicate x is true or false.

Subset 0/1 Loss

Similar to the Hamming loss, the *subset 0/1 loss* can also be considered a generalization of the standard 0/1 loss to multiple labels, albeit in a very different way. Unlike the former, the latter does not consider individual labels in isolation but takes all labels of an example into account at the same time. It corresponds to the fraction of examples for which at least one label is mispredicted. As a false prediction for a single label is penalized as much as predicting several labels of an example incorrectly, the subset 0/1 loss is considered particularly challenging to optimize. It is formally defined as

$$\mathcal{M}_{Subs.}(Y, \hat{Y}) \coloneqq \frac{1}{N} \sum_{n=1}^{N} [\![\boldsymbol{y}_n \neq \hat{\boldsymbol{y}}_n]\!]. \tag{3.10}$$

Limitations of Hamming and Subset 0/1 Loss

Both the Hamming and the subset 0/1 loss have disadvantages when used to evaluate the predictive performance of MLC methods. On the one hand, the Hamming loss is often close to zero, as in many multi-label datasets the label cardinality, i.e., the average number of labels that are relevant to an example, is very small. Many rare labels, which are only relevant to few examples, exist in such a dataset. Consequently, even an overly simple classifier that predicts all labels as irrelevant performs well with respect to these rare labels. In such a case, the Hamming loss indicates high predictive accuracy, and more sophisticated approaches, which can distinguish between relevant and irrelevant labels, only achieve minor improvements in the reported score. On the other hand, the subset 0/1 loss is a very stringent measure that is often close to one. When dealing with datasets that contain a large number of labels, it becomes increasingly difficult to predict all labels correctly. Because the prediction for an example is considered incorrect as a whole, if only a single mistake is made, this particular measure does not discriminate between approaches that predict correctly for most of the labels and methods that perform poorly for all labels.

Confusion Matrices

To compensate for the disadvantages of the Hamming and subset 0/1 loss, and to provide a more differentiated evaluation of a classifier's abilities and shortcomings, additional measures should be included in an empirical analysis. Instead of merely focusing on the (in)correctness of predictions, such measures often differentiate between predictions for relevant and irrelevant labels. Their definition is usually based on two-dimensional confusion matrices that characterize a model's predictions in terms of the number of *true positives* (TP), *false positives* (FP), *true negatives* (TN), and *false negatives* (FN). Given predictions for N examples and K labels, one obtains $N \cdot K$ binary confusion matrices C_{nk} according to Table 3.3. The binary confusion matrices may be aggregated via element-wise addition to assess the quality of the predictions in terms of a single score, and the scores that are computed based on an aggregated confusion matrix may be averaged. Depending on the order of aggregation and averaging, different strategies to obtain an overall performance estimate are possible. To facilitate the notation of evaluation measures, which are based on confusion matrices that have been aggregated in different ways, we define various functions for the aggregation of binary confusion matrices. First of all, this includes a function for aggregating confusion matrices across all available examples and labels according to the formula

Table 3.3: A binary confusion matrix that compares the prediction for a label $\hat{y}$ to the corresponding ground truth y.

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

$$\mathrm{aggr}_{Micro}(Y, \hat{Y}) \coloneqq C_{11} \oplus \cdots \oplus C_{1K} \oplus \cdots \oplus C_{N1} \oplus \cdots \oplus C_{NK}, \tag{3.11}$$

where $\oplus$ denotes the element-wise addition of confusion matrices.

In addition, we also use a function that returns a single confusion matrix for a particular example $\boldsymbol{x}_n$ by aggregating the binary confusion matrices that correspond to individual labels. It is formally defined as

$$\mathrm{aggr}_{Ex.\text{-}w.}(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n) \coloneqq C_{n1} \oplus \cdots \oplus C_{nK}. \tag{3.12}$$

Finally, if we are interested in obtaining an aggregated confusion matrix for a particular label, we use the aggregation function

$$\mathrm{aggr}_{L.\text{-}w.}(\boldsymbol{y}_k, \hat{\boldsymbol{y}}_k) \coloneqq C_{1k} \oplus \cdots \oplus C_{Nk}, \tag{3.13}$$

where $\boldsymbol{y}_k$ and $\hat{\boldsymbol{y}}_k$ correspond to the ground truth and predictions for a particular label λ_k.

Micro Averaging

In combination with a *bipartition metric* $\mathcal{M} : \mathbb{N}^{2\times2} \rightarrow \mathbb{R}$ that takes a previously aggregated confusion matrix as an argument, the functions given above can be used to assess the quality of predictions for several examples and labels in terms of a single score. When using the aggregation function in (3.11), the overall evaluation score computes as

$$\mathcal{M}_{Micro}(Y, \hat{Y}) \coloneqq \mathcal{M}\left(\mathrm{aggr}_{Micro}(Y, \hat{Y})\right). \tag{3.14}$$

We refer to such an evaluation measure, where all examples and labels are considered equally important, as *micro-averaged*.

Example-wise Averaging

When using the aggregation function in (3.12), the evaluation scores that result from applying a bipartition metric $\mathcal{M}$ to the aggregated confusion matrices for individual examples must finally be averaged to obtain a single score. As a classifier usually predicts for given examples independently, such an *example-wise averaged* measure

$$\mathcal{M}_{Ex.\text{-}w.}(Y, \hat{Y}) \coloneqq \frac{1}{N} \sum_{n=1}^{N} \mathcal{M}\left(\mathrm{aggr}_{Ex.\text{-}w.}(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n)\right). \tag{3.15}$$

appears to be a natural choice to assess its predictive performance and is used for most experiments in this work.

Label-wise Averaging

Similarly, a *label-wise averaged* evaluation measure uses the aggregation function in (3.13). In this case, an aggregated confusion matrix is first obtained for each label, and the individual scores that result from applying a bipartition metric $\mathcal{M}$ to each one are finally averaged. Compared to micro- and example-wised averaged measures, which are often dominated by more frequent labels, such an evaluation measure

$$\mathcal{M}_{L.\text{-}w.}(Y, \hat{Y}) := \frac{1}{K} \sum_{k=1}^{K} \mathcal{M}\left(\text{aggr}_{L.\text{-}w.}(\boldsymbol{y}_k, \hat{\boldsymbol{y}}_k)\right) \tag{3.16}$$

is particularly useful if high predictive accuracy is desired for all labels, including the very rare ones.

Precision and Recall

Precision and *recall* are among the bipartition metrics that provide useful information about the predictions of a model. In contrast to the Hamming loss and the subset 0/1 loss, where smaller scores are better, these two metrics should be maximized, i.e., methods that achieve larger scores outperform methods with smaller scores. Given a confusion matrix that has been obtained by using one of the aggregation schemes in (3.11), (3.12) or (3.13), precision is defined as

$$\mathcal{M}_{Prec.}(C) := \frac{TP}{TP + FP}. \tag{3.17}$$

It penalizes cases where a label is incorrectly predicted as relevant (FP) and therefore suffers from predicting relevant labels as irrelevant (FN). Small scores according to this metric indicate that a model tends to mispredict many labels as relevant. Predicting more conservatively, i.e., only predicting labels as relevant if they belong to an example with high probability, improves the precision of a model but typically comes with a deterioration in terms of recall. This is because recall suffers from predicting relevant labels as irrelevant (FN), as can be seen in the formula

$$\mathcal{M}_{Rec.}(C) := \frac{TP}{TP + FN}. \tag{3.18}$$

F1-measure

As precision and recall can both be maximized at the expense of the other, a predictive model should ideally perform well in terms of both of these bipartition measures. The *F1-measure*, which is preferred over precision and recall in this work, strives for such a balance. Given a previously aggregated confusion matrix, it calculates as

$$\mathcal{M}_{F1}(C) := \frac{2TP}{2TP + FP + FN}, \tag{3.19}$$

and corresponds to the equally weighted harmonic mean of precision and recall.[10] Due to the properties of the harmonic mean, poor performance in terms of either precision or recall results in a decline of the resulting F1 score.

10: More general formulations of the measure include a parameter $\beta \in [0, +\infty]$ that allows to control the trade-off between precision and recall (cf. Section 5.2).

Other Measures

Besides the metrics used in this work, many additional measures for evaluating multi-label classifiers can be found in the literature. Besides a wide variety of bipartition measures, this also includes *ranking measures*. This particular family of evaluation measures is useful if a predictive model imposes an order on the labels, e.g., by sorting them according to their probability to be relevant to an example. We refer to survey articles for a complete overview of commonly used bipartition and ranking measures. E.g., Tsoumakas, Katakis, and Vlahavas (2009), Sorower (2010), Gibaja and Ventura (2014), or M.-L. Zhang and Z.-H. Zhou (2014) provide a comprehensive overview of this aspect of multi-label classification.

3.4 Statistical Significance Tests

Comparison Across Multiple Datasets

The evaluation measures discussed in Section 3.3 can be used to assess the quality of multi-label predictions that are provided by a classifier for unseen test data in terms of a numerical score γ. Usually, an empirical investigation of an MLC method's capabilities and limitations demands a comparison of the performance estimates obtained for different datasets $\mathcal{D}_1, \dots, \mathcal{D}_D$ with those of one or several competing approaches. In the literature, tabular or graphical representations are most commonly used to compare the observations $\gamma_{i1}, \dots, \gamma_{iD}$ that result from the application of different classification methods f_i to a particular dataset. In addition, the use of *statistical significance tests* may help to conclude whether individual methods are superior to their competitors. Given a particular approach that tends to outperform another one in a series of experiments, statistical tests determine the probability $1 - p$ that the differences in predictive performance are not produced by chance. The differences are considered *statistically significant* with respect to a *significance level* α if the inequality $p < \alpha$ holds.[11]

11: Commonly used values for the significance level α are 1%, 5% or 10% (Demšar, 2006)

Ranks and Average Ranks

The application of statistical tests often requires to obtain a ranking of competing classification approaches by their predictive performance. Such a ranking may also be reported in scientific publications for a more comprehensive overview of experimental results. For this purpose, we sort the tested classifiers according to a bijective permutation function $\tau_d : \mathbb{N} \rightarrow \mathbb{N}$ that takes their respective performance on a specific dataset $\mathcal{D}_d$ into account. It assigns *ranks* to individual competitors, such that $\tau_d(i) < \tau_d(j), \forall i, j$ if the evaluation score γ_{id} achieved by the i-th classifier is better than the score γ_{jd} that corresponds to the j-th classification method.[12] The *average rank* of a classifier f_i across all D datasets may be computed as

$$r_i = \frac{1}{D} \sum_{d=1}^{D} \tau(i). \tag{3.20}$$

12: For a fair comparison, average ranks should be assigned in the case of ties.

It allows to compare the performance of competing approaches across several datasets.

Friedman Test

Demšar (2006) discusses different statistical tests for the empirical evaluation of machine learning methods. In this work, we rely on the *Friedman test* (M. Friedman, 1937, 1940) to verify whether the evaluation scores achieved by several classifiers in a series of experiments on different datasets are similar. The null hypothesis to be testified or refuted by the Friedman test states that all tested classifiers perform equally, i.e., their evaluation scores follow the same distribution. Under the null hypothesis, if the number of available datasets D and the number of tested classifiers I is sufficiently large[13] , the *Friedman statistic*

13: Demšar (2006) advices to use $D > 10$ and $I > 5$ as a rule of thumb.

$$\chi_F^2 = \frac{12D}{I(I+1)} \left(\sum_{i=1}^{I} r_i^2 - \frac{I(I+1)^2}{4} \right) \tag{3.21}$$

is distributed according to a Chi-squared distribution with $D - 1$ degrees of freedom. Following the suggestion of Demšar (2006), we instead use the less conservative statistic

$$F_F = \frac{(D-1)\,\chi_F^2}{D\,(I-1) - \chi_F^2} \tag{3.22}$$

with $(I-1)\,(D-1)$ degrees of freedom. If the statistic F_F exceeds a certain critical distance, corresponding to a quantile of the Chi-squared distribution, the null hypothesis is rejected. Tables of maximum critical distances, given the degrees of freedom and a desired significance level α, can be found in statistical books or online.[14] In cases where only a few classifiers or datasets are available, exact critical distances, such as the ones provided by Martin, Leblanc, and Toan (1993), should be used.

14: `https://people.richland.edu/james/lecture/m170/tbl-chi.html`

Nemenyi Post-hoc Test

If the null hypothesis of the Friedman test is rejected, we can employ a post-hoc test to identify classifiers that significantly outperform one or several of their competitors. As proposed by Demšar (2006), we use the *Nemenyi test* (Nemenyi, 1963) for this purpose. It states that the predictive performance of two classifiers is significantly different if their average ranks differ by at least the critical distance

$$q_\alpha \sqrt{\frac{I\,(I+1)}{6D}}, \tag{3.23}$$

where q_α corresponds to the Studentized range statistic divided by $\sqrt{2}$. Critical values for q_α are provided by Demšar (2006), depending on the number of classifiers I and the significance level α.

3.5 Label Dependence

Examples of Label Dependencies

In many real-world applications of multi-label classification, dependencies between the available labels can be observed in the data. In such case, the relevance or irrelevance of individual labels correlates with each other. For example, in text classification, where text documents should be annotated with one or several topics, one could observe that the label "foreign affairs" is more likely to be relevant to a document if it belongs to the topic "politics". The other way around, from the relevance of a label, one may conclude that another label is unlikely to be relevant. For example, in text classification, this could be the case if documents annotated with the label "sports" are unlikely to belong to the topic "foreign affairs". The idea of capturing and exploiting correlations, such as co-occurrences and exclusions, between labels to improve the predictive performance of multi-label classifiers has become a driving force for research on multi-label classification. Many MLC methods proposed in recent years, including approaches that are presented in this work, are motivated by this idea.

Marginal and Conditional Dependence

Dembczyński, Waegeman, et al. (2012) provide a more formal view of the different types of stochastic dependencies that may be found in multi-label data. The authors distinguish between *conditional* and *marginal* (unconditional) label dependence. The former refers to dependencies that are locally restricted to a particular region in the feature space. In such a case, the joint distribution of a label vector $\boldsymbol{y}$ is conditioned on an example $\boldsymbol{x}$, i.e.,

$$p(\boldsymbol{y}|\boldsymbol{x}) = p(y_1|\boldsymbol{x}) \times \cdots \times p(y_K|\boldsymbol{x}) . \tag{3.24}$$

In contrast, marginal dependence is independent of any concrete example. Consequently, the joint distribution of a label vector $\boldsymbol{y}$ can be written in terms of marginals $p(y_k)$, i.e., independent of any particular observation $\boldsymbol{x}$, as

$$p(\boldsymbol{y}) = p(y_1) \times \cdots \times p(y_K) . \tag{3.25}$$

Decomposability and Non-decomposability

Furthermore, Dembczyński, Waegeman, et al. (2012) provide strong empirical and theoretical arguments that not only the type of dependencies in a dataset but also the type of evaluation measure to be optimized strongly influence to what extent the exploitation of label correlations can be expected to result in an improvement of predictive performance. In the literature, one usually distinguishes between label-wise *decomposable* and *non-decomposable* evaluation measures. On the one hand, the former can be calculated by applying a suitable single-label evaluation function $\mathcal{M}_k$ to each label individually and aggregating the results, i.e.,

$$\mathcal{M}(Y, \hat{Y}) = \sum_{k=1}^{K} \mathcal{M}_k \left(Y_k, \hat{Y}_k \right) , \tag{3.26}$$

where $Y_k, \hat{Y}_k \in \{0,1\}^{N \times 1}$ denote the ground truth and the predictions for a single label λ_k, respectively. On the other hand, a non-decomposable evaluation measure cannot be expressed in this form. For example, a prominent representative of decomposable measures is the Hamming loss in (3.9). In contrast, the subset 0/1 loss in (3.10) and the F-measure in (3.19), unless applied in a label-wise manner according to (3.16), are non-decomposable evaluation metrics.

Target Measure Optimization

Given that the labels in a dataset are not stochastically independent, the results provided by Dembczyński, Waegeman, et al. (2012) suggest that learning algorithms may benefit from capturing marginal dependencies regardless of whether a decomposable or a non-decomposable evaluation measure should be optimized. The ability to model conditional dependencies is crucial when optimizing non-decomposable measures, such as the subset 0/1 loss. Consequently, a single MLC method can usually not be expected to provide optimal predictions with respect to different types of evaluation measures. For example, depending on the dataset, minimization of the Hamming loss can result in poor performance according to the subset 0/1 loss or vice versa. Ideally, a learning algorithm should offer the ability to be tailored to a specific metric, depending on the use case at hand. Unfortunately, it often remains unclear what kind of measure different approaches aim to optimize. Instead of relying on a methodology that is suited for a specific use case, the ability to capture correlations in the label space is often considered a universal goal of MLC methods, regardless of the target measure.

3.6 Transformation Methods

Among the most popular approaches to multi-label classification are so-called *problem transformation methods*. They are based on transforming the original multi-label learning task into several sub-problems that can be solved by means of binary or multi-class classification algorithms. In the following, we introduce some of the most well-known transformation methods. We also discuss their capabilities and shortcomings when it comes to optimizing different evaluation measures, their ability to model marginal or conditional label dependence, as well as their computational efficiency.

Binary Relevance

Label-wise Decomposition

The *binary relevance* (BR) method is based on decomposing a multi-label problem into a series of binary classification tasks. Each one is concerned with the prediction of a single label. Given a dataset with K labels, it requires training a series of independent base classifiers $f_1, \dots, f_K$ on binary training sets

$$\widetilde{\mathcal{D}}_k = \{(\boldsymbol{x}_n, y_{nk}) \mid 1 \leq n \leq N\}, \tag{3.27}$$

which are derived from the original multi-label data. Given an example to predict for, each of the classifiers $f_k : \mathcal{X} \rightarrow \{0, 1\}$ provides a prediction for the corresponding label λ_k. Combining the predictions for the individual labels results in a binary label vector

$$\hat{\boldsymbol{y}} = (f_1(\boldsymbol{x}), \dots, f_K(\boldsymbol{x})). \tag{3.28}$$

If the learning algorithm that is used to tackle the individual sub-problems can predict probabilities, the BR method can also be used to provide a ranking of the labels by their probability to be relevant to an example.

Advantages and Disadvantages

As the information, which is provided to solve individual sub-problems, is restricted to a single label, the BR method cannot model any label dependence. Nevertheless, it is a natural choice when one is interested in optimizing decomposable evaluation measures, as they allow for a label-wise decomposition of the optimization problem, according to (3.26). Due to its conceptional simplicity and because BR can be instantiated with a wide variety of binary learning algorithms, it is probably the most commonly used approach to multi-label classification. Moreover, because each label is considered independently rather than taking combinations of labels into account, this particular transformation method comes with small computational overhead. The computational costs of training a BR model and providing predictions for unseen examples scale linearly with the number of labels.

Stacked Binary Relevance

The binary relevance method has received great attention in the literature, including several proposals that aim to overcome the limitations mentioned before. However, in the remainder of this work, we restrict ourselves to the basic concept introduced above, as it provides a straightforward solution for the label-wise optimization of decomposable evaluation measures. M.-L. Zhang, Y.-K. Li, et al. (2018) discuss several extensions to the BR approach. Most of them give up the strict separation of base classifiers to facilitate the exploitation of label correlations. They are usually referred to as *dependent* or *stacked binary relevance* (Montañes et al., 2014; Tsoumakas, Dimou, et al., 2009).

Label Powerset

Conversion into a Multi-class Problem

The *label powerset* (LP) transformation method, which originates from Boutell et al. (2004), transforms a multi-label classification problem into a multi-class classification task, where each label vector that is present in the training data is treated as a separate class. Given an injective mapping function $\sigma : \mathcal{Y} \rightarrow \mathbb{N}$ that assigns a unique number to each distinct label vector $\boldsymbol{y} \in Y$, the LP method transforms a multi-label dataset into a multi-class dataset

$$\widetilde{\mathcal{D}} = \{(\boldsymbol{x}_n, \sigma(\boldsymbol{y}_n)) \mid 1 \leq n \leq N\}, \tag{3.29}$$

which is used to train a multi-class classifier $f : \mathcal{X} \rightarrow \mathbb{N}$. The inverse of the mapping function σ^{-1} is used to obtain the label vector that should be predicted for a given example. As a result, the predictions of a LP model are restricted to label vectors that can be found in the training dataset. Depending on the class that is predicted by the base classifier, the multi-label prediction in form of a binary label vector computes as

$$\hat{\boldsymbol{y}} = \sigma^{-1}(f(\boldsymbol{x})). \tag{3.30}$$

Multi-class Binarization Techniques

Unlike the BR approach that can be implemented using binary classifiers, the LP method requires the base classifier to differentiate between an arbitrary number of classes. Therefore, an implementation of this particular problem transformation method using binary classification algorithms requires further decomposition of the meta-problem via binarization techniques, such as "one-against-all" training (see, e.g., Mehra and Gupta, 2013, for a survey on multi-class classification). If the multi-class classification method, which is used by an instantiation of the LP approach, can output a probability distribution over all classes, this problem transformation method can also be used for label ranking. In such a case, the rank of a label results from summing up the probabilities of the label vectors that indicate the relevance of the respective label.

Advantages and Disadvantages

As the label powerset transformation method treats different combinations of labels as mutually exclusive classes, it takes label dependence into account. Dembczyński, Waegeman, et al. (2012) provide theoretical and empirical arguments that the LP approach estimates the mode of the joint conditional distribution and therefore is a natural choice when one is interested in minimizing the subset 0/1 loss. However, despite the interesting theoretical properties of this problem transformation method, its practical relevance is limited due to the computational overhead it entails. As the number of possible label combinations 2^K grows exponentially with the number of label K, datasets with many labels and great label cardinality are challenging to solve with the LP approach. In such a scenario individual examples are likely associated with an unique label vector. Consequently, the multi-class classification task, which results from considering each label vector as a distinct class, requires dealing with many classes. Because of this increase in problem complexity, the base classifier is more likely to assign the wrong class to an example.

Pruned Problem Transformation

An approach that is closely related to the label powerset method is referred to as *pruned problem transformation* (PPT) (Read, 2008). It aims at reducing the complexity of the underlying multi-class problem by omitting classes that correspond to infrequent label vectors and replacing them with classes that refer to more frequent subsets of the affected labelsets. Such as the LP method, PPT does not allow for a prediction of label vectors that are missing from the training data. However, training several PPT classifiers on random subsets of the training data may circumvent this limitation. With such an approach, individual labels are only associated with an example if they are considered relevant by a majority of the meta classifiers (Gibaja and Ventura, 2014).

Random Labelsets

Dealing with Label Subsets

Compared to binary relevance, where individual labels are dealt with in isolation, and the label powerset method, where all labels associated with an example are considered jointly, *random k-labelsets* (RAkEL) strives for a compromise between these extremes. According to the authors, Tsoumakas, Katakis, and Vlahavas (2010), the method is motivated by the need to overcome the computational limitations and practical issues of the LP approach in applications with a large number of examples and labels without giving up on the ability to take label correlations into account. Similar to pruned problem transformation, RAkEL is based on breaking up the sets of labels that are associated with individual examples into smaller subsets. As the name of the method suggests, the subsets are chosen randomly, such that their size is equal to a hyper-parameter k. Training a RAkEL model requires building a collection of LP classifiers $F = \{f_1, \dots, f_T\}$ that are concerned with different, either overlapping or disjoint, subsets of the available labels $\mathcal{L}_t \subset \mathcal{L}$ with $|\mathcal{L}_t| = k$. RAkEL employs a label-wise voting scheme, as described in the previous section, to aggregate the predictions of the individual base classifiers. For each label, the voting is restricted to classifiers that take the respective label into account. Only if a label is included in the majority of the labelsets provided by these meta classifiers, it is considered relevant by the overall RAkEL classifier.

Advantages and Disadvantages

In applications to large-scale data, where the training time and predictive accuracy of the label powerset method suffer from the exponential number of label combinations, RAkEL may exhibit superior results in terms of training efficiency and predictive performance (Tsoumakas, Katakis, and Vlahavas, 2010). Although both approaches are explicitly aimed at modeling label correlations, the latter is restricted to exploiting dependencies between up to k labels. Hence, the extent to which RAkEL takes label dependence into account heavily depends on the hyper-parameter setting. For this reason, and due to the label-wise combination of the base classifiers, it remains unclear what evaluation measure RAkEL seeks to optimize (Dembczyński, Waegeman, et al., 2012).

Extensions to RAkEL

Similar to the problem transformation methods discussed earlier, several extensions to RAkEL have been proposed in the literature. In particular, this includes approaches where the label subsets are not chosen randomly but are selected deliberately. Selection methods of this kind are employed by RAkEL++ (Rokach, Schclar, and Itach, 2014), which also comes with a revised voting mechanism for aggregating the predictions of individual base classifiers.

Classifier Chains

Sequences of Binary Classifiers

Similar to the binary relevance method, a *classifier chain* (CC) (Read, Pfahringer, Holmes, and Frank, 2011) employs several binary classification models, each responsible for predicting a single label. Unlike the former approach, which deals with the labels in isolation and therefore neglects any dependencies between them, the exploitation of conditional label dependence is an explicit goal of the latter. It is achieved by training the binary classification models $f_1, \dots, f_K$ that are part of a classifier chain one after the other and providing information about the labels that are handled by preceding models to subsequent ones. Given a bijective permutation function $\tau : \mathbb{N}^K \rightarrow \mathbb{N}^K$ that imposes an order on the labels, the k-th model in the chain is concerned with the $\tau(k)$-th label. It is trained on an augmented dataset

$$\widetilde{\mathcal{D}}_k = \{(\widetilde{\boldsymbol{x}}_n, \boldsymbol{y}_n) \mid 1 \leq n \leq N\}, \tag{3.31}$$

which incorporates information about the labels for which a model has already been trained. Each example in such a training dataset corresponds to the form

$$\widetilde{\boldsymbol{x}} = \left(x_1, \dots, x_L, y_{\tau(1)}, \dots, y_{\tau(k-1)}\right), \tag{3.32}$$

i.e., it is not only represented by its original features but is augmented with the ground truth labels that are modeled by preceding models. As a result, classifiers that are later in the chain may take the relevance or irrelevance of previously handled labels into account to make decisions for the label they are concerned with. When applying a classifier chain to an unseen example, the binary classifiers in the chain are queried in the order they have been trained to obtain predictions for the corresponding labels. The predictions of preceding classifiers are passed to their successors to comply with the form of the data used for training.

Advantages and Disadvantages

Classifier chains typically exhibit superior performance compared to the binary relevance method when it comes to optimizing non-decomposable evaluation measures that benefit from the exploitation of conditional label dependence. Unlike the LP method or RAkEL, this does not come with a significant computational overhead. Instead, the complexity of training and prediction depends linearly on the number of labels, even though the base classifiers cannot be trained in parallel as with a BR decomposition. A drawback of the CC approach is its susceptibility to the label permutation used for training and prediction. Given different orders of the labels, the performance of the resulting classifier chains most probably varies, as their base classifiers are trained on different datasets. If an incorrect prediction is made by a classifier early in the chain, the error is propagated through the chain and may negatively impact the decision of subsequent classifiers.

Probabilistic Classifier Chains

Whereas the CC method, as proposed by Read, Pfahringer, Holmes, and Frank (2011), relies on binary base classifiers that predict whether individual labels are relevant or irrelevant, Dembczyński, Cheng, and Hüllermeier (2010) generalize the underlying concept by using base classifiers that are capable of estimating the probability of individual labels being relevant. This extension, which is referred to as *probabilistic classifier chains*, aims to approximate the joint distribution of the labels in a step-wise manner, as suggested by the product rule of probability in (3.24). Due to its theoretical foundation, the use of probabilistic classifier chains to minimize the subset $0/1$ loss is well-justified (Dembczyński, Waegeman, et al., 2012). On the downside, this approach comes with exponential costs at prediction time, as a search for the label vector that maximizes the joint probability requires to consider all possible combinations of the available labels.

Ensembles of Classifier Chains

To reduce the negative effect, a suboptimal choice of the label permutation may have on the performance of a classifier chain, Read, Pfahringer, Holmes, and Frank (2011) propose to use *ensembles of classifier chains*. This approach requires training several classifier chains using different permutations of the labels. Finally, the predictions of the individual models are averaged to predict for an unseen example. However, similar to RAkEL, which also employs such a label-wise averaging scheme, it is unclear what evaluation measure ensembles of classifier chains tend to optimize (Dembczyński, Waegeman, et al., 2012).

Dynamic Classifier Chains

Another attempt to overcome the main drawback of the CC approach, which is the susceptibility to the label permutation, is known in the literature as *dynamic classifier chains*. Rather than relying on a fixed order of the labels determined before training even starts, with this approach, the most promising label to focus on is selected during the training phase whenever a new base classifier should be added to the model. The strategy for selecting the next label typically depends on the type of classifiers in the chain. For example, methods to dynamically train CC models have been developed in the context of tree-based models (Loza Mencía, Kulessa, et al., 2022), including random decision trees (Kulessa and Loza Mencía, 2018) and gradient boosted decision trees (Bohlender, Loza Mencía, and Kulessa, 2020), or Naive Bayes classifiers (Trajdos and Kurzynski, 2019).

3.7 Adaptation Methods

Instead of relying on problem transformation methods to decompose multi-label problems into binary or multi-class classification tasks, as previously discussed in Section 3.6, MLC can also be approached by developing algorithms specifically tailored to this particular problem domain. We refer to this kind of approach as a *problem adaptation method* and provide an overview of existing adaptation methods, which are closely related to the approaches presented in this work, in the following. Per the topic of this thesis, we focus on symbolic learning methods that result in decision trees or rule-based models. However, different types of machine learning techniques, including unsupervised approaches or statistical methods, such as support vector machines or neural networks, have been considered for solving MLC problems as well. A more comprehensive overview is provided by survey articles on the topic, for example by Gibaja and Ventura (2014) or M.-L. Zhang and Z.-H. Zhou (2014).

Multi-Output Decision Trees

Generalization of Split Criteria

Decision trees (cf. Section 2.1) are arguably one of the oldest supervised learning approaches, and they are commonly used to solve classification problems. Consequently, means to apply tree-based methods to multi-label classification tasks have been investigated very early. Compared to traditional decision trees, as used for binary or multi-class classification, the leaf nodes of *multi-output decision trees* may incorporate information about several labels instead of being restricted to one of the available classes. Their construction is often based on generalizations of existing split criteria, such as the information gain, that allow assessing the quality of possible splits with respect to multiple labels (Clare and King, 2001). This also applies to the use of random decision trees, which have been investigated in the context of MLC because they are faster to train than conventional decision trees (X. Zhang et al., 2010).

Predictive Clustering Trees

Compared to traditional tree learning algorithms, *predictive clustering trees* (Blockeel, Raedt, and Ramon, 1998) (PCT) follow a different approach for tree construction that is well-suited for multi-dimensional classification or regression problems. They employ a clustering algorithm to partition the data at each node such that the intra-cluster variation is minimized (Gjorgjioski, Kocev, and Džeroski, 2011). Each leaf node of the resulting tree consists of prototypical training examples with similar labels. Given an unknown example to predict for, its labeling is derived from the prototypical examples belonging to the same leaf.

Ensembles of Multivariate Trees

In general, multi-output decision trees, including predictive clustering trees, are often used in ensemble methods, e.g., in random forests (cf. Section 2.3), and have been shown to achieve strong predictive results (Kocev et al., 2007; Madjarov et al., 2012; Qingyao Wu et al., 2016).

Multivariate Boosting

Variants of AdaBoost

Adaptation methods that rely on boosting (cf. Section 2.3) to learn an ensemble of weak learners are particularly relevant to this work. In Chapter 6, we propose to use the boosting framework for learning rule-based multi-label classification models. Existing MLC approaches that are based on this particular learning technique include generalizations of the AdaBoost algorithm. Whereas *AdaBoost.MH* (Schapire and Singer, 2000) and its variants (e.g., Diao et al., 2002; Esuli, Fagni, and Sebastiani, 2006) aim to minimize the Hamming loss, *AdaBoost.MR* (Schapire and Singer, 2000) is suited for the label-wise minimization of ranking losses. Most boosting algorithms intended for use in MLC, including those that focus on ranking losses, are restricted to decomposable loss functions. A noteworthy exception is *AdaBoost.LC*, which enables minimizing a family of non-decomposable losses that includes the Subset 0/1 loss as a special case.

Boosted Multi-Output Decision Trees

Whereas AdaBoost, and its successors that are deliberately designed for MLC, traditionally use decision stumps that predict for a single label, more recent boosting approaches to MLC tend to employ multi-output decision trees as their ensemble members. Besides approaches aimed at the Hamming loss (e.g, Johnson and Cipolla, 2005; Si et al., 2017; Yan, Tešić, and Smith, 2007; Z. Zhang and C. Jung, 2020), this also includes algorithms that are tailored to ranking losses (e.g., Dembczyński, Kotłowski, and Hüllermeier, 2012; Y. H. Jung and Tewari, 2018).

Rule-based Methods

Association Rules

Rule-based learning methods are conceptually close to decision trees. They are often preferred in applications where interpretable models, which can be inspected and understood by humans, are desired. As previously discussed in Section 2.2, rule learning approaches that rely on techniques for association rule mining are primarily used to learn descriptive models. Several approaches that apply the underlying concepts to multi-label classification problems can be found in the literature (B. Li et al., 2008; Thabtah and Cowling, 2007; Thabtah, Cowling, and Peng, 2004, 2006). When dealing with multi-label data, the ability to identify frequently re-occurring items is particularly well-suited to discover patterns among the ground truth labels of training examples. Similar to multi-output decision trees, this allows learning rules that comprise information about several labels rather than being restricted to a single class as in traditional classification settings.

Separate-and-Conquer Approaches

In contrast to association rule mining, methods based on the separate-and-conquer paradigm aim to learn predictive models that can provide accurate predictions for unseen data. The principles of separate-and-conquer learning have first been generalized to the multi-label setting by Loza Mencía and Janssen (2016). In their work, the authors investigate two different approaches that aim to capture label dependencies in the form of rules. On the one hand, they rely on a problem transformation method similar to stacked binary relevance (cf. Section 3.6). This allows using traditional single-label rule learners based on the SeCo principle for tackling multi-label classification problems. Even though with the stacked binary relevance approach, a rule model is built for each label separately, it provides the individual base learners with information about the predictions made for other labels and therefore allows to model conditional label dependencies. On the other hand, the authors investigate a multi-label SeCo approach, where rules that are learned at a particular training iteration may test for the presence or absence of labels according to the predictions of previous rules. Similar to the use of stacked binary relevance, rules that are learned by this method can model local label correlations. The multi-label SeCo approach was later extended by Rapp, Loza Mencía, and Fürnkranz (2018). Their work is devoted to the challenge of inducing multi-label rules, which predict for several labels simultaneously and therefore can capture co-occurrences or mutual exclusions between multiple labels, in a computationally efficient manner. In Chapter 5, we propose several extensions to existing SeCo-based multi-label classification approaches.

3.8 Dimensionality Reduction

In the digital world, where globally operating web platforms reach billions daily, the amount of available multi-label data increases steadily and opens the door for novel use cases. However, applying MLC methods to large-scale problems, such as product recommendation (Rahul Agrawal et al., 2013), social network analysis (Denton et al., 2015), or the categorization of online documents (Prabhu and Varma, 2014) demands highly scalable learning methods that can be trained on large amounts of high-dimensional data. The problem transformation methods in Section 3.6, as well as the adaptation methods in Section 3.7, are usually unable to meet the strict time and memory constraints of such real-world applications. Instead, to compensate for the high computational demands that come with vast numbers of training examples, attributes, or labels, methods for dimensionality reduction are indispensable in practice.

Sampling Techniques

Random Sampling

Many classification methods, in particular ensemble methods (cf. Section 2.3), use sampling techniques to obtain random subsets of the available training examples. Due to the ability to produce more diverse ensembles and increase the variance among ensemble members, which may improve predictive accuracy, it is an essential aspect of random forests (Breiman, 2001) and stochastic gradient boosting (J. H. Friedman, 2002). In addition, as it reduces the number of examples to be dealt with by a learning algorithm, it may also significantly reduce training times. For this reason, it is often used in highly scalable classification systems. For example, LightGBM (Ke et al., 2017) incorporates *gradient-based one-side sampling* (GOSS) into the training procedure. This sampling technique prioritizes examples for which a model does not yet deliver accurate predictions while at the same time trying to maintain the class distribution to avoid deterioration of a model's predictive performance even when focusing on a small fraction of the available examples.

Stratified Sampling

In multi-label classification, where each example is associated with several labels with varying frequency, the development of stratified sampling methods, which result in samples that are representative of the overall data distribution, is a conceptually challenging problem. Sechidis, Tsoumakas, and Vlahavas (2011) propose two methods for stratified sampling from multi-label data. They aim to maintain the frequency of individual labels or entire labelsets, respectively. Szymański and Kajdanowicz (2017) extend their approach by the ability to take pairs of labels into account. Merrillees and L. Du (2021) present an alternative method that is specifically designed for extreme multi-label classification.

Under- and Oversampling

Besides the computational challenges that come with large multi-label datasets, the label imbalance (cf. Section 3.2) tends to be more pronounced in domains with large numbers of examples and labels and may hinder the effectiveness of MLC methods. Sampling methods have recently been shown to be an effective strategy to deal with this problem, e.g., by undersampling examples that are harmful to a learning method's predictive performance (Charte et al., 2013, 2019; B. Liu and Tsoumakas, 2020), by increasing the weight of examples that are difficult to classify (Charte et al., 2013, 2019), or even by introducing synthetic examples to compensate for underrepresented regions in the feature space (Charte et al., 2019; B. Liu, Blekas, and Tsoumakas, 2022).

Feature Processing

Feature Selection

The computational demands of classification methods are also affected by the number of features that must be considered during training. To improve the scalability of existing learning approaches, methods that reduce the dimensionality of the feature space have become an established topic of research on classification methods. For example, Liang Sun, Ji, and Ye (2019) elaborate on methods specifically tailored to the particularities of MLC problems. In particular, this includes methods for *feature selection*. They aim to identify a subset of the available attributes that is well-suited to solve a given classification task by omitting irrelevant and redundant attributes or focusing on the ones that are most strongly correlated with the target variables. On the one hand, this includes unsupervised methods, e.g., methods like *principal component analysis* (Wold, Esbensen, and Geladi, 1987) or *latent semantic indexing* (Deerwester et al., 1990), that are commonly used in single-label classification and can be applied to multi-label data without any modifications. On the other hand, supervised methods explicitly take information about the labels into account. Pereira et al. (2018) provide an extensive overview of feature selection methods that are intended for use in multi-label classification and propose a taxonomy for categorizing existing techniques. In particular, they distinguish between selection methods, which are implemented as a preprocessing step and therefore are independent of any particular MLC algorithm, and selection strategies, which are embedded into a specific learning algorithm. Furthermore, the available methods differ in how they take combinations of labels into account. On the one hand, methods that apply a supervised method for feature selection to each label independently are reasonable if one is interested in optimizing label-wise decomposable evaluation measures (e.g., Tsoumakas, Katakis, and Vlahavas, 2008; Yiming Yang and Pedersen, 1997). On the other hand, more recent approaches usually take label dependencies into account for selecting a subset of the available attributes. For example, they are based on problem transformation methods, such as PPT or LP (cf. Section 3.6). Spolaôr et al. (2013) provides a comparison of feature selection methods of this kind (e.g., Doquire and Verleysen, 2011; Reyes, Morell, and Ventura, 2015; Trohidis et al., 2008). Other approaches can directly deal with multiple labels without the need to transform the input data (e.g., Jian et al., 2016; Y. Lin, Q. Hu, J. Liu, J. Chen, et al., 2016; Y. Lin, Q. Hu, J. Liu, and Duan, 2015; Shao et al., 2013; M.-L. Zhang, Peña, and Robles, 2009).

Feature Extraction

Rather than selecting a subset of the attributes present in a dataset, methods for *feature extraction* obtain new attributes by combining and transforming existing ones. Again, this includes unsupervised methods that are known from traditional classification settings and ignore any information about ground truth labels present in the training data. Prominent examples are the application of *self-organizing feature maps* or *latent semantic indexing* in text classification (Luo and Zincir-Heywood, 2005), as well as generalizations of the *linear discriminant analysis* to the multi-label setting (C. H. Park and Lee, 2008; Wang, Ding, and H. Huang, 2010). In contrast, the mapping to a feature space with reduced dimensions, which results from the application of supervised feature extraction methods (e.g., Xu, 2018; Xu et al., 2016; K. Yu, S. Yu, and Tresp, 2005; M.-L. Zhang and L. Wu, 2014; Yin Zhang and Z.-H. Zhou, 2010), does not only depend on the feature values of the available training examples but also results from taking their respective ground truth labels into account.

Label Space Transformation

Hierarchy of Multi-label Classifiers

Unlike in single-label classification, where each example always corresponds to a single class, the computational complexity of multi-label classification algorithms is directly affected by the number of labels included in a dataset. In connection with the increasing relevance of extreme multi-label classification, where large numbers of labels should be dealt with, methods that reduce the dimensionality of the label space have attracted much attention in the literature. An early approach to *label space transformation* is the HOMER algorithm proposed by Tsoumakas, Katakis, and Vlahavas (2008). It transforms a multi-label problem into a tree-shaped hierarchy of simpler MLC tasks by employing a clustering algorithm to identify groups of similar labels. Each leaf node in the hierarchy corresponds to a single label. In contrast, internal nodes are concerned with the union of the labelsets that correspond to their children. Each node employs a multi-label classifier that is either implemented as an adaptation method or relies on a problem transformation approach. It is trained on the examples that are associated with at least one of the node's labels. The prediction of unseen examples starts at the root node. It follows a recursive procedure where an example is passed to a child node if it is concerned with at least one of the labels that are predicted as relevant by their parent node. The final prediction is the union of the labels corresponding to child nodes to which an example has been forwarded.

Label Reduction with Association Rules

Charte et al. (2012) follow a different approach where an algorithm for association rule discovery is applied to the ground truth labels of the training examples. The resulting association rules capture unconditional label co-occurrences that have been found in the data, i.e., cases where the presence of a specific label implies the presence of another label. This representation can be used to carry out a pre-processing phase where inferred labels are removed from the data. Finally, after an MLC method has been trained on the dataset with reduced dimensions, its predictions for unseen examples are amended by adding labels implicitly rendered as relevant by the previously obtained association rules.

Label Embeddings

The label space transformation approaches that are most commonly used for extreme multi-label classification are typically based on establishing a mapping from the label vectors present in a dataset to a label space with reduced dimensions. This enables to limit the amount of memory and the computational efforts necessary to process datasets that come with a large number of labels. To obtain predictions from a multi-label classifier that has been trained on a low-dimensional subspace, the label vector it provides for an unseen example is ultimately projected back to the original label space. Existing embedding methods mainly differ in the choice of their compression and decompression techniques. An early instantiation of embedding methods is *compressed sensing* (D. Hsu et al., 2009). It aims at mapping the label space to a low-dimensional subspace via a random projection based on the label sparsity. Its main drawback is the computationally costly prediction phase. *Compressed labeling* (T. Zhou, Tao, and X. Wu, 2012) and *principal label space transformation* (Tai and H.-T. Lin, 2012) were proposed as alternatives that allow for a more efficient decompression of label matrices. Whereas the former employs a clustering approach to reconstruct labelsets from the original label space, the latter relies on singular value decomposition. Other approaches are based on *bloom filters* (Cissé et al., 2013) or *landmark labels* (Balasubramanian and Lebanon, 2012; Bi and Kwok, 2013). In contrast to the previously mentioned approaches, the idea of taking information about the feature space into account for establishing a mapping to a low-dimensional subspace, rather than focusing exclusively on the label vectors that are present in a dataset, has become a driving force for the development of novel embedding methods (e.g., Bhatia et al., 2015; Y.-N. Chen and H.-T. Lin, 2012; Kimura, Kudo, and Lu Sun, 2016; Kumar et al., 2019; Z. Lin et al., 2014; Liang Sun, Ji, and Ye, 2010; Yi Zhang and Schneider, 2011; Zhu et al., 2018). Other innovations in the field are motivated by the desire to tailor embedding methods to different evaluation measures (e.g., K.-H. Huang and H.-T. Lin, 2017) or support applications with missing labels (e.g., Y. Li and Youlong Yang, 2020).

A Modular Framework for Learning Multi-label Rules

4

In this chapter, we provide a unified view of different rule-based approaches to multi-label classification that have been proposed in the past. In recent years, an increasing number of publications have been devoted to using rule learning methods for tackling this particular problem domain. Based on the outcome of these studies, we argue that rules enable to model multi-label data naturally and are a versatile tool for building classification models that are specifically tailored to different use cases and requirements. Despite the varying characteristics of the rule-based models that have been used for MLC, the methodologies and algorithms used for inducing rules tend to share many of the underlying ideas and techniques. Based on this observation, we identify the algorithmic aspects that are essential to all rule-based problem adaption methods and integrate them into a consistent and modular framework. Not only can existing multi-label rule learning methods be viewed as instantiations of this framework, but it may also serve as a basis for designing new rule learning algorithms that are specifically tailored to the particularities of multi-label classification. In Chapter 5 and Chapter 6, we present instantiations of the presented framework to demonstrate its practical relevance. Furthermore, we rely on the framework's flexibility in Chapter 7 and Chapter 8, where we exploit its modularity to investigate several extensions to the previously proposed algorithms.

4.1 Multi-label Rules

Rule Bodies and Heads

Following the notion of rules traditionally used in single-label classification, i.e., for tackling binary or multi-class classification problems (cf. Section 2.2), we aim to model multi-label data by learning predictive rules of the form

$$f : Head \leftarrow Body.$$

The body of a rule consists of one or several conditions that specify the examples to which the rule applies, and the head provides a prediction for these covered examples. In single-label classification, conditions always refer to one of the attributes in a dataset. Such a condition c_l compares an example's value for the l-th attribute to a constant using a relational operator, such as $=$ and $\neq$, if the attribute A_l is nominal, or $\leq$ and $>$, if the attribute is numerical or ordinal. When dealing with multi-label classification tasks, other types of bodies are also conceivable. In the MLC setting, a variety of options also exist when it comes to the heads of rules. Whereas single-label classification rules must assign a prediction to the only available class, the heads of multi-label rules may comprise information for more than a single label. In the following, we recapitulate the terminology proposed by Loza Mencía, Fürnkranz, et al. (2018) to distinguish between various forms of multi-label rules. In addition, we outline some of the most relevant methods that have been investigated

Loza Mencía, Fürnkranz, Hüllermeier, and Rapp (2018): 'Learning Interpretable Rules for Multi-Label Classification' in the literature to learn such rules and discuss to which extent they are suited to express the different types of label dependencies that may be hidden in multi-label data.

Deterministic vs. Probabilistic Predictions

First of all, we distinguish between *deterministic* and *probabilistic* predictions. On the one hand, the former is often preferred when simple and interpretable models are needed. In such a case, the heads of rules assign binary values $\hat{p}_k \in \{0,1\}$ to the k-th label to indicate whether it is relevant ($\hat{p}_k = 1$) or irrelevant ($\hat{p}_k = 0$) to the covered examples.[1] Besides rule learning algorithms that rely on the separate-and-conquer paradigm (Klein, Rapp, and Loza Mencía, 2019; Loza Mencía and Janssen, 2016; Rapp, Loza Mencía, and Fürnkranz, 2018, 2019), binary rules are also used by methods based on association rule discovery (Lakkaraju, Bach, and Leskovec, 2016; B. Li et al., 2008; Thabtah and Cowling, 2007; Thabtah, Cowling, and Peng, 2004, 2006), genetic algorithms (Allamanis, Tzima, and Mitkas, 2013; Cano et al., 2013), as well as evolutionary classification systems (Arunadevi and Rajamani, 2011; Ávila-Jiménez, Gibaja, and Ventura, 2010). Moreover, in Chapter 5, we elaborate on a SeCo-based MLC method that makes use of deterministic rules. On the other hand, rules with probabilistic predictions provide information about the distribution of labels in a dataset, most commonly given in the form of real-valued scores $\hat{p}_k \in \mathbb{R}$. Such scores express a preference for predicting an individual label as relevant or irrelevant. Similar to the rules induced by boosting-based learning approaches for single-label classification, e.g., by ENDER (Dembczyński, Kotłowski, and Słowiński, 2010), this representation is natural when using statistical optimization techniques for solving multi-label problems. Hence, probabilistic rules form the basis of the models that result from the boosting-based MLC method presented in Chapter 6.

1: For brevity, we use the shorthand notation $\hat{p}_k$ and $\neg\hat{p}_k$ to denote binary predictions that indicate the relevance and irrelevance of a label, respectively.

Label-dependent Rules

One possibility to express correlations in the label space by means of rules is to not only consider the attributes in a dataset but also the labels for constructing the conditions that may be included in a rule's body. This is similar to classifier chains (cf. Section 3.6), where the predictions of classifiers that are earlier in the chain may be taken into account by subsequent classifiers. However, when transferring this idea to rules, information about labels, for which predictions are already available, is propagated at the level of individual rules rather than entire models. S.-H. Park and Fürnkranz (2008) use binary rules of the form $\hat{p}_1 \leftarrow \hat{p}_2$ to capture co-occurrences of labels. Such rules predict a label as relevant if another label is considered relevant as well. Similarly, they model disjoint labels that do not co-occur in the data by using rules of the form $\neg\hat{p}_1 \leftarrow \hat{p}_2$. Following the naming conventions in (Loza Mencía, Fürnkranz, et al., 2018), we characterize such rules, where all conditions in the body are concerned with labels, as *fully label-dependent*. Whereas this particular kind of rules is suited for capturing marginal label dependencies, which are not restricted to a specific region in the feature space, *partially label-dependent* rules enable to model conditional label dependence. Loza Mencía and Janssen (2016) utilize a stacked binary relevance approach to induce partially label-dependent rules, such as $\hat{p}_1 \leftarrow \hat{p}_2 \land c_1$ or $\neg\hat{p}_1 \leftarrow \hat{p}_2 \land c_1$, using RIPPER (Cohen, 1995) as a base learner. Such rules predict a label as relevant or irrelevant, depending on whether one or several other labels have already been associated with the examples in a particular region of the feature space. In addition, the authors demonstrate that the separate-and-conquer paradigm, which many traditional rule learners are based on, can be generalized to the multi-label setting. Compared to using a binary black-box classifier, such a problem adaptation approach provides more fine-grained control over the label dependencies that individual rules should take into account. Moreover, the rules are not susceptible to a predefined order of the labels, as is the case with a CC approach. This is because the labels a rule should predict for are selected implicitly during training, depending on the objective that guides the search for the most desirable rules. In contrast to problem transformation methods that employ single-label base classifiers, problem adaptation approaches are not restricted to the induction of label-dependent rules. Instead, as discussed below, they allow for the construction of rules that predict for multiple labels simultaneously.

Multi-label Heads

The different types of rules discussed so far focus on the prediction for a single label. We refer to the heads of such rules as *single-label heads*. When using label-dependent rules for modeling label correlations, as discussed earlier, combinations of several rules are necessary to represent interactions between labels. As an alternative, the construction of *multi-label heads*, which predict for several labels simultaneously, enables to model conditional dependencies more compactly. In particular, multi-label heads are a natural choice for the representation of co-occurrences that hold for certain regions of the feature space. This pattern, frequently found in multi-label data, can easily be modeled by rules such as $\hat{p}_1, \hat{p}_2 \leftarrow c_1$. Accordingly, local exclusions of labels can also be expressed through multi-label heads, e.g., by learning rules of the form $\hat{p}_1, \neg\hat{p}_2 \leftarrow c_1$. The induction of multi-label heads is often implemented as a post-processing step. For example, they can be constructed from association rules by merging the single-label heads of rules that cover overlapping regions of the feature space (B. Li et al., 2008; Thabtah and Cowling, 2007; Thabtah, Cowling, and Peng, 2006). The use of problem adaption methods facilitates the induction of rules with multiple predictions in the head. However, naive implementations of such an approach can be computationally expensive, as the number of label combinations that must be considered for each rule grows exponentially with the number of available labels (cf. Section 4.3). Existing SeCo algorithms for the induction of binary rules prune the search for multi-label heads to overcome this limitation (Klein, Rapp, and Loza Mencía, 2019; Rapp, Loza Mencía, and Fürnkranz, 2018). This enables to omit unpromising solutions rather than taking all possible combinations of

	Body	Example
Single-label head		
	Label-independent	$\hat{p}_1 \leftarrow c_1$
	Fully label-dependent	$\hat{p}_1 \leftarrow \hat{p}_2$
	Partially label-dependent	$\hat{p}_1 \leftarrow \hat{p}_2 \wedge c_1$
Partial multi-label head		
	Label-independent	$\hat{p}_1, \hat{p}_2 \leftarrow c_1$
	Fully label-dependent	$\hat{p}_1, \hat{p}_2 \leftarrow \hat{p}_3$
	Partially label-dependent	$\hat{p}_1, \hat{p}_2 \leftarrow \hat{p}_3 \wedge c_1$
Complete multi-label head		
	Label-independent	$\hat{p}_1, \neg\hat{p}_2, \neg\hat{p}_3, \dots, \hat{p}_K \leftarrow c_1$

Table 4.1: Examples of different types of binary multi-label rules that provide predictions for $K > 3$ labels and are characterized by the form of their body and head. Instead of solely focusing on the relevance of labels, the conditions of label-dependent rules may also test for their irrelevance. Accordingly, rules with multi-label heads may indicate both the relevance and irrelevance of labels.

labels into account. In the following, we characterize multi-label heads that are concerned with a subset of the available labels as *partial*. In contrast, *complete* heads provide a prediction for each of the available labels.

Exemplary Rules

Table 4.1 provides an overview of the terminology used to distinguish between different types of multi-label rules, alongside examples that illustrate their respective characteristics. For simplicity, the given table focuses on binary rules. Nevertheless, the different representations of a rule's body and head are also conceivable when using probabilistic rules. In such a case, the head of a rule predicts one or several real-valued scores. Accordingly, the body of a label-dependent rule may contain numerical conditions that test for the probabilistic scores provided by other rules.

4.2 Assemblage of Rule Models

Rule Models

We aim at learning predictive models that consist of several (multi-label) rules. Without loss of generality, we assume that the rules are given in a particular order. This enables us to account for methods where the prediction for unseen examples depends on the order of the rules, as is the case with SeCo algorithms (cf. Section 4.4). We denote a rule-based model, consisting of T rules, as a sequence

$$F = (f_1, \ldots, f_T) \,. \tag{4.1}$$

All rules in a model are usually of the same form. For example, an algorithm that aims at the induction of probabilistic single-label rules does typically not produce rules that provide deterministic predictions or are concerned with more than a single label.

Default Rules

Unlike decision trees, which cover the entire feature space (cf. Section 2.1), individual rules only provide predictions for a particular region of the feature space for which the conditions in their body are satisfied. Rule-based models typically include a *default rule* to account for cases where no other rule does cover an example. It does not contain any conditions in its body and covers all examples for which a model may predict.

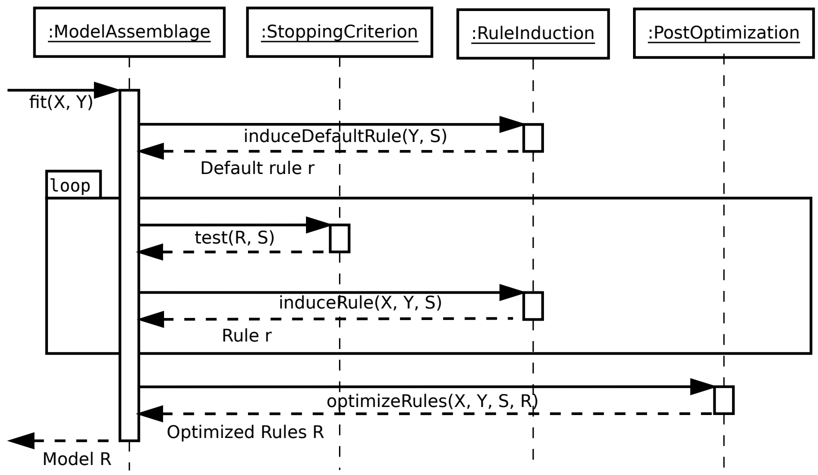


Figure 4.1: A UML sequence diagram that provides a high-level overview of the interactions between the components involved in the induction of a rule-based model. The feature values of the training examples X and their ground truth labels Y are provided as the input to the algorithm. Individual rules are learned with respect to statistics S that incorporate information about the ground truth labels of individual examples and the corresponding predictions of the current model. The loop that is responsible for learning new rules is exited as soon as a certain stopping criterion is met.

The predictions of a default rule may be deterministic or probabilistic, depending on the type of predictions provided by the other rules in the model. However, a default rule is always supposed to predict for all available labels, regardless of whether the other rules provide partial or complete predictions. For example, SeCo algorithms, such as those by (Rapp, Loza Mencía, and Fürnkranz, 2018) or (Klein, Rapp, and Loza Mencía, 2019), employ binary default rules that predict a label as irrelevant unless most training examples are associated with the respective label. In cases like these, where the order of the rules matters for prediction, the default rule comes last, i.e., the other rules take precedence.

Sequential Rule Induction

We induce the rules to be included in a model using an iterative algorithm, where new rules are learned with respect to their predecessors. As can be seen in Figure 4.1, the assemblage of a model starts with the induction of a default rule. It applies to all examples, regardless of the region in the feature space they belong to. The following rules should focus on subspaces that the preceding rules do not yet handle accurately. According to Figure 4.1, they are constructed by following a sequential procedure. For the induction of a single rule, the following aspects must be taken into account:

- **Stopping Criterion.** The number of rules included in a model is controlled by one or several stopping criteria. If at least one criterion is satisfied, the rule induction process is stopped rather than learning a new rule. A straightforward implementation of such a criterion is based on counting the number of rules induced so far. If the model's size has reached a predefined number of rules, the induction of rules is put to an end. Empirical studies have shown that the predictive performance may suffer from too many rules being included, which introduces the risk of fitting noise in the data Rapp, Loza Mencía, and Fürnkranz (2019) . This shows the demand for stopping criteria that are effective in preventing overfitting.[2] Among others, stopping criteria based on the *minimum description length* (MDL) principle have successfully been used in single-label rule learning, e.g., by RIPPER (Cohen, 1995). The generalization of MDL-based stopping criteria to the MLC setting remains for future work. As an alternative, *early stopping*, as it is often employed by boosting algorithms, such as XGBoost (T. Chen and Guestrin, 2016) or LightGBM (Ke et al., 2017), may be used. It is based on keeping track of a model's performance on an otherwise unused portion of the training data and terminates the training process as soon as the performance stagnates or declines.

Rapp, Loza Mencía, and Fürnkranz (2019): 'On the Trade-off Between Consistency and Coverage in Multi-label Rule Learning Heuristics'

2: The use of stopping criteria is also referred to as *pre-pruning* (Fürnkranz, 1997).

- **Rule Induction.** Learning a new rule requires determining a certain region in feature space to be covered by the rule and choosing the conditions in its body accordingly. There is typically a trade-off between the *coverage* and *consistency* of a rule. The former corresponds to the size of the covered region, whereas the latter refers to the fraction of examples within the covered subspace for which a rule's predictions are correct (Janssen and Fürnkranz, 2008; Rapp, Loza Mencía, and Fürnkranz, 2019). The predictions assigned by a rule's head should be chosen with regard to the covered examples. If partial heads are desired, it is necessary to decide on a fraction of the labels to predict for.

Post-Optimization

An optional post-processing step may be conducted after the induction of new rules has stopped. Initially, at each iteration of the rule induction process, information about the current model is taken into account to construct a new rule. However, only after the rule induction has been completed, the model's performance can be verified as a whole. Consequently, this last stage of training enables to carry out optimizations that aim at fine-tuning a model globally. This may include the removal or addition of entire rules, as well as modifications to existing ones. Prominent examples of such optimization strategies are *post-pruning* algorithms, such as *reduced error pruning* (REP), that aim at improving the quality of a model by removing superfluous conditions from individual rules (Fürnkranz, 1997). Another successful post-optimization strategy is employed by RIPPER (Cohen, 1995), where individual rules are relearned in the context of all preceding and following rules.

4.3 Induction of Single Rules

Finding Body and Head

Hüllermeier, Fürnkranz, Loza Mencía, Nguyen, and Rapp (2020): 'Rule-based Multi-label Classification: Challenges and Opportunities'

At the core of the rule learning framework presented in this chapter are the algorithmic components responsible for the induction of individual rules, including the default rule. As previously outlined by Hüllermeier, Fürnkranz, et al. (2020) , constructing a rule requires identifying the conditions that should be included in its body. We discuss this aspect in the section "Candidate Generation" below. The conditions in a rule's body specify the region in the feature space covered by the rule. The search for suitable conditions can be omitted when learning a default rule, as it does not contain any conditions in its body. In addition, the head of a rule must be found, as discussed in the section "Evaluation of Candidates". The predictions it provides for individual labels should be chosen with regard to the covered region, such that the rule predicts accurately for the covered training examples.

Label Space Statistics

As depicted in Figure 4.1, we rely on so-called *label space statistics* as the basis for learning rules. They incorporate information about the ground truth labels of individual training examples and the corresponding predictions of the rules that have already been induced until a particular iteration of the sequential rule induction process. The semantics of the statistics may vary depending on the type of algorithm used. For example, SeCo algorithms characterize the predictions for individual examples and labels in terms of confusion matrix elements, i.e., they differentiate between true positives, false positives, true negatives, and false negatives (cf. Table 3.3), based on the ground truth and the predictions a model provides. Furthermore, maintaining binary or real-valued weights enables tracking whether individual labels of the training examples have already been covered in previous iterations of a SeCo or weighted covering algorithm. In contrast, boosting methods rely on gradients that indicate whether a model provides accurate predictions regarding a particular loss function or if the predictions leave room for improvement. Depending on the target measure an algorithm addresses, the statistics for individual examples must not necessarily be associated with individual labels but may also correspond to pairs of labels or label subsets. As new rules alter the predictions for training examples they cover, the corresponding statistics must be updated when a new rule is added.

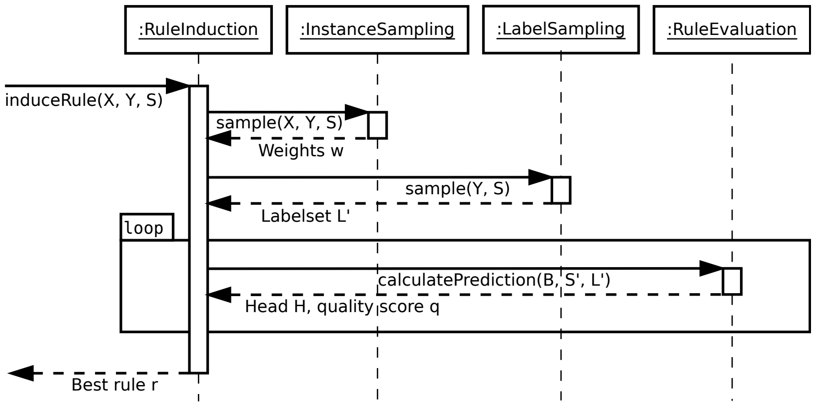


Figure 4.2: An illustration of the individual steps that are necessary for inducing a single rule. A rule can be learned on a subset of the training examples, which is specified by assigning weights $\boldsymbol{w}$ to individual examples. The predictions of a rule may be restricted to a subset of the labels $\mathcal{L}$. The search for the best rule f requires to enumerate the candidate bodies that may be used by potential rules. For each body, a corresponding head must be found. Its predictions for individual labels depend on statistics S' corresponding to the examples that are covered by the rule's body. The quality of potential rules is assessed in terms of numerical scores q.

Candidate Generation

Search for Candidate Rules

The induction of an individual rule requires making up candidates that can be added to an existing model. The number of candidates that are considered when searching for a new rule is usually large and substantially exceeds the size of the model that is eventually produced. In general, different strategies for the generation of potential rules are conceivable. The size of the search space, i.e., the number of rules that can be constructed, is typically immense. Rather than employing an exhaustive search, algorithms for rule induction are typically guided by heuristics or rely on theoretical guarantees to focus on the most promising candidates while pruning large regions of the search space. The quality of potential rules must be assessed and compared to each other to decide which one results in the greatest improvement when added to a model.

Instance Sampling

The number of possible candidates heavily depends on the dimensionality of the feature space. The bodies of candidate rules are usually constructed from the feature values of the available training examples, which are provided in a two-dimensional *feature matrix* $X \in \mathbb{R}^{N \times L}$. A subset of the training examples can optionally be selected via *instance sampling* to reduce the impact a large number of training examples has on the complexity of a rule learning algorithm. We represent a subset that results from such a sampling scheme in terms of a weight vector $\boldsymbol{w} \in \mathbb{R}^N$, where each element $w_n \in \boldsymbol{w}$ assigns a real-valued weight to the corresponding example $\boldsymbol{x}_n$. On the one hand, if $w_n = 0$, the rule induction algorithm ignores the corresponding example. We refer to such examples as *out-of-sample*. On the other hand, if $w_n > 0$, the corresponding example is included in the sample used for the construction of candidate rules. Rules are obliged to predict accurately for examples with larger weights, as they are considered more important than examples with smaller weights. If no sampling scheme is used, all examples are weighted equally, i.e., $w_n = 1, \forall 1 \leq n \leq N$. As shown in Figure 4.2, all candidate rules are constructed with respect to the same weights to ensure that they are comparable to each other. Different sampling methods can be used for dimensionality reduction and ensuring diversity within a model. For example, they are employed in random forests (Breiman, 2001) or stochastic gradient boosting (J. H. Friedman, 2002). The out-of-sample examples that result from applying a sampling method can also be helpful to obtain an unbiased estimate of a rule's quality on independent training examples that have not been used to construct the rule. For example, out-of-sample estimates can be used to prune individual rules once they have been learned (Fürnkranz and Widmer, 1994), as discussed below.

Label Sampling

Unlike in binary or multi-class classification, where each example is always associated with exactly one class, the dimensionality of the output space may vastly differ in different applications of multi-label classification, as can be seen in Table 3.1. In contrast to many problem transformation methods, where individual base classifiers are restricted to a single or several predetermined labels (cf. Section 3.6), problem adaptation methods have access to information about the entire label space. It is given as a *label matrix* $Y \in \{0,1\}^{N \times K}$, which stores the ground truth labels of all available training examples. Inducing a new rule requires selecting the labels it should predict for and deciding on the predictions that should be assigned to these labels by its head. Consequently, the complexity of the rule induction algorithm is directly affected by the number of labels in a dataset. A subset of the available labels can optionally be selected via *label sampling* to overcome the computational demands that result from a high-dimensional label space. We denote the set of labels that must be taken into account for generating candidate rules as $\mathcal{L}' \subseteq \mathcal{L}$. Instead of choosing $\mathcal{L}'$ at random, the ground truth labels Y and the statistics S, which incorporate information about a model's predictions, may be taken into account for a more informed selection. For example, a viable strategy is to focus on labels that the current model inadequately handles.

Construction of Bodies and Heads

As mentioned above, the induction of a new rule is based on evaluating a large number of candidate rules. Among these candidates, the best one is selected and added to the model. The generation of a single candidate requires identifying the conditions to be included in its body and constructing a corresponding head that provides accurate predictions for the training examples that satisfy these conditions:

- **Enumeration of Rule Bodies.** Many rule learning algorithms conduct a *top-down search* to search for potential bodies. This includes methods for single-label classification, such as FOIL (Quinlan, 1990), CN2 (Clark and Boswell, 1991), RIPPER (Cohen, 1995), or ENDER (Dembczyński, Kotłowski, and Słowiński, 2010), as well as approaches to multi-label classification (Klein, Rapp, and Loza Mencía, 2019; Loza Mencía and Janssen, 2016; Rapp, Loza Mencía, and Fürnkranz, 2018). A top-down search starts with an empty body that is iteratively refined by adding new conditions.[3] Adding a condition results in fewer examples being covered, i.e., the rule is successively tailored to a certain feature subspace. We discuss the details of top-down rule induction in Section 6.3. Refinements of a body are usually selected greedily, i.e., at each iteration, the search focuses on a single refinement. However, when learning small models, where individual rules have a large impact on the prediction for unseen examples, the accuracy of rules may benefit from an expansion of the search space, as achievable by the conduction of a *beam search*. Rather than focusing on a single refinement, a beam search explores a fixed number of alternatives (Fürnkranz, Gamberger, and Lavrač, 2012). More recently, *branch-and-bound* algorithms that rely on theoretical guarantees to prune the search

3: It is also possible to use a *bottom-up search*, where rules are successively generalized by removing conditions. However, this approach has been found to be problematic when learning from dispersed training examples (Fürnkranz, 2002).

space have been proposed to construct certifiably optimal rules (Angelino et al., 2018; Boley et al., 2021; Webb, 1995). The exploitation of theoretical properties also forms the foundation of approaches based on association rule mining. They have successfully been used to generate candidate rules in the multi-label setting (Lakkaraju, Bach, and Leskovec, 2016; B. Li et al., 2008; Thabtah, Cowling, and Peng, 2004, 2006). Instead of directly constructing the bodies of rules, many approaches that generate candidates from decision trees can be found in the literature. This includes algorithms, such as *RuleFit* (J. H. Friedman and Popescu, 2008) or *SIRUS* (Bénard et al., 2021), that deduce small and interpretable rule sets from much larger and less comprehensible tree-based models. Moreover, the corpus of candidate bodies, which may be derived from an ensemble of decision trees, has also been shown to be well-suited for the empirical comparison of different strategies for rule evaluation and selection (Rapp, Loza Mencía, and Fürnkranz, 2019).

- **Construction of Corresponding Heads.** For each candidate body that is considered when searching for a new rule, a corresponding head must be constructed. As a rule only provides predictions for examples covered by its body, its head should be tailored to the covered training examples. To consider the ground truth labels of these examples and the corresponding predictions of the current model, we rely on the label space statistics mentioned earlier to construct the head of a rule. It is built with respect to a subset of the statistics $S' \subseteq S$ corresponding to the covered training examples. Once the predictions provided by a rule's head have been determined, an estimate of the rule's quality can be computed. It serves as the basis for comparing different candidate rules to each other and deciding on the best one. We elaborate on the construction of rule heads and the assessment of their quality in the following section "Evaluation of Candidates".

Tie Breaking

Two or more candidate rules may be assigned the same quality in practice. Rule learning algorithms often employ a *tie-breaking* strategy to decide which one should be preferred. A commonly used rule of thumb is to prefer more general rules, i.e., rules that contain fewer conditions in their bodies. Alternatively, rules covering fewer training examples may be preferred, as they are less prone to overfit the training data (Fürnkranz, Gamberger, and Lavrač, 2012).

Incremental Reduced Error Pruning

Once a rule induction algorithm has decided on the best rule to be added to a model, additional effort may be put into optimizing this particular rule. This is especially relevant when learning small models, where inaccurate rules heavily affect predictive accuracy. In particular, research on single-label rule learning has shown that the quality of rules can substantially be improved by *incremental reduced error pruning* (IREP) (Fürnkranz and Widmer, 1994) while being computationally more efficient than post-pruning strategies. IREP, as used by the rule learning algorithm of the same name and later employed by more advanced rule learners, such as RIPPER (Cohen, 1995) or SLIPPER (Cohen and Singer, 1999), aims at optimizing individual rules by removing trailing conditions from their bodies that may result in overfitting of the training data. This requires applying an instance sampling method whenever a new rule should be learned to partition the set of available training examples into two distinct subsets. While the examples drawn by the sampling method are used to construct a prototypical rule, the out-of-sample examples are used to prune it afterward. We refer to the former as the *grow set*, whereas the latter is called the *prune set*.[4] The prune set is used to obtain an unbiased estimate of the rule's quality when applied to examples that have not been considered for its construction. If a rule overfits the examples in the grow set, it most likely performs poorly on the prune set. Removing conditions from its body and keeping track of the resulting changes in quality, measured on the prune set, enables to find a modification of the rule's original body that can be expected to better generalize to unseen data. In multi-label classification, where rules may predict for several labels simultaneously, it is possible to not only modify the body of a rule but also its head. Removing conditions from the body causes a rule to become more general, i.e., it is likely to cover more examples. For some labels, this may result in a deterioration of predictive accuracy if the distribution of the corresponding ground truth labels changes due to covering a larger feature subspace. In such a case, it might be beneficial to adjust the set of predicted labels to account for the changes in the covered region. A systematic investigation of pruning methods for the application in multi-label rule learning algorithms remains for future research.

4: Typically, one strives for a ratio of 2:1 when partitioning the training data into a grow and a prune set (Cohen, 1995; Cohen and Singer, 1999; Fürnkranz and Widmer, 1994).

Evaluation of Candidates

Construction of Rule Heads

As mentioned before, for each candidate body considered by a rule induction algorithm, a corresponding head must be found. As discussed in Section 4.1, different types of multi-label heads are conceivable. Accordingly, different implementations for the construction of heads are necessary, depending on whether rules should be concerned with one or several labels and whether deterministic or probabilistic predictions are desired. Despite these variants, the construction of rule heads generally includes the following steps:

I. **Label Selection.** Unless concerned with the induction of complete rules that take all available labels into account, the construction of rule heads requires selecting a subset of labels to predict for. Compared to single-label heads, where a linear search may be used to identify the label for which a rule can provide the most accurate prediction, the construction of partial heads is more challenging. The latter is closely related to multi-label classification with *partial abstention* (Nguyen and Hüllermeier, 2020), where classifiers are allowed to deliver predictions only for labels they are most certain about while abstaining on others. Both MLC with partial abstention and the creation of partial multi-label heads demands a trade-off between the accuracy of predictions and their completeness with respect to the available labels. Unfortunately, due to the combinatorial complexity that results from a large number of labels, it is impractical to take all possible label subsets into account.[5] For this reason, alternatives to an exhaustive search among all possible label combinations, based on the theoretical properties of commonly used multi-label rule learning heuristics, were investigated in the past (Rapp, Loza Mencía, and Fürnkranz, 2018). By controlling

5: Given K labels, $2^K - 1$ different combinations of labels exist.

the trade-off between the accuracy and completeness of multi-label heads via parameterization, they can successfully be used in separate-and-conquer rule learning (cf. Section 5.4). Similarly, Nguyen and Hüllermeier (2020) provide solutions for the efficient computation of partial predictions with respect to different evaluation measures. The problem has also been studied in the context of multi-output decision trees that store predictions for several labels in their leaf nodes. Si et al. (2017) use an approach based on gradient boosting, where partial predictions for a fixed number of labels are deduced from first-order gradients. Z. Zhang and C. Jung (2020) extend this idea by the ability to take second-order gradients into account to decide which labels should be addressed by a leaf.

II. **Computation of Predictions.** Once a set of labels to be included in a rule's head has been found, the values $\hat{p}_k$ that should be assigned to the respective labels λ_k must be determined. On the one hand, when concerned with deterministic rules, i.e., if individual labels should be predicted as either relevant ($\hat{p}_k = 1$) or irrelevant ($\hat{p}_k = 0$), the predicted values should be chosen such that they are correct for most covered training examples. Some algorithms (see, e.g., Rapp, Loza Mencía, and Fürnkranz, 2019) are restricted to rules that model the relevance (or irrelevance) of labels. If an example is not covered by any of the rules, the default rule applies and predicts previously unassigned labels as irrelevant (or relevant). The models that result from such approaches are easy to interpret, as no conflicts between the prediction of rules may arise, even if several rules cover an example.[6] On the other hand, probabilistic rules can provide information about the label distribution of examples they cover. In particular, their predictions for individual labels may reflect conditional probabilities, as implemented by RIPPER (Cohen, 1995) and often used in ensembles of decision trees, e.g., in random forests (Breiman, 2001). However, alternative representations of probabilistic predictions are conceivable. For example, the statistical uncertainty of predictions may be taken into account when specifying the number of relevant and irrelevant examples for a label (Hüllermeier and Waegeman, 2021). In boosting algorithms like ENDER (Dembczyński, Kotłowski, and Słowiński, 2010), where rules assign real-valued scores $\hat{p}_k \in \mathbb{R}$ to individual labels, the semantics of the predicted values depend on the loss function to be minimized during training. As discussed in Chapter 6, they are derived from the statistics S', resulting from the loss function's partial derivatives.

6: Models that exclusively focus on the relevance or irrelevance of labels are also referred to be given in *disjunctive normal form* (DNF) or *conjunctive normal form* (CNF), depending on whether logical AND or OR operators are used to concatenate the conditions in the rules' bodies (Fürnkranz, Gamberger, and Lavrač, 2012).

III. **Quality Assessment.** Comparing different candidate rules to each other requires assessing the quality of their predictions in terms of a numerical confidence score $q \in \mathbb{R}$. Many traditional rule learners rely on *heuristics* to estimate a rule's quality based on confusion matrices. A large number of rule learning heuristics have been investigated in the literature (see, e.g., Fürnkranz and Flach, 2005; Fürnkranz, Gamberger, and Lavrač, 2012; Janssen and Fürnkranz, 2010, for an overview). This includes parameterized heuristics, such as the F-measure (Rijsbergen, 1979) or the M-estimate (Cestnik, 1990), that allow for a flexible trade-off between a rule's consistency and coverage. In Section 5.3, we show that separate-and-conquer rule learning algorithms can be tailored to different multi-label evaluation measures by weighing these two aspects accordingly. When dealing with rules that predict for several labels simultaneously, it is necessary to obtain a single score that reflects the overall quality of a rule, considering all labels in its head. Similar to the bipartition evaluation metrics discussed in Section 3.3, different aggregation and averaging strategies are conceivable when using heuristics to assess the quality of partial or complete multi-label rules (Rapp, Loza Mencía, and Fürnkranz, 2018). With traditional heuristics, rules are evaluated in isolation, i.e., previously induced rules do not affect the assessment of their quality. This is different from boosting algorithms, where rules are rated by the improvement they introduce to an existing model. As discussed in Chapter 6, they assess the quality of rules in terms of an *objective function* that depends on a particular loss function and may include an optional *regularization term* to penalize overly specific rules.

Shrinkage

Once the head of a rule has been determined, the predictions it consists of can optionally be post-processed. This is particularly relevant when learning probabilistic rules (cf. Section 4.1), where the real-valued scores $\hat{p}_k \in \mathbb{R}$ assigned to individual labels λ_k may be multiplied by a parameter $\eta \in (0, 1)$. This technique, which reduces the impact of individual rules on the model, is known as *shrinkage* (Hastie, Tibshirani, and J. H. Friedman, 2009). It may help to prevent overfitting when learning large numbers of rules. In the literature on gradient descent, the parameter η is commonly referred to as the *learning rate*.

4.4 Rule-based Prediction

Determining Predictions

After a rule model has been trained on labeled training data, it can be used to obtain predictions for unseen examples. The method employed for prediction should be chosen in accordance with the training algorithm used to induce the rules in the first place. In any case, deducing the predictions for an example from a rule model always involves the following two steps:

I. **Test for Coverage.** As rules only predict for examples that satisfy the conditions in their bodies, the fraction of rules that cover a given example must be identified. This always includes the default rule, which applies to all examples.

II. **Aggregation of Predictions.** A final consistent prediction for the available labels must be deduced from the heads of the rules that cover a given example. Depending on the types of heads, their predictions may contradict each other. The method that is used to aggregate the information they entail must be able to resolve such conflicts. Ideally, the aggregation scheme should be tailored to a specific evaluation measure, which the model aims to optimize.

Decision Lists

Deterministic rules induced by a separate-and-conquer algorithm are usually given in the form of a *decision list*. It stores the rules in a particular order (typically the order of their induction). The default rule is always appended to the end. When predicting for unseen examples, the rules are processed in the given order, i.e., rules that have been induced earlier take precedence over their successors. This enables to account for the fact that later iterations of a SeCo algorithm are restricted to feature subspaces that have not been covered yet. As a result, rules constructed during later iterations are not evaluated with respect to the entire feature space but must only predict accurately for yet uncovered regions. Nevertheless, they may overlap with previously covered regions for which one of the previous rules predicts more accurately. To counteract this issue, examples are always handled by the first rule they satisfy. This ensures that rules only predict for regions they have been tailored to. When dealing with models that meet the properties of a DNF or CNF, i.e., if no contradictions between the predictions of individual rules may arise, the final predictions are invariant to the order of the rules, as long as the default rule comes last. Loza Mencía and Janssen (2016) introduce the notion of *multi-label decision lists*. With their approach, the single-label heads of rules are applied to a given example only if the respective label has not been dealt with by one of the previous rules. This concept can also be used to obtain predictions from multi-label rules that are concerned with several labels at the same time (Klein, Rapp, and Loza Mencía, 2019; Rapp, Loza Mencía, and Fürnkranz, 2018).

Linear Combination of Rules

If the rules in a model are not inherently tailored to different regions of the feature space, as with SeCo approaches, the prediction for an example usually results from a linear combination of rules. This applies to boosting methods, weighted covering, and ensembles of rules deduced from random samples of the training data. When dealing with deterministic rules that assign binary predictions $\hat{p}_k \in \{0, 1\}$ to individual labels, an unweighted voting scheme may be employed to determine the final prediction for a single label or even entire labelsets. In such a case, each rule that covers a given example provides a vote for predicting an individual label λ_k as relevant ($\hat{p}_k = 1$) or irrelevant ($\hat{p}_k = 0$). Similarly, weighted voting may be conducted when dealing with probabilistic predictions $\hat{p}_k \in \mathbb{R}$, which express the degree to which a label is considered to be relevant or irrelevant by a particular rule (cf. Section 4.1). Probabilistic rules may also provide more detailed information about the label distribution in the region they cover. Such a representation enables to use a wide variety of methods for the aggregation of predictions, commonly used in ensembles of decision trees. This includes, but is not restricted to, *Laplace correction* (Provost and Domingos, 2003), techniques based on the *Dempster-Shafer theory of evidence* (Lu, 1996), the *cautious rule of combination* (Denœux, 2006), or approaches that measure *aleatoric* and *epistemic uncertainty* (Hüllermeier and Waegeman, 2021).

Statistically Optimal Prediction

As different evaluation metrics with varying characteristics exist in multi-label classification, a model should ideally be built with respect to a specific target measure. However, to obtain predictions that can be expected to be optimal with respect to a certain measure, not only the construction of a rule-based model but also the aggregation of individual rules at prediction time should be tailored to the measure at hand. Whereas decomposable evaluation metrics, such as the Hamming loss, can easily be optimized via label-wise aggregation (Dembczyński, Cheng, and Hüllermeier, 2010), the optimization of non-decomposable metrics is more challenging. Consequently, several publications have been devoted to the problem of obtaining statistically optimal predictions from multi-label classifiers. For example, this includes algorithms for maximizing the F-measure (Cheng et al., 2012; Dembczyński, Waegeman, et al., 2011; Jasinska et al., 2016; Waegeman et al., 2014), which can potentially be applied to rule-based models if they provide information about the distribution of labels, e.g., in the form of probabilities. Similarly, techniques that aim to improve the predictive performance of a classifier in terms of the subset 0/1 loss have been proposed in the literature (Nam et al., 2017; Senge, Coz, and Hüllermeier, 2013). Also closely related to rule-based prediction, where the information provided by individual rules must be aggregated into an overall prediction, is the work by Nguyen, Hüllermeier, et al. (2020) . It investigates different possibilities to aggregate complete predictions provided by the individual classifiers in an ensemble and includes an evaluation of their suitability for the optimization of commonly used multi-label metrics. Instead of focusing on a specific target metric, Papagiannopoulou, Tsoumakas, and Tsamardinos (2015) propose a probabilistic framework for rectifying the probabilistic predictions of MLC models, which can be applied to different types of models, including rule-based ones. It aims to enforce adherence to the marginal label dependencies that are discovered in a dataset.

Nguyen, Hüllermeier, Rapp, Loza Mencía, and Fürnkranz (2020): 'On Aggregation in Ensembles of Multilabel Classifiers'

4.5 Discussion

In this chapter, we presented a unified view of existing rule learning approaches that have explicitly been designed to meet the requirements of multi-label classification problems. Rather than discussing the algorithmic details of individual methods, the goal of this chapter was to highlight the various aspects that are essential to rule-based adaptation methods and point out different techniques that may be used to implement them. This high-level view of the problem domain is complemented by Chapter 5 and Chapter 6, where the underlying methodology and technical details of two different rule learning approaches are discussed. Whereas one of these approaches is motivated by the need for human-interpretable models and uses the separate-and-conquer paradigm, the other focuses on predictive performance and relies on the gradient boosting framework. Despite these fundamental differences regarding the construction and evaluation of candidate rules, both algorithms share many similarities, such as the use of a greedy top-down search for the refinement of rules or the possible utilization of sampling techniques. The implementation of said methods benefits from the modular framework that was proposed in this chapter, as its modularity allows to focus on the algorithmic differences rather than re-implementing functionality that both approaches have in common. It also allows both approaches to benefit from optimizations and approximations for the effective induction of rules that we elaborate on in Chapter 7. Moreover, the versatility of the proposed framework is illustrated by a rule learning approach for application to medical data recently proposed by Rapp, Kulessa, et al. (2021) . It aims to discover indicators for infectious diseases in medical records, based on their correlation to the number of infections that have been reported to public health institutions for corresponding periods in time. Even though this algorithm is not aimed at multi-label classification but addresses a quite different learning task instead, it resembles many algorithmic aspects that have been outlined in this chapter.

Rapp, Kulessa, Loza Mencía, and Fürnkranz (2021): 'Correlation-based Discovery of Disease Patterns for Syndromic Surveillance'

Induction of Multi-label Rules using the Separate-and-Conquer Approach

5

As discussed in Section 2.2, many traditional rule learning approaches for binary or multi-class classification, such as FOIL (Quinlan, 1990), CN2 (Clark and Boswell, 1991), or RIPPER (Cohen, 1995), use the separate-and-conquer paradigm for the assemblage of rule-based models. In this chapter, we discuss a series of publications that focus on a generalization of the SeCo covering algorithm to multi-label classification problems. Learning algorithms that follow this particular approach are mainly motivated by the need for simple models that can easily be analyzed and inspected by humans. SeCo-based rule learning methods do not only meet this requirement because they induce deterministic rules that indicate whether individual labels are relevant or irrelevant to the examples they cover, but also because they are well-suited for the construction of DNFs. When learning a DNF, all rules are obliged to focus on either the presence or absence of a particular label, depending on whether it is irrelevant or relevant to the majority of examples, i.e., the individual rules are meant to provide predictions for examples and labels that are not correctly dealt with by a simple default rule. A model of this kind is straightforward to comprehend, as the predictions of different rules may not conflict with each other. Consequently, each rule can be considered a local explanation that applies to all examples it covers, regardless of the remaining rules or the order in which they have been learned. Following a discussion of multi-label rule learning heuristics in Section 5.2, in Section 5.3, we present an empirical study that emphasizes the need for configurable learners that can flexibly use different heuristics, depending on which evaluation measure should be optimized by a model. Whereas said study is restricted to single-label rules, we elaborate on the possibility of capturing local label dependencies through multi-label heads in Section 5.4. In contrast to label-dependent rules, which allow expressing global and local dependencies by taking the predictions of previously induced rules into account (cf. Section 4.1), multi-label heads enable single rules to provide a joint prediction for several labels. Multi-label rules must not be intepreted in the context of their predecessors and therefore are more in line with the properties of a DNF.

5.1 The Label Covering Problem

Partially Covered Examples

As previously outlined in Section 2.2, a SeCo algorithm induces rules in a sequential manner, where one rule is added after the other. When dealing with single-label problems, the examples it covers are removed from the training dataset whenever a new rule has been learned. This is referred to as the "separate"-step. Unfortunately, a generalization of this strategy to the multi-label setting is not straightforward. As discussed in Section 4.1, the predictions of multi-label rules may be restricted to a subset of the available labels or even a single label. Consequently, during the covering process, the existing rules are likely to provide predictions for some labels of an example but not for others. Hence, a suitable strategy is needed to separate uncovered examples from (partially) covered ones and decide at which point an example should be removed from the training data.

Separation Strategies

This problem is also acknowledged by Loza Mencía and Janssen (2016). The authors present the first attempt to apply the SeCo paradigm to multi-label classification problems and investigate several strategies to deal with partially covered examples. On the one hand, they consider removing an example as soon as it is covered by at least one rule, even if this rule does not predict for all available labels. On the other hand, they retain covered examples in the training dataset unless their labels are fully covered or partially covered to at least a certain extent. As it remains unclear which of these variants should be preferred, in the following, we introduce a flexible methodology that allows implementing each one of them.

Label Weights

Initializing and Updating Label Weights

We use a matrix $W \in \{0, 1\}^{N \times K}$ to keep track of the examples and labels covered by previously induced rules. Each element in the matrix assigns a binary weight w_{nk} to the k-th label of the n-th example. Initially, to indicate that all examples and labels are fully uncovered, all elements in the matrix are set to one. Whenever a new rule has been learned, the weights that correspond to examples covered by the rule must be updated. This requires setting the weights of labels for which the new rule predicts to zero, whereas the other weights remain unchanged.

Multi-label Covering

The structure of a multi-label SeCo algorithm[1] that uses a weight matrix W is outlined in Algorithm 1. It starts by constructing a default rule that provides a prediction for all available labels. Typically, the default rule predicts the *majority value* $\Lambda_k \in \{0, 1\}$ for each available label. The majority value for a particular label λ_k indicates whether the label is associated with most of the provided training examples ($\Lambda_k = 1$) or if it is absent from most training examples ($\Lambda_k = 0$). Afterward, label space statistics S, which serve as a basis for learning additional rules with respect to a heuristic $\mathcal{H}$ are computed. As discussed in Section 5.2 below, the label space statistics are given as binary confusion matrices that specify for each example and label whether the prediction of a

1: By using a weight matrix with real-valued weights, as well as a suitable strategy for updating them, the proposed methodology can easily be generalized to a weighted covering algorithm. However, we consider such an extension to be out of the scope of this chapter.

Algorithm 1: A multi-label separate-and-conquer algorithm

input : Training examples $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_n^N$, heuristic $\mathcal{H}$, at least one stopping criterion
output: List of rules F

$f_1 : \hat{\boldsymbol{y}}_1 \leftarrow b_1$ = induce default rule
S = compute label space statistics
W = set weight of each example and label to 1
for $t = 2, 3, \dots$ **until** a stopping criterion is met **do**
    $\boldsymbol{w}_t$ = obtain a weight for each example via instance sampling
    $f_t : \hat{\boldsymbol{y}}_t \leftarrow b_t$ = induce best rule w.r.t. $\mathcal{H}$ and S, ignoring examples and labels with zero weights according to $\boldsymbol{w}_t$ or W
    W = UPDATE_WEIGHT_MATRIX $(\mathcal{D}, W, f_t)$
return list of rules $F = (f_2, f_3, \dots, f_1)$

Algorithm 2: UPDATE_WEIGHT_MATRIX

input : Training examples $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_n^N$, weight matrix W, rule $f : \hat{\boldsymbol{y}} \leftarrow b$
output : Updated weight matrix W

foreach example $\boldsymbol{x}_n$ that satisfies the body b **do**
  for $\hat{p}_k \in \hat{\boldsymbol{y}}$ **do**
    $w_{nk} \in W = 0$
return weight matrix W

candidate rule is correct or incorrect. Unlike the statistics S, which only depend on the training examples' ground truth labels, the weight matrix W is updated at each iteration of the covering process according to Algorithm 2. Examples and labels for which the corresponding weight has been set to zero must be ignored when assessing the quality of potential rules. In addition, an instance sampling method that assigns a weight to each training example can optionally be used. This enables the implementation of different strategies to deal with partially covered examples. For example, if partially covered examples, for which most labels have already been dealt with, should be removed from the training process, they may be given zero weights.

Stopping Criteria

Terminating the Covering Process

As can be seen in Algorithm 1, the induction of new rules comes to an end as soon as a certain stopping criterion is met. The investigation of suitable pre-pruning techniques has traditionally been an essential aspect of research on rule learning methods (see, e.g., Fürnkranz and Flach, 2004, for an analysis of different techniques). They aim to prevent overfitting by discouraging the inclusion of overly specific rules in a model. This goal can either be achieved by early termination of a SeCo algorithm or by filtering suboptimal rules once training has finished. For example, *CN2* (Clark and Boswell, 1991) and FOSSIL (Fürnkranz, 1994) employ a filter criterion to decide whether individual rules should be excluded from a model, whereas FOIL (Quinlan, 1990) and RIPPER (Cohen, 1995) rely on the information-theoretic MDL stopping criterion. Unfortunately, there is no prior work on generalizing these approaches from single-label classification to multi-label problems. Even though their application to multiple labels is most probably feasible in a binary relevance fashion, it can be expected that the optimization of different multi-label evaluation measures demands varying implementations of pre-pruning techniques.

Coverage Stopping Criterion

Due to the lack of better solutions, we rely on a rather conservative *coverage stopping criterion* in the remainder of this chapter. It stops the induction of rules as soon as all examples and labels for which the default rule provides an incorrect prediction are covered, i.e., as soon as the sum

$$\sum_{n=1}^{N} \sum_{k=1}^{K} \left(w_{nk} \cdot [\![y_{nk} \neq \Lambda_k]\!] \right) \tag{5.1}$$

equals zero or falls below a predefined threshold.

Algorithm 3: Application of a multi-label decision list to an example

input : Example $\boldsymbol{x}$, multi-label decision list F
output: Prediction $\hat{\boldsymbol{y}}$

$\hat{\boldsymbol{y}} = (?, ?, \ldots, ?)$
foreach rule $f : \hat{\boldsymbol{p}} \leftarrow b \in F$ **do**
  if example $\boldsymbol{x}$ satisfies the body b **then**
    foreach $\hat{p}_k \in \hat{\boldsymbol{p}}$ **do**
      if $\hat{y}_k \in \hat{\boldsymbol{y}} \neq ?$ **then**
        $\hat{y}_k = \hat{p}_k$
return prediction $\hat{\boldsymbol{y}}$

Prediction for Several Labels

Multi-label Decision Lists

As seen in Algorithm 1, the rules that result from a multi-label SeCo algorithm are given in the order of their induction. The default rule is always located at the end. Loza Mencía and Janssen (2016) refer to such a model as a *multi-label decision list*. It requires the order of the rules to be taken into account when obtaining predictions for unseen examples. As label space statistics, which have been assigned zero weights during previous iterations of the training algorithm, do not affect the quality of a rule, it may provide inaccurate predictions for the corresponding examples and labels. To counteract this potential problem, the predictions of rules that have been induced early in the training process should be given greater priority than the predictions of their successors.

Determining Predictions

Algorithm 3 illustrates how predictions for a given example can be obtained from a multi-label decision list. For this purpose, the rules in the model are processed in the given order. If a rule covers an example, its prediction for a particular label only takes effect if none of the previously processed rules have provided a prediction for the respective label. The default rule, located at the end of a multi-label decision list, is responsible for assigning a default prediction to all labels that have not been dealt with by the previous rules. If the model assembled by a multi-label SeCo algorithm satisfies the properties of a DNF, all rules, except for the default rule, provide the same prediction for a particular label. In such a case, Algorithm 3 does not depend on the order of the rules. However, even in this restricted setting, the default rule must only be applied to labels that have not been handled by one of the other rules.

5.2 Multi-label Heuristics

Consistency-Coverage Trade-off

Rule learning algorithms based on the separate-and-conquer paradigm usually employ a heuristic-guided search for rules that model regularities in the training data. All common rule learning heuristics weigh between two aspects, namely consistency and coverage, and it is commonly accepted that the choice of the heuristic has a significant impact on the predictive performance of a learning algorithm (Fürnkranz and Flach, 2005; Fürnkranz, Gamberger, and Lavrač, 2012; Janssen and Fürnkranz, 2010). On the one hand, rules should be consistent, i.e., their predictions should be correct for as many of the covered examples as possible. On the other hand, rules with great coverage, i.e., rules that cover many examples, tend to be more reliable, even though they may be less consistent.

Traditional Notation

Two-Dimensional Confusion Matrices

Rule learning heuristics are most commonly denoted as mathematical functions $\mathcal{H} : \mathbb{N}^{2\times 2} \rightarrow \mathbb{R}$, which map a two-dimensional confusion matrix to a real-valued score that allows comparing the estimated quality of different candidate rules. Traditionally, as shown in Table 3.3, a confusion matrix consists of the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) predicted by a rule. Given a ground truth label $y_{nk} \in \{0, 1\}$ and the prediction of a deterministic rule $\hat{p}_{nk} \in \{0, 1\}$ for a particular label, the elements of a binary confusion matrix C_{nk} are defined as follows.

$$\begin{aligned} TP &:= [\![y_{nk} = 1 \land \hat{p}_{nk} = 1]\!] \quad & FP &:= [\![y_{nk} = 0 \land \hat{p}_{nk} = 1]\!] \\ TN &:= [\![y_{nk} = 0 \land \hat{p}_{nk} = 0]\!] \quad & FN &:= [\![y_{nk} = 1 \land \hat{p}_{nk} = 0]\!] \end{aligned} \tag{5.2}$$

Limitations of the Notation

However, in a multi-label setting, we have found this notation misleading and limiting at times. In binary classification, a rule learner is typically restricted to rules covering examples of the minority class, whereas the default rule deals with examples that correspond to the majority class. In such a case, the terms “true positives” and “false negatives” refer to examples of the minority class, whereas examples of the majority class are referred to as “true negatives” or “false positives”. In multi-label classification, where each example may be associated with several labels, the distribution of individual labels may vary. Consequently, the semantics of the terms above differ from label to label. If the rules are not obliged to predict the same outcome for a single label but are free to predict positively or negatively, these terms become even more confusing. Furthermore, the term “negatives” does not distinguish between labels of covered examples for which a rule explicitly provides a negative prediction and labels of uncovered examples that are not affected by a rule’s predictions at all. In the following, we propose an alternative notation that aims to overcome these practical issues and limitation.

A More Flexible Notation

Three-Dimsionsal Confusion Matrices

To facilitate the definition of rule learning heuristics in a multi-label classification setting, we propose using three-dimensional confusion matrices consisting of eight elements in total. Multi-label rule learning heuristics that assess the quality of rules in terms of such confusion matrices can be viewed as functions $\mathcal{H} : \mathbb{N}^{2\times 2\times 2} \rightarrow \mathbb{R}$. To refer to the different confusion matrix elements, we use names that consist of three symbols. Each of these symbols corresponds to a certain aspect characterizing the prediction of a binary rule for a particular example and label. The first symbol specifies whether the example is “covered” (C) or “uncovered” (U) by the rule. The second symbol identifies “relevant” (R) or “irrelevant” (I) labels according to ground truth. The third symbol depends on whether the rule predicts “positively” (P) or “negatively” (N) for the respective label. Accordingly, the elements of a binary and three-dimensional confusion matrix C_{nk} that characterizes a prediction for an example $\boldsymbol{x}_n$ and label λ_k are defined as follows.

$$\begin{aligned}
CRP &:= [\![\boldsymbol{x}_n \text{ is covered } \land y_{nk} = 1 \land \hat{p}_{nk} = 1]\!] \\
CRN &:= [\![\boldsymbol{x}_n \text{ is covered } \land y_{nk} = 1 \land \hat{p}_{nk} = 0]\!] \\
CIP &:= [\![\boldsymbol{x}_n \text{ is covered } \land y_{nk} = 0 \land \hat{p}_{nk} = 1]\!] \\
CIN &:= [\![\boldsymbol{x}_n \text{ is covered } \land y_{nk} = 0 \land \hat{p}_{nk} = 0]\!] \\
URP &:= [\![\boldsymbol{x}_n \text{ is uncovered } \land y_{nk} = 1 \land \hat{p}_{nk} = 1]\!] \\
URN &:= [\![\boldsymbol{x}_n \text{ is uncovered } \land y_{nk} = 1 \land \hat{p}_{nk} = 0]\!] \\
UIN &:= [\![\boldsymbol{x}_n \text{ is uncovered } \land y_{nk} = 0 \land \hat{p}_{nk} = 1]\!] \\
UIP &:= [\![\boldsymbol{x}_n \text{ is uncovered } \land y_{nk} = 0 \land \hat{p}_{nk} = 0]\!]
\end{aligned} \tag{5.3}$$

Shorthand Notations

We further introduce the following shorthand notations to denote examples and labels that are covered by a rule and for which the rule's prediction is correct or incorrect, respectively.

$$\begin{aligned}
C_{correct} &:= CRP + CIN \\
C_{incorrect} &:= CRN + CIP
\end{aligned} \tag{5.4}$$

Accordingly, the following symbols refer to labels of uncovered examples for which the prediction of a rule would be correct or incorrect if covered by the rule.

$$\begin{aligned}
U_{correct} &:= URP + UIN \\
U_{incorrect} &:= URN + UIP
\end{aligned} \tag{5.5}$$

The shorthand notations given above allow defining multi-label rule learning heuristics in a simple and unambiguous way.

Selected Heuristics

Generalization to Multiple Labels

In the following, we use the previously introduced notation to introduce a selection of rule learning heuristics used in the remainder of this chapter. All of the considered heuristics are known from the single-label classification setting. However, if a rule is allowed to predict for several simultaneously, different possibilities exist to obtain an overall estimate of the rule's quality across the respective labels. For this purpose, the different averaging strategies that are discussed in Section 3.3 can be used.

Label-wise averaged Heuristics

First of all, an aggregated confusion matrix can be obtained for each label individually by aggregating the corresponding binary confusion matrices across all available training examples, as shown in (3.12). In this case, an estimate of a rule's quality with respect to multiple labels is computed by applying a given heuristic function to each and averaging the resulting scores. The overall quality of a rule according to such a *label-wise averaged heuristic* calculates as

$$\frac{1}{|h|} \sum_{\hat{p}_k \in \hat{\boldsymbol{p}}} \mathcal{H}(\sum_{n=1}^{N} C_{nk} w_{nk}), \tag{5.6}$$

where $|\hat{\boldsymbol{p}}|$ denotes the number of labels for which a head $\hat{\boldsymbol{p}}$ predicts and the values $\hat{p}_k \in \{0, 1\}$ correspond to discrete predictions it provides for the respective labels. As discussed in Section 5.1, the binary weights $w_{nk} \in \{0, 1\}$ specify whether individual examples and labels should affect the quality of a rule or whether they should be ignored, e.g., because they have already been dealt with by one of the previously induced rules.

Micro-averaged Heuristics

In addition, we also consider *micro-averaged heuristics*, where a heuristic function is applied to a single confusion matrix resulting from an aggregation across all available examples and labels, similar to (3.11). A heuristic of this kind computes the overall quality of a rule as

$$\mathcal{H}(\sum_{n=1}^{N} \sum_{\hat{p}_k \in \hat{\boldsymbol{p}}} C_{nk} w_{nk}). \tag{5.7}$$

Whereas the computation of an evaluation score according to the former strategy satisfies the properties of label-wise decomposability in (3.26), this is not necessarily the case when using the latter strategy.

Hamming Accuracy Heuristic

First of all, we consider a heuristic function that is similar to the Hamming loss in (3.9). In contrast to loss functions like the Hamming loss, where smaller values indicate better performance, rule learning heuristics are traditionally designed such that greater heuristic valus are better. Following this convention, we define the *Hamming accuracy* of a rule as

$$\mathcal{H}_{Hamm.}(C) := \frac{C_{correct} + U_{incorrect}}{C_{correct} + C_{incorrect} + U_{correct} + U_{incorrect}}. \tag{5.8}$$

Precision Heuristic

In addition, we consider a heuristic function that corresponds to the precision metric in (3.17). It can be written as

$$\mathcal{H}_{Prec.}(C) := \frac{C_{correct}}{C_{correct} + C_{incorrect}} \tag{5.9}$$

and corresponds to the number of labels covered and correctly predicted by a rule. Unlike Hamming accuracy, which comes with a reward for labels of uncovered examples for which a rule's prediction would be incorrect if covered, the precision heuristic does only take covered examples into account.

Recall Heuristic

Similarly, we can define a heuristic that corresponds to the recall metric in (3.18) as

$$\mathcal{H}_{Rec.}(C) := \frac{C_{correct}}{C_{correct} + U_{correct}}. \tag{5.10}$$

It assesses the quality of a rule as the fraction of covered examples among all examples for which the rule's predictions would be correct.

F-measure Heuristic

The heuristics in (5.9) and (5.10) are not useful in practice, as they focus exclusively on the consistency or coverage of rules. Instead, rule learning heuristics that strive for a balance between these two aspects are needed. The question of how to trade off the consistency and coverage of rules is especially relevant in multi-label classification, where different and potentially competing evaluation measures exist (cf. Section 3.3). Because a single learner cannot optimize all of them at the same time, the rule learning heuristic it employs should most probably be adjusted to a particular target measure. In Section 5.3, we show empirically that it is necessary to weigh these properties differently, depending on which performance measure should be optimized by a model. The F-measure allows controlling the trade-off between the precision and recall of rules, depending on a user-configurable parameter β. A special case of this performance metric, where $\beta = 1$, is shown in (3.19). In the general case, a multi-label rule learning heuristic that corresponds to the F-measure can be written as

$$\mathcal{H}_F(C) := \frac{\left(1+\beta^2\right) C_{correct}}{\left(1+\beta^2\right) C_{covered} + \beta^2 U_{correct} + C_{incorrect}} \text{ with } \beta \geq 0. \tag{5.11}$$

If $\beta < 1$, the consistency of a rule is considered more important than its consistency. If $\beta > 1$, greater emphasis is put on its coverage instead.

M-estimate Heuristic

The M-estimate (Cestnik, 1990) is another well-known rule learning heuristic that allows controlling the trade-off between consistency and coverage via a user-configurable parameter. We use the previously introduced notation to define this heuristic as

$$\mathcal{H}_M(C) := \frac{C_{correct} + \left(m \frac{C_{correct}+U_{correct}}{C_{correct}+C_{incorrect}+U_{correct}+U_{incorrect}}\right)}{C_{correct} + C_{incorrect} + m} \text{ with } m \geq 0. \tag{5.12}$$

The parameter m controls the trade-off between the consistency and coverage of a rule. If $m = 0$, the M-estimate is equivalent to the precision heuristic in (5.9). As m approaches infinity, the M-estimate becomes equivalent to *weighed relative accuracy* (WRA) (Fürnkranz, Gamberger, and Lavrač, 2012).

5.3 A Study on Rule Selection

Motivation

As previously argued in Section 5.2, the choice of a suitable rule learning heuristic has a significant impact on the effectiveness of a learning algorithm based on the separate-and-conquer paradigm. Whereas the properties of different heuristics have been studied quite extensively in the realm of single-label classification (see, e.g., Fürnkranz and Flach, 2005; Janssen and Fürnkranz, 2008), there is no such work that considers the particularities of multi-label classification. This is surprising as the quality of multi-label predictions is usually assessed in terms of various performance measures that a single learner cannot optimize at the same time (cf. Section 3.3). Even though some of them originate from measures used in binary or multi-class classification, different ways to aggregate the predictions for individual examples and labels, such as example-wise or micro averaging, exist in MLC. Some measures like the subset 0/1 loss are even unique to the multi-label setting. For this reason, it can be expected that different heuristics are needed, depending on which multi-label measure should be optimized by a model.

Goals of the Study

Rapp, Loza Mencía, and Fürnkranz (2019): 'On the Trade-off Between Consistency and Coverage in Multi-label Rule Learning Heuristics'

In the following, we present an experimental study that investigates the trade-off between the consistency and coverage of individual rules in the multi-label classification setting. The goal of this study published by Rapp, Loza Mencía, and Fürnkranz (2019) is to understand better how these two aspects should be weighed to assess the quality of candidate rules during training. This question is especially relevant if one is interested in a model that optimizes a certain multi-label evaluation measure. The study revolves around a method that allows for a flexible assemblage of rule-based models, given a predefined set of candidate rules. Said method enables the use of different multi-label heuristics to select candidate rules that should be included in a model. We analyze empirically how the predictive performance and characteristics of rule-based models are affected by varying heuristics. Furthermore, we demonstrate how models that aim to optimize a given multi-label evaluation measure can deliberately be trained by choosing a suitable heuristic. Finally, by comparing the experimental results to a state-of-the-art rule learner, we emphasize the need for configurable approaches that can flexibly be tailored to different multi-label measures. For reasons of brevity, we restrict the discussion to micro averaged evaluation measures, as well as to the Hamming and subset 0/1 loss.

Generation of Candidates

Desired Properties

For the following study, we rely on a method that allows us to generate a large number of rules for a given training dataset in a short amount of time.[2] The rules should ideally be unbiased, i.e., they should not be biased in favor of a certain heuristic, and they should be diverse, i.e., general rules should be included as well as specific rules. Given that these requirements are met, we consider the generated rules to be representative samples from the space of all possible rules, which is too large to be explored exhaustively. We use the generated candidate rules as a starting point for building different models. They consist of a subset of rules selected with respect to a specific heuristic (cf. Section "Candidate Selection") and filtered according to a threshold (cf. Section "Threshold Selection). Whereas the first step yields a theory with great coverage, the threshold selection aims at improving its consistency.

2: The source code is available online at https://github.com/mrapp-ke/RuleGeneration

Extracting Rules from Random Forests

Following the principles of the binary relevance transformation method (cf. Section 3.6), we train multiple random forests (Breiman, 2001), using varying configuration parameters, for each available label and extract rules from their decision trees.[3] As illustrated in Algorithm 4, we repeat the process until a predefined number of rules T has been generated. Each random forest consists of a predefined number of decision trees (we restrict the number of trees to 10). To ensure that we can generate diverse rules later, we vary the configuration parameter $depth \in [0, 8]$ that specifies the maximum depth of trees (unrestricted, if $depth = 0$) (cf. Algorithm 4, TRAIN_FOREST). For the construction of individual trees,

3: We use the random forest implementation that is provided by the WEKA (Hall et al., 2009) project. It is available at https://www.cs.waikato.ac.nz/ml/weka

Algorithm 4: Iterative generation of rules from random forests

input : Minimum number of rules to be generated T
output: Rule set F

$F = \emptyset$
while $|F| < T$ **do**
  foreach $\lambda_k \in \mathcal{L}$ **and** $depth \in [0, 8]$ **do**
    $rf =$ TRAIN_FOREST $(\lambda_k, depth)$
    $F = F \cup$ EXTRACT_RULES (rf)
return rule set F

the random forest method samples from the available training examples with replacement. In addition, each time a new node should be added to a decision tree, only a random selection of attributes is considered. This guarantees a diverse set of trees to be built. To extract rules from a random forest (cf. Algorithm 4, EXTRACT_RULES), we traverse all paths from the root node to a leaf in each decision tree. We ignore paths that lead to a leaf where the majority value Λ_k is predicted. Consequently, all rules that are generated with respect to a particular label have the same head $\hat{y}_k = 1$ or $\hat{y}_k = 0$, if $\Lambda_k = 0$ or $\Lambda_k = 1$, respectively. A rule's body consists of a conjunction of all conditions encountered on the path from the root node to the corresponding leaf.

Candidate Selection

Separate-and-Conquer Strategy

We use a separate-and-conquer strategy for selecting a subset of the available candidate rules. New rules are added to a model until the coverage stopping criterion in (5.1) is met. Because all candidate rules provide the same prediction for a particular label, the resulting model meets the properties of a DNF. Whenever a new rule is added to a model, the examples and labels it predicts for are marked as covered (cf. Section 5.1), and the following rule is chosen according to its predictions for yet uncovered examples and labels.

Heuristics and Tie-breaking

To create different models, we select subsets of the rules generated earlier using varying heuristics $\mathcal{H}$ (cf. Section 5.2) to evaluate potential candidates. If two candidates are assigned the same quality, we prefer the one that covers more examples or contains fewer conditions in its body. Whenever a new rule is added, the overall coverage of the model increases as more positive labels are covered. However, a new rule may introduce incorrect predictions for some labels. As a result, the consistency of the model may decrease.

Threshold Selection

Problems of Unfiltered Models

As described earlier, we use a SeCo strategy to select more rules until all examples and labels for which a simple default rule mispredicts are covered. The default rule predicts the majority value $\Lambda_1, \ldots, \Lambda_K$ for each label. In this way, the coverage of the resulting model is maximized at the expense of consistency. This is because each rule contributes to the overall coverage but might introduce incorrect predictions in some cases.

Filtering Rules

To trade off between these two aspects, we use a threshold ϕ that aims at diminishing the effects of inconsistent rules. It is compared to the performance scores that result from applying the heuristic $\mathcal{H}$ to each rule previously added to a model. For assessing the quality of a rule, the rule's predictions on the entire training data are taken into account. This is different from the candidate selection, where examples that are already covered by previously selected rules are not considered. Because the candidate selection aims at selecting non-redundant rules, which cover the feature space as uniformly as possible, it considers rules in the context of their predecessors. In contrast, the threshold ϕ takes effect at prediction time, when no order is imposed on the rules. Only rules whose quality exceeds the threshold contribute to the prediction.

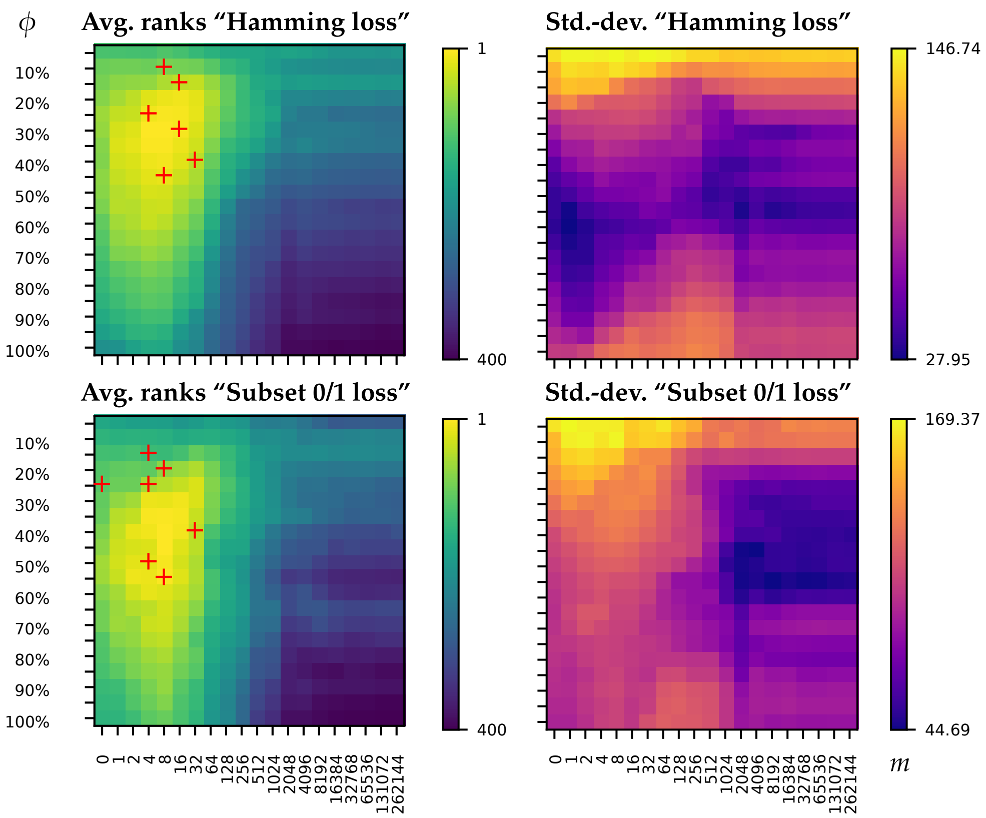


Figure 5.1: Ranks and standard deviation of average ranks over several datasets with respect to Hamming and subset 0/1 loss using different parameters m (horizontal axis) and ϕ (vertical axis). The best parameter settings for different datasets are specified by red + signs.

Experimental Evaluation

Experimental Setup

We have conducted a large number of experiments on different datasets that emphasize the need to use varying heuristics for candidate selection and filtering to obtain models tailored to specific multi-label measures. We further compare our method to different baselines to demonstrate the benefits of flexibly adjusting a learner to different measures rather than employing a general-purpose learner. We applied the previously described method to eight different data sets, namely "Birds", "CAL500", "Emotions", "Enron", "Flags", "Genbase", "Medical", "Scene" and "Yeast" (cf. Section 3.2). We set the minimum number of generated rules to $T = 300,000$. For candidate selection, we used different instantiations of the M-estimate in (5.12). Its parameter was set to $m = 0, 2^1, 2^2, \dots, 2^{19}$. For each of these variants, we applied varying thresholds ϕ. They have been chosen such that they are satisfied by at least 100%, 95%, . . . , 5% of the selected rules. All experimental results have been obtained via 10-fold cross validation. In addition to the M-estimate, we also used the F-measure heuristic in (5.11) with varying β parameters. As the conclusions that can be drawn from these experiments are very similar to those for the M-estimate, we mostly focus on the latter. Among the performance measures that we report are micro averaged precision and recall. Moreover, we assess the quality of models in terms of the micro averaged F1 score, as well as the Hamming and subset 0/1 loss (cf. Section 3.3). For a broad analysis, we trained $20^2 = 400$ models per dataset using the same candidate rules but selecting and filtering them differently using varying combinations of the parameters m and ϕ. We visualize the predictive performance and characteristics of the resulting models as two-dimensional matrices of scores (cf., e.g, Figure 5.1). One dimension corresponds to the m parameter. The other one refers to the threshold ϕ. Some of the used datasets ("CAL500", "Flags" and "Yeast") contain very frequent labels that are relevant to most examples. This is rather atypical in MLC and causes the unintuitive effect that the removal of individual rules via the parameter ϕ results in a model with greater recall and lower precision. To compare different parameter settings across multiple datasets, we worked around this effect by inverting the affected labels.

Predictive Performance

Figure 5.1 and Figure 5.2 depict the average ranks of the tested configurations according to different multi-label evaluation measures. For each dataset, we determined the rank of all 400 parameter settings. Afterward, we averaged them over all datasets. The depicted standard deviations show that the optimal parameter settings for a specific measure may vary depending on the dataset. However, there is always an area in the parameter space where a good setting can be found with high certainty for each measure. It can clearly be seen that precision and recall are competing measures. The former is maximized by choosing small values for m and filtering extensively. The latter benefits from large values for m and no filtering. Interestingly, setting $m = 0$, i.e., selecting candidates according to the precision heuristic, does not result in models with the highest overall precision. This is in accordance with Figure 5.3, where the models with the highest F1 score do not result from using the F-measure heuristic with $\beta = 1$ for candidate selection. Instead, optimizing the F1 score requires small values for m to emphasize the consistency of rules while enforcing a certain coverage simultaneously. The same applies to the Hamming and subset 0/1 loss, albeit both of these evaluation measures demand to put even more weight on consistency and filtering more extensively than F1.

Model Characteristics

Besides their predictive performance, we are also interested in the characteristics of the models. Figure 5.4 shows how the number of rules in a model and the average number of conditions are affected by varying parameter settings. The number of rules independently declines when using greater values for the parameter m or smaller values for ϕ, resulting in less complex models that humans can comprehend more easily. The average number of conditions is mostly affected by the parameter m. Table 5.2 provides an example of how different parameters affect the model characteristics. It shows rules that predict the same label but have been selected by two fundamentally different approaches. The first approach ($m = 16, \phi = 0.3$) reaches high scores according to the F1-measure, Hamming loss, and subset 0/1 loss, whereas the second one ($m = 262144, \phi = 1.0$) results in high recall.

Baseline Comparison

Although the goal of the presented study was not to develop a method that generally outperforms existing rule learners, we want to ensure that we achieve competitive results. For this reason, we compared our method to JRip, WEKA's implementation of RIPPER (Cohen, 1995), using the binary relevance method. RIPPER uses incremental reduced error pruning (IREP) and post-optimizes the induced rule set by default. Although our approach could use such optimizations, this is out of the scope of our experimental study. For a fair comparison, we also report the results of JRip without using IREP and with post-optimization

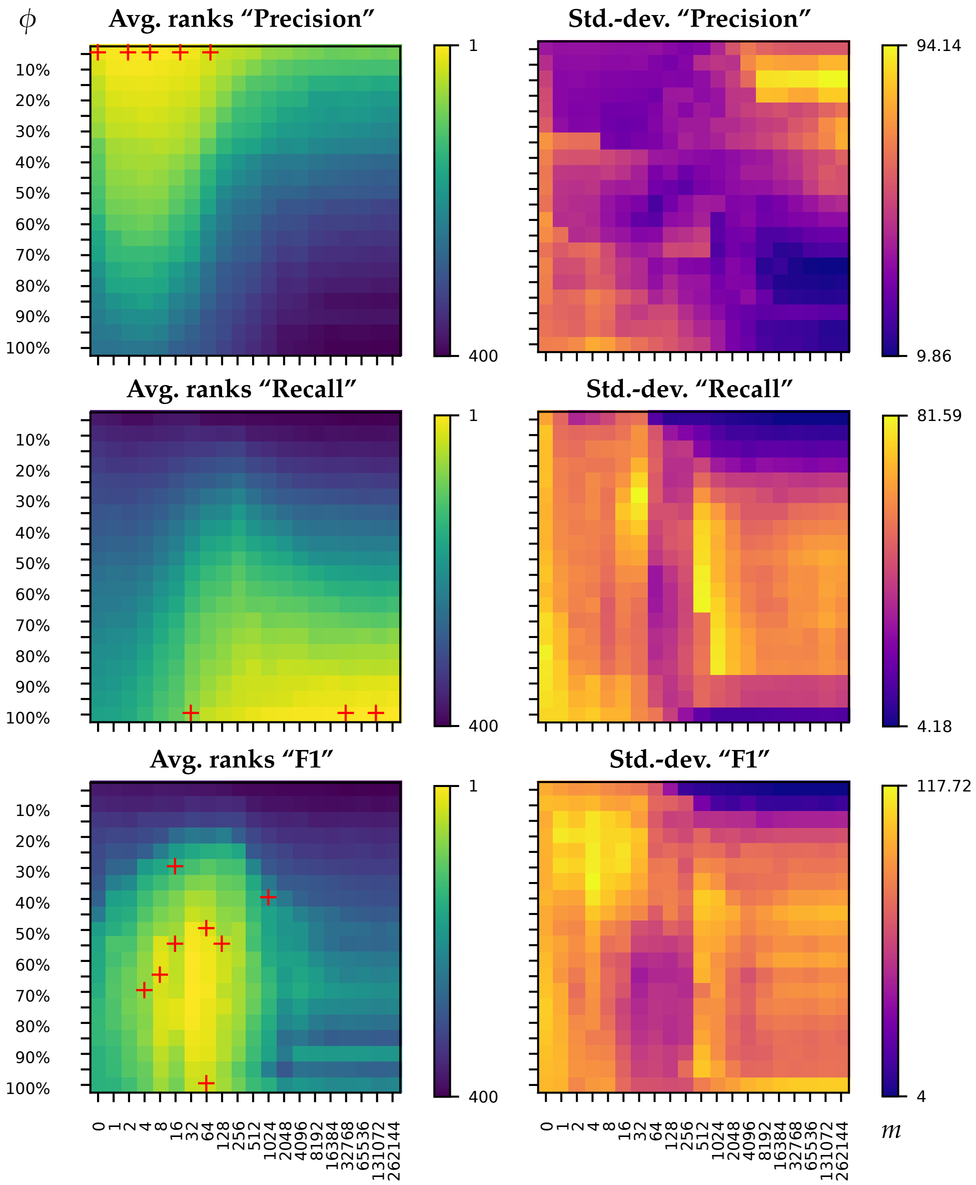


Figure 5.2: Ranks and standard deviation of average ranks over several datasets with respect to micro averaged precision, recall and F1 measure. The best parameter settings for different datasets are specified by red + signs.

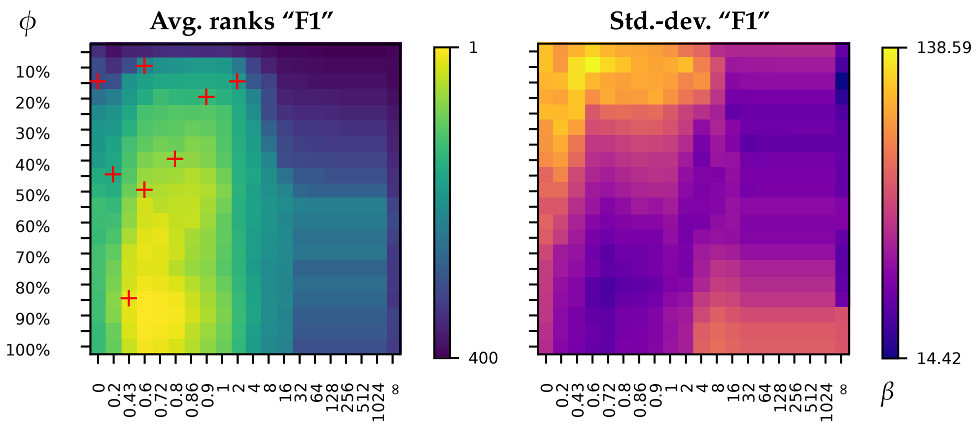


Figure 5.3: Ranks and standard deviation of average ranks over several datasets with respect to micro averaged F1 measure, when using the F-measure with varying β parameters (horizontal axis) instead of the M-estimate for candidate selection. The best parameter settings for different datasets are specified by red + signs.

Table 5.1: Predictive performance of RIPPER using IREP and post-optimization (R_3), without using post-optimization (R_2) and using neither IREP nor post-optimization (R_1) compared to different approaches for candidate selction that try to optimize micro averaged F1 (M_{F1}), Hamming loss (M_H) or subset 0/1 loss ($M_{0/1}$).

Dataset	Micro averaged F1				Hamming loss				Subset 0/1 loss			
	R_1	R_2	R_3	M_{F1}	R_1	R_2	R_3	M_H	R_1	R_2	R_3	$M_{0/1}$
Birds	43.65	41.12	**46.01**	45.33	5.61	5.52	**4.83**	4.90	55.80	54.43	**48.52**	51.15
CAL500	33.63	33.18	33.76	**40.10**	17.86	16.34	14.61	**13.98**	100.00	100.00	100.00	100.00
Emotions	56.96	58.68	60.97	**65.20**	24.88	24.62	22.79	**22.35**	81.96	79.60	**76.40**	77.58
Enron	50.57	53.05	**55.33**	51.07	5.65	5.30	**5.07**	5.46	93.83	92.01	**90.84**	92.19
Flags	71.81	72.96	**74.85**	72.83	26.98	25.92	**24.80**	26.61	84.53	82.95	**79.00**	90.18
Genbase	98.83	98.68	98.68	**99.14**	0.11	0.12	0.12	**0.08**	2.72	3.17	3.17	**2.11**
Medical	81.40	83.67	**84.81**	81.67	0.99	0.90	**0.85**	1.02	33.26	30.09	**27.84**	33.57
Scene	63.97	63.25	64.55	**67.44**	12.13	12.75	11.97	**11.07**	53.39	55.46	53.76	**50.27**
Yeast	58.65	60.41	61.19	**64.25**	21.50	21.71	21.23	**20.76**	91.27	92.14	90.82	**88.25**
Avg. rank	3.44	3.00	1.67	1.78	3.44	2.89	1.67	1.89	2.89	2.67	1.56	2.11

turned off.[4] We tested three different configurations of our approach. The parameters m and ϕ used by these approaches have been determined on a validation set using nested 5-fold cross validation on the training data. For the approach M_{F1}, the parameters have been chosen such that the F1-measure is maximized. M_H and $M_{0/1}$ were tuned with respect to the Hamming and subset 0/1 loss, respectively. According to Table 5.1, our method can achieve reasonable predictive performance. With regard to the measure they try to optimize, our approaches generally rank before JRip with optimizations turned off (R_1), which is the competitor that is conceptually closest to our method. Although IREP positively affects the predictive performance, our approaches tend to outperform JRip with IREP enabled but without any post-optimization (R_2). Despite the absence of advanced pruning and post-processing techniques, our approaches can even surpass the fully-fledged variant of JRip (R_1) on some data sets. We consider these results as a clear indication that it is indispensable to flexibly adapt the heuristic used by a rule learner if one aims at deliberately optimizing a specific multi-label performance measure.

4: We do not consider the random forests from which we generate rules as relevant baselines. This is because random forests use voting to make a prediction, which is fundamentally different from rule learners that model a DNF. Also, we train random forests consisting of a vast number of trees with varying depths to generate diverse rules. In our experience, these random forests perform poorly compared to commonly used configurations.

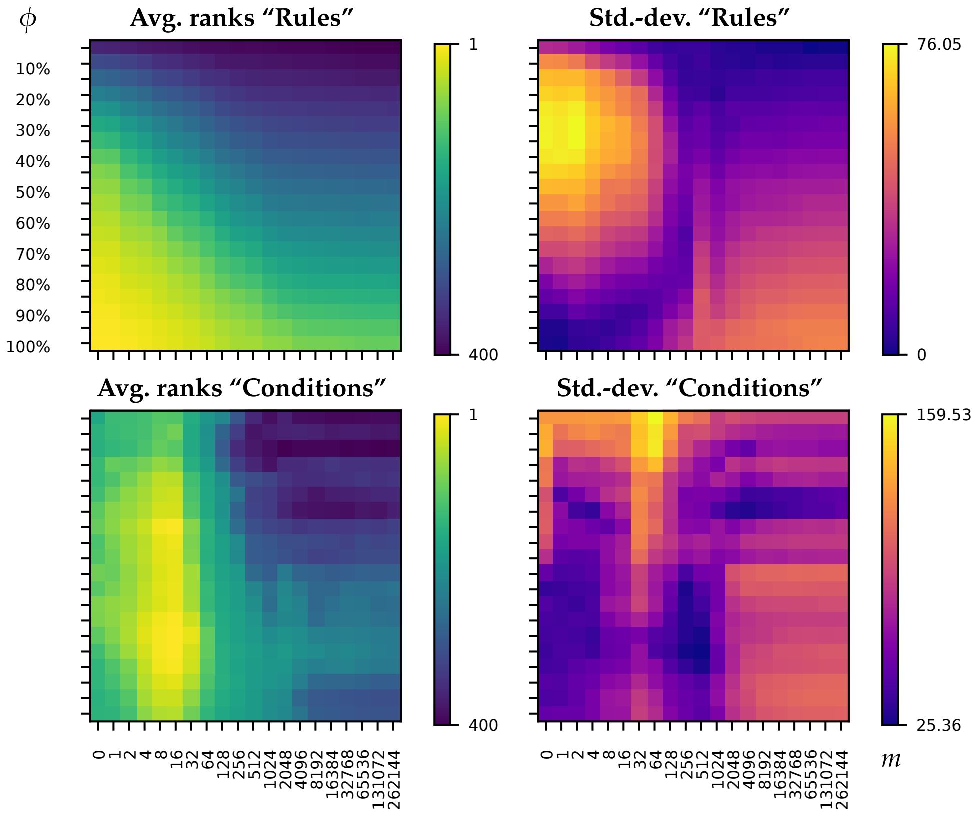


Figure 5.4: Ranks and standard deviation of average ranks over several datasets with respect to the number of rules and conditions. A smaller rank means more rules or conditions.

Table 5.2: Exemplary rule sets predicting the label *786.2:Cough* of the dataset "Medical", which consists of textual radiology reports that were categorized into diseases.

$m = 16, \phi = 0.3$	Mi. Precision = 74.07%, Mi. Recall = 78.26%
Cough $\leftarrow$ *cough* $\wedge$ *aldrich* $\wedge$ *opacity* $\wedge \neg$ *tachypnea* $\wedge \neg$ *streaky* $\wedge \neg$ *side* $\wedge \neg$ *distal* $\wedge \neg$ *diaphragm*	
Cough $\leftarrow$ *cough* $\wedge \neg$ *x-rays* $\wedge \neg$ *vomiting* $\wedge \neg$ *proximity* $\wedge \neg$ *hematuria* $\wedge \neg$ *focal*	
Cough $\leftarrow$ *cough* $\wedge \neg$ *group* $\wedge \neg$ *edema* $\wedge \neg$ *fever*	
Cough $\leftarrow$ *cough* $\wedge \neg$ *lobe* $\wedge \neg$ *breathing*	
Cough $\leftarrow$ *coughing*	
$m = 262144, \phi = 1.0$	Mi. Precision = 65.61%, Mi. Recall = 89.57%
Cough $\leftarrow$ *cough* $\wedge \neg$ *ureteral* $\wedge \neg$ *stones* $\wedge \neg$ *contrast*	
Cough $\leftarrow$ *coughing*	
Cough $\leftarrow$ *code*	
Cough $\leftarrow$ *substance*	

5.4 Search for Multi-label Heads

Motivation

In Section 5.3, we demonstrated that a separate-and-conquer rule learning algorithm can be tailored to varying evaluation measures by trading off the consistency and coverage of rules differently. However, the study presented in the previous section is restricted to single-label rules, which are not allowed to provide predictions for more than a single label. If one is interested in optimizing a non-decomposable evaluation measure, such as the F1-measure or the subset 0/1 loss, the use of single-label rules most likely limits the effectiveness of such an approach, as there is a consensus that non-decomposable measures require taking interactions between labels into account (cf. Section 3.5). In the following, we investigate a method by Klein, Rapp, and Loza Mencía (2019) that aims to overcome this limitation by constructing partial multi-label heads if appropriate. Including predictions for multiple labels in the head of a rule enables to model local dependencies, such as co-occurrences and other types of interdependencies, between the respective labels (cf. Section 4.1).

Klein, Rapp, and Loza Mencía (2019): 'Efficient Discovery of Expressive Multi-label Rules using Relaxed Pruning'

Computational Challenges

As the number of label combinations for which a rule may predict increases exponentially with the number of available labels, the induction of multi-label rules is computationally challenging. The methodology discussed in this section is built upon theoretical findings by Rapp, Loza Mencía, and Fürnkranz (2018) , who have shown that rules with multiple labels in their heads can be constructed efficiently by exploiting certain properties of multi-label rule learning heuristics and pruning the search for multi-label heads accordingly. However, preliminary experiments revealed that multi-label heads are unlikely to be learned by such an approach due to its restrictiveness. Therefore, in (Klein, Rapp, and Loza Mencía, 2019), we propose to relax the pruning by introducing a bias towards larger multi-label heads to circumvent this practical issue. They further show empirically that their approach indeed results in more multi-label heads being found and does not come with a significant drawback in terms of training efficiency.

Rapp, Loza Mencía, and Fürnkranz (2018): 'Exploiting Anti-monotonicity of Multi-label Evaluation Measures for Inducing Multi-label Rules'

Label Space Pruning

Structure of the Algorithm

The construction of deterministic multi-label heads is particularly challenging, as the number of label combinations that can potentially be included in a head increases exponentially with the number of available labels. In (Rapp, Loza Mencía, and Fürnkranz, 2018) we propose to exploit certain properties of commonly used multi-label rule learning heuristics to mitigate the computational complexity of searching for multi-label heads. More specifically, they rely on *anti-monotonicity* and *decomposability* to prune the search space. A multi-label rule learning algorithm based on the separate-and-conquer principle is used in their work. As discussed in Section 5.1, it learns new rules iteratively by focusing on examples and labels for which no predictions are available yet. If a significant fraction of an example's labels has already been dealt with, the respective example is ignored entirely in subsequent iterations. The resulting rules are included in a multi-label decision list. The rules are applied in the order of their induction to obtain predictions for yet unseen examples (cf. Algorithm 3). If a rule covers an example, its predictions take effect unless a previous rule has already predicted for the respective labels. The algorithm performs a top-down greedy search to learn new rules, starting with an empty body. By adding additional conditions to a rule's body, the rule is successively specialized and covers fewer examples in the process. For each candidate body, a corresponding single- or multi-label head, which models the labels of the covered examples as accurately as possible, must be found.

Search Through the Label Space

When a suitable (multi-label) head should be found for a given body, potential label combinations are evaluated with respect to a heuristic using a breadth-first search. Instead of performing an exhaustive search through the label space, which is infeasible in practice due to its exponential complexity, the search is pruned by leaving out unpromising label combinations, as illustrated in Figure 5.5. Depending on the characteristics of the given heuristic function, the search for multi-label heads may either be pruned by exploiting anti-monotonicity or decomposability. In both cases, the best possible solution is still guaranteed to be found. We focus on the latter in the following because decomposability is a stronger criterion compared to anti-monotonicity. It enables pruning the search space more extensively and comes with linear costs, i.e., the best multi-label head can be inferred from considering each label separately.

Definition 5.4.1 *A multi-label heuristic $\mathcal{H}$ allows for "pruning by decomposability" if the following conditions are met:*

I. *If the head $\hat{\boldsymbol{p}}$ of a multi-label rule $\hat{\boldsymbol{p}} \leftarrow b$ includes a discrete prediction $\hat{p}_k \in \hat{\boldsymbol{p}}$ for which the corresponding single-label head rule $\hat{p}_k \leftarrow b$ does not reach the heuristic quality q_{max}, the multi-label rule cannot reach that quality either (and vice versa).*

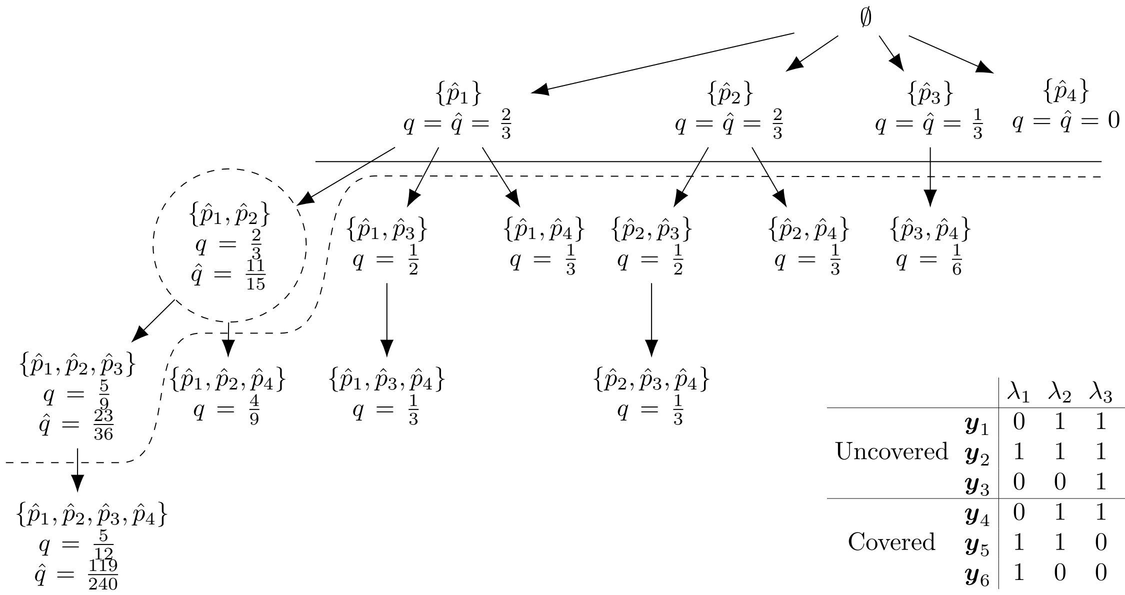
$$\exists k \, (\hat{p}_k \in \hat{\boldsymbol{p}} \wedge \mathcal{H}(\hat{p}_k \leftarrow b) < q_{max}) \Longleftrightarrow \mathcal{H}(\hat{\boldsymbol{p}} \leftarrow b) < q_{max}$$

II. *If all single label head rules $\hat{p}_k \leftarrow b$ which correspond to the predictions of the multi-label head $\hat{\boldsymbol{p}}$ reach the heuristic quality q_{max}, the multi-label rule $\hat{\boldsymbol{p}} \leftarrow b$ reaches said quality as well (and vice versa).*

$$\mathcal{H}(\hat{p}_k \leftarrow b) = q_{max} \, , \forall \hat{p}_k \, (\hat{p}_k \in \hat{\boldsymbol{p}}) \Longleftrightarrow \mathcal{H}(\hat{\boldsymbol{p}} \leftarrow b) = q_{max}$$

Pruning by Decomposability

According to Definition 5.4.1, we can safely prune the search space by restricting the evaluation to all possible single-label heads for a given body. To construct the best possible multi-label head, the highest heuristic quality among all single-label heads is determined, and those that achieve the highest quality are combined, while the others are discarded. In particular, Definition 5.4.1 is less restrictive than the definition of decomposability in (3.26). The latter requires that the predictive quality across several labels can be computed by aggregating the quality estimates for individual labels in a suitable way. The former does not require the ability to compute the overall quality across multiple labels directly. It rather guarantees that the best-rated prediction for multiple labels results from a combination of the predictions that achieve the highest quality when considered in isolation. As we have shown formally in (Rapp, Loza Mencía, and Fürnkranz, 2018), some multi-label heuristics, such as the micro-averaged F1-measure, which are not decomposable in the traditional sense, are still suited for pruning by decomposability.



		λ_1	λ_2	λ_3	λ_4
	$\boldsymbol{y}_1$	0	1	1	0
Uncovered	$\boldsymbol{y}_2$	1	1	1	1
	$\boldsymbol{y}_3$	0	0	1	0
	$\boldsymbol{y}_4$	0	1	1	0
Covered	$\boldsymbol{y}_5$	1	1	0	0
	$\boldsymbol{y}_6$	1	0	0	0

Figure 5.5: Search for the best discrete multi-label head given the labels λ_1, λ_2, λ_4 and λ_4. The examples with ground truth label vectors $\boldsymbol{y}_4$, $\boldsymbol{y}_5$, $\boldsymbol{y}_6$ are assumed to be covered, whereas those that correspond to $\boldsymbol{y}_1$, $\boldsymbol{y}_3$ and $\boldsymbol{y}_4$ are not. Label combinations below the solid line can be pruned by decomposability, whereas the dashed line corresponds to relaxed pruning.

Practical Limitations

Although pruning by decomposability enables to efficiently induce multi-label heads, experiments have revealed that such patterns are unlikely to be learned in practice. This is due to the characteristics of common heuristic functions that focus exclusively on the consistency of a rule's predictions and on its generality regarding the covered examples but do not reward the increase in coverage that results from the inclusion of multiple predictions in the head. Such heuristics tend to prefer single-label predictions over multi-label heads because the predictive quality for different labels usually varies. For example, if two rules with the same body but different single-label heads achieve a quality of 0.89 and 0.88, respectively, predicting both labels usually results in a performance decline compared to the value 0.89 — typically resulting in a value between 0.89 and 0.88. However, opting for the multi-label head would arguably be a good choice. First, the resulting rule would have greater coverage with respect to the available labels. Second, it evaluates to a quality score that is only slightly worse than that of the best single-label rule. In the following, we present a strategy to overcome the bias towards single-label predictions, referred to as *relaxed pruning*. We argue that strict upper bounds in terms of computational complexity can still be guaranteed when relaxing the search for multi-label heads. Moreover, as shown empirically, the training process tends to terminate earlier due to the increased coverage of the induced rules. An experimental evaluation also reveals that the proposed approach discovers more label dependencies and that the use of relaxed pruning results in more compact models that reach predictive results comparable to existing approaches.

Table 5.3: Exemplary calculation of lifted heuristic values $\hat{q}$ as the product of a rule's original quality q and the relaxation lift that is obtained from a relaxation lift function ρ.

$\lvert h \rvert$	q	$\rho(\lvert h \rvert)$	$\hat{q}$
1	0.70	1.00	$0.70 \cdot 1.00 = 0.7000$
2	0.67	1.07	$0.67 \cdot 1.07 = 0.7169$
3	0.63	1.12	$0.63 \cdot 1.12 = 0.7056$

Relaxation Lift Functions

Introducing a Bias Towards Multi-label Heads

The pruning strategy described in the previous section completely neglects combinations of labels with similar, but not equal, heuristic quality. As illustrated by the example above, when pruning by decomposability, single-label heads with marginally greater heuristic quality are preferred to multi-label heads that are rated slightly worse. Relaxed pruning aims at tolerating minor declines in terms of a rule's overall quality in favor of greater coverage. We expect more expressive rules to be learned by relaxing the pruning constraints and introducing a bias towards multi-label heads. The main challenge of introducing such a bias revolves around two questions. First, the desired degree of the bias is unclear, i.e., how much of a decline in heuristic quality is tolerable. Second, the ideal number of labels in the head is unknown, particularly if rules may also predict the absence of labels. As both factors highly depend on the dataset at hand, providing any recommendations is difficult. Moreover, the training efficiency potentially suffers from relaxed pruning, as more label combinations are considered.

Lifted Heuristic Values

We introduce a bias towards multi-label heads by multiplying a rule's heuristic quality q with a dynamic weight $\Delta \in \mathbb{R}$, which we refer to as a *relaxation lift*. In order to prefer larger multi-label heads, Δ must increase with the number of labels in the head. The relaxation lift, which we refer to as *lift* in the remainder of this chapter, affects the decline in a rule's heuristic quality that is acceptable in favor of predicting for more labels. To specify a relaxation lift for every number of labels $|h| \in [1, K]$ possibly contained in a head $\hat{\boldsymbol{p}}$, we use *relaxation lift functions* $\rho : \mathbb{R}_+ \longrightarrow \mathbb{R}$ that map a given number of labels to a relaxation lift Δ. Although the function is only applied to natural numbers, explicitly allowing real numbers as the input of the function ρ facilitates its definition. Given the heuristic quality of a rule q and the number of labels $|h|$ for which it predicts, a *lifted heuristic value* $\hat{q}$ can be calculated via a lift function ρ as

$$\hat{q} = q \cdot \rho\left(|h|\right). \tag{5.13}$$

An example of how to calculate lifted heuristic values is given in Table 5.3. Such values are meant to be used as substitutes for the numerical scores that result from a heuristic function $\mathcal{H}$.

KLN Relaxation Lift Function

The proposed framework for relaxed pruning flexibly allows for the utilization of different relaxation lift functions with varying characteristics and effects on the rule induction process. The first lift function that is considered in this work, is referred to as the *KLN relaxation lift function*. It is defined as

$$\rho_{KLN}\left(|h|\right) \coloneqq 1 + k \cdot \ln\left(|h|\right), \tag{5.14}$$

i.e., it calculates the natural logarithm of the number of labels $|h|$ for which a rule predicts, multiplied by a user-configurable parameter $k \geq 0$. Adding an offset of 1 to the calculated lift ensures that $\Delta = 1$ in the case of single-label heads. The extent of the lift increases with greater values for the parameter k. Due to the natural logarithm, the function becomes less steep as the number of labels increases. This is necessary to prevent a bias towards multi-label heads with a very large number of labels.

Peak Relaxation Lift Function

The second lift function used in this work is referred to as the *Peak lift function*. Compared to the KLN lift function, it puts more emphasis on preventing too many labels from being included in the head of a rule. With an increasing number of labels $1, \dots, k$, where k is a configurable parameter, referred to as the *peak*, the lift first becomes greater until it decreases again. This enables to introduce a bias towards heads that predict for a specific number of labels, as their quality is lifted more than others. Given the peak k, the desired lift at the peak Δ_{max}, the total number of labels K, and a parameter c that affects the function's curvature ($c = 1$ results in a linear slope), the peak relaxation lift function calculates as

$$\rho_{Peak}(|h|) := \begin{cases} f(|h|, k, 1) & \text{if } |h| \leq k \\ f(|h|, k, K) & \text{otherwise,} \end{cases} \tag{5.15}$$

where the auxiliary function f is defined as

$$f(x, a, b) := 1 + \left(\frac{x-b}{a-b}\right)^{\frac{1}{c}} \cdot (\Delta_{max} - 1)$$

As a potential advantage of the peak lift function, we consider its ability to control the impact on training efficiency introduced by relaxed pruning. Using smaller values for the peak k is expected to result in more extensive pruning and, therefore, less computational overhead. Compared to the KLN lift function, it is also less susceptible to including too many labels in the heads of rules, which may result in a deterioration of a model's predictive accuracy. In addition, the peak lift function can be adapted more flexibly via the parameters k, Δ_{max} and c. On the downside, as these parameters significantly affect the learned model, this flexibility also comes with a certain risk of misconfiguration. A visualization of the peak lift function and the KLN lift function is given in Figure 5.6.

Relaxed Pruning

Modifications of the Search Algorithm

When assessing the quality of potential rules in terms of a lifted heuristic value $\hat{q}$ rather than the value q that results from a traditional heuristic function, it is necessary to adjust the search for multi-label heads. In the following, we show that strictly pruning according to Definition 5.4.1, as suggested in (Rapp, Loza Mencía, and Fürnkranz, 2018), does not yield the best head in terms of $\hat{q}$. Hence, we propose an alternative pruning strategy and discuss the necessary changes in detail. The discussion is accompanied by an example that illustrates the proposed approach.

Suboptimal Pruning

When pruning by decomposability, the best (multi-label) head is obtained by combining all single-label heads with the best heuristic value (cf. Figure 5.5). By giving a simple counter-example, we show that this is impossible when searching for the head with the highest lifted heuristic value. Consider two heads $\{\hat{p}_1\}$ and $\{\hat{p}_2\}$, with heuristic values of 0.8 and 0.75, respectively. As we do not lift the quality of single-label heads, the lifted and unlifted heuristic values are equal in this case. When using label-wise averaging, the unlifted heuristic value of the multi-label head $\{\hat{p}_1, \hat{p}_2\}$ calculates as 0.8 + 0.75 / 2 = 0.775. Assuming that greater values are better, the single-label head $\{\hat{p}_1\}$ would therefore be preferred over the combination of both heads. However, assuming that the lift for two labels is 1.1, the lifted heuristic value of the latter evaluates to

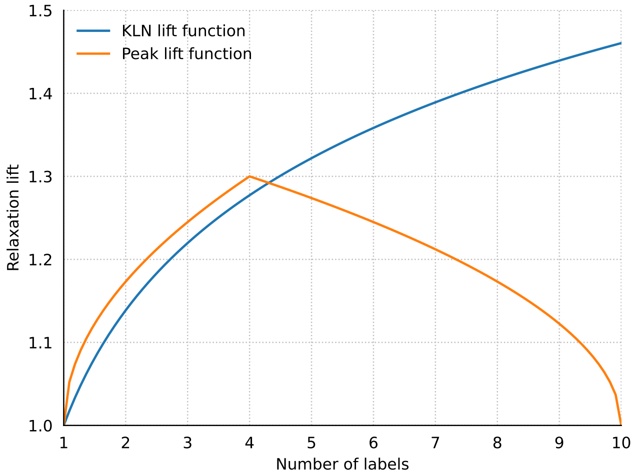


Figure 5.6: Visualization of the KLN relaxation lift function (with $k = 0.2$) and the Lift relaxation function with ($k = 4$, $c = 2$ and $\Delta_{max} = 1.3$) for $1, \ldots, 10$ labels.

$0.775 * 1.1 = 0.8525$. Consequently, the combination of both heads is rated better than both of the corresponding single-label heads when considering their lifted heuristic values and, therefore, should be preferred. As a result, we conclude that the search space pruning we suggested in (Rapp, Loza Mencía, and Fürnkranz, 2018) is not suited to find the best head in terms of its lifted quality.

Relaxed Pruning for Decoposable Heuristics

We adjust the original pruning algorithm, as previously described in section "Label Space Pruning", based on two observations. First, the best lifted head of size k results from applying the lift to the head with the highest unlifted heuristic value of size k. As all heads that predict for k labels are subject to the same lift, a head of size k with a worse unlifted heuristic value cannot achieve a better lifted heuristic value. Thus, we obtain the best head of a certain size in terms of its lifted heuristic value by finding the best unlifted head. Second, when dealing with a decomposable heuristic function, we can guarantee that the best unlifted head of size k results from combining the k best single-label heads. In particular, this applies to heuristics that are computed according to label-wise averaging, as shown in (5.6). The basic structure of the relaxed pruning algorithm is shown in Algorithm 5. Similar to the original pruning algorithm, we need to evaluate all single-label heads for a given rule body (cf. solid line in Figure 5.5). In accordance with our observations, we sort the individual single-label heads in decreasing order by their respective heuristic values (Algorithm 5, line 2). Afterward, we process the available single-label heads in sorted order, starting with the best-rated one. At each step, we consider adding the next single-label prediction to the existing head. This requires first computing the unlifted heuristic value of the resulting multi-label head. When using a decomposable heuristic function, it is not necessary to re-evaluate each considered multi-label head. Instead, its unlifted quality can be calculated as the average of the heuristic values that correspond to the single-label predictions it consists of (cf. Algorithm 5, line 6). Afterward the lift for a head of the current size is retrieved from the lift function (cf. Algorithm 5, line 7). Multiplying the lift with the unlifted heuristic value results in the lifted quality of the current multi-label head (cf. Algorithm 5, line 8). During the entire process, we keep track of the head with the best lifted heuristic value $\hat{q}_{best}$. Instead of generating all possible multi-label heads, we determine an upper bound $\hat{q}_{upper}$ of the lifted heuristic value that can be reached

Algorithm 5: Search for multi-label heads using relaxed pruning

input : Label-wise aggregated confusion matrices $C_1, \ldots, C_K$ with $C_k = \sum_n (C_{nk} w_{nk})$, label-wise averaged heuristic $\mathcal{H}$, relaxation lift function ρ

output: Single- or multi-label head $\hat{\boldsymbol{p}}$, lifted heuristic value $\hat{q}$

1 compute unlifted heuristic values $q_1, \ldots, q_K$ with $q_k = \mathcal{H}(C_k)$
2 find permutation $\tau : \mathbb{N}^+ \rightarrow \mathbb{N}^+$ such that $q_{\tau(i)} \geq q_{\tau(i+1)}, \forall i \in [1, K)$
3 initialize best head $\hat{\boldsymbol{p}} = \left\{\hat{p}_{\tau(1)}\right\}$
4 initialize best lifted heuristic value $\hat{q}_{best} = q_{\tau(1)}$
5 **for** $k = 2$ **to** K **do**
6     $q = \text{avg}\left(q_{\tau(1)}, \ldots, q_{\tau(k)}\right)$
7     $\Delta = \rho(k)$
8     $\hat{q} = q \cdot \Delta$
9   **if** $\hat{q} \geq \hat{q}_{best}$ **then**
10         $\hat{q}_{best} = \hat{q}$
11   add prediction $\hat{p}_{\tau(k)}$ to $\hat{\boldsymbol{p}}$
12     $\Delta_{max} = \max_{k<i\leq K} \rho(i)$
13     $\hat{q}_{upper} = q \cdot \Delta_{max}$
14   **if** $\hat{q}_{upper} < \hat{q}_{best}$ **then**
15   **break**
16 **return** best head $\hat{\boldsymbol{p}}$, lifted heuristic value $\hat{q}_{best}$

by adding more predictions to the current head. The upper bound is obtained by multiplying the unlifted heuristic value q_k of the current head with size k by the highest remaining lift Δ_{max} (cf. Algorithm 5, line 12), i.e., it calculates as $\hat{q}_{upper} = q_k \cdot \Delta_{max}$ (cf. Algorithm 5, line 13). If $\hat{q}_{upper} < \hat{q}_{best}$, we can abort the search procedure, as the best quality encountered so far cannot be reached by larger heads (cf. Algorithm 5, line 14). This is due to the fact that the unlifted heuristic value of the current head cannot be improved by adding more predictions, as the search starts with the best-rated single-label head. When using a decomposable heuristic, pruning the search for multi-label heads in this way still guarantees the best-rated head in terms of its lifted heuristic value to be found.

Example

In addition to Algorithm 5, we illustrate the relaxed pruning algorithm by providing an example. It is based on the rule heads that are depicted in Figure 5.5 and assumes the KLN lift function with $k = 0.14$ to be used. Given heads that predict for two, three or four labels, this particular lift function evaluates to $\rho(2) = 1.1$, $\rho(3) = 1.15$, and $\rho(4) = 1.19$. Once the unlifted heristic values of all possible single-label rules are computed, the multi-label heads on the outer left path of Figure 5.5 are constructed. For the head $\{\hat{p}_1\}$, the lifted heuristic value and the maximum lifted value calculate as $\hat{q} = 2 / 3 = \hat{q}_{best}$ and $\hat{q}_{upper} = 2 / 3 \cdot 1.19 = 0.793$ (rounded to three decimal places). Because $\hat{q}_{upper} \geq \hat{q}_{best}$, we cannot stop the search at this point. For the head $\{\hat{p}_1, \hat{p}_2\}$, the upper bound $\hat{q}_{upper}$ stays the same, but the lifted heuristic value evaluates to $\hat{q} = 2 / 3 \cdot 1.1 = 0.733 = \hat{q}_{best}$ and exceeds the best quality seen so far. As $\hat{q}_{upper} \geq \hat{q}_{best}$, we must continue the search by considering the head $\{\hat{p}_1, \hat{p}_2, \hat{p}_3\}$, for which we obtain $\hat{q} = 5 / 9 \cdot 1.15 = 0.639$ and $\hat{q}_{upper} = 5 / 9 \cdot 1.19 = 0.661$. As the pruning criterion $\hat{q}_{upper} < \hat{q}_{best}$ is satisfied, we terminate the search and return the best head found so far. The dashed line in Figure 5.5 indicates which heads need to be examined when using relaxing pruning. By chance, the best lifted and unlifted heads are the same in this example.

Pruning for Non-decomposable Heuristics

In (Rapp, Loza Mencía, and Fürnkranz, 2018) we show that many multi-label heuristics, even some of those not computed via label-wise averaging, meet the requirements in Definition 5.4.1 and therefore are suited for pruning by decomposability. In contrast, relaxed pruning does not guarantee to find the best multi-label head in terms of lifted heuristic values unless the heuristic function is decomposable according to (3.26). When dealing with a non-decomposable heuristic, combining the best k single-label heads according to Algorithm 5 does not necessarily result in the best lifted head because the individual labels are not weighed equally. For example, this applies to the micro-averaged F-measure according to (5.7) and (5.11). Despite this limitation, we still utilize non-decomposable heuristics in the experimental study below and acknowledge that the use of relaxed pruning can merely be considered as an approximation in this case. According to our experiments, even though it does not guarantee the best head for a given body to be found, this approximation seems to work well in practice — most likely because we relax the search for optimal heuristic values anyway.

Computational Overhead

Because relaxed pruning computes the unlifted and lifted heuristic values of multi-label heads based on the heuristic values of the corresponding single-label heads, it does not require any additional computations of quality estimates compared to the original pruning algorithm as described in section "Label Space Pruning". Nevertheless, in the worst case, it requires constructing $K - 1$ additional multi-label heads (cf. outer left path in Figure 5.5). However, as the quality of these heads can be computed based on the confusion matrices of the corresponding single-label heads, these additional steps are computationally cheap. Moreover, our experiments reveal that the computational overhead introduced by relaxing the search for multi-label heads is often negligible, as it tends to result in simpler models with fewer rules that are learned more quickly.

Retaining Previously Constructed Heads

Rules are specialized during the rule refinement process by adding additional conditions to their bodies. According to preliminary experiments, searching for a new (multi-label) head whenever a rule has been modified, as we suggested in (Rapp, Loza Mencía, and Fürnkranz, 2018), often results in situations where previously constructed heads are discarded in favor single-label heads with lower coverage but a higher (lifted) heuristic value. Keeping the original head and modifying the body often results in a better rule in such situations. Based on this finding, we decided to retain the original head instead of searching for a new one each time the body is modified. As a positive side effect of this modification, the time required for building a model usually decreases because the search for new heads must be conducted less frequently.

Constraints on Rules

In addition to retaining previously constructed heads, we require each rule to provide as least as many correct predictions as incorrect ones, which effectively imposes a lower bound on the quality of the rules. In preliminary experiments, we found this constraint to be helpful to prevent suboptimal predictions from being included in the heads for the sake of increasing its lift. Moreover, we require each prediction provided by a head to result in at least one uncovered label to be predicted correctly. This prevents the inclusion of predictions that do not affect a rule's unlifted heuristic but increase its lift. For example, such a situation might occur if previously induced rules already deal with all occurrences of a particular label.

Experimental Evaluation

Experimental Setup

To demonstrate the effectiveness of our approach, we conducted an empirical study using several benchmark datasets and varying rule learning heuristics to evaluate candidate rules. Besides the predictive performance achieved by different approaches, we also analyze the characteristics of the resulting models, compare the time they require for training, and provide examples of multi-label rules learned by the proposed method. We tested the proposed method using relaxed pruning on seven multi-label datasets, namely "Birds", "CAL500", "Emotions", "Flags", "Medical", "Scene", and "Yeast" (cf. Section 3.2), using predetermined splits into a training and a test set and compared it to the approach in (Rapp, Loza Mencía, and Fürnkranz, 2018) using the same configuration.[5] . To isolate the influence relaxed pruning has on the learned models, we implemented the idea of retaining previously constructed heads and imposing additional constraints on the learned rules not only in the algorithm that makes use of relaxed pruning, but also integrated it into the baseline. Both approaches were able to learn label-dependent rules as proposed by Loza Mencía and Janssen (2016), i.e., conditions that check for the presence or absence of labels that previous rules have assigned may be included in the bodies of rules. In the following, we refer to such conditions as *label conditions*. For evaluating candidate rules during training, we used the F-measure in (5.11) with $\beta = 0.5$, using micro- and label-wise averaging according to (5.7) and (5.6). In addition, we considered the Hamming accuracy heuristic in (5.8), which is invariant to the averaging strategy used. For all of these heuristics and each dataset, we determined the best configuration of the KLN and peak relaxation lift function by conducting a 5-fold cross validation on the training data. If two settings achieved the same performance, we chose the one with a greater lift, as it typically results in a more compact model. After the best parameter setting had been determined, we used it to train a model, which was afterward evaluated on the test set. Unlike in Section 5.3, we did not enforce that the predictions of rules for individual labels are either positive or negative. Instead, we tested different configurations, where the rules were either obliged to focus exclusively on the relevance of labels (denoted as +) or were allowed to predict both positively and negatively (denoted as ±).

5: The source code and datasets are available at `https://github.com/keelm/SeCo-MLC/tree/relaxed-pruning`.

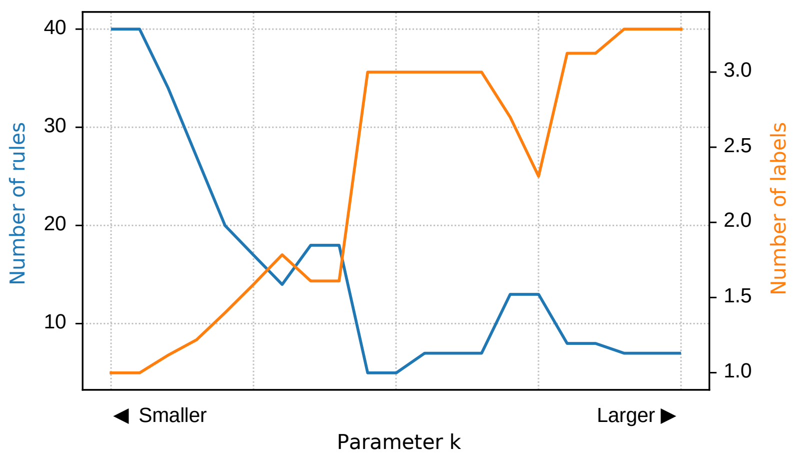


Figure 5.7: Analysis of how the number of rules in a model and the average number of labels in their heads are affected by different configurations of the KLN relaxation lift function on the dataset "Flags". Larger values for the parameter k result in greater relaxation lifts. With increasing lift, the rule heads tend to predict for more labels, resulting in less rules being included in a model overall.

Model Characteristics

Figure 5.7 shows the number of rules learned on the dataset "Flags" and the average number of labels included in their heads, depending on the extent of the lift that results from using the KLN lift function and the label-wise averaged F-measure. As expected, greater lifts tend to result in heads that predict for more labels on average. At the same time, the number of rules in a model usually decreases because rules that predict for several labels rather than a single one have greater coverage, and therefore fewer rules are needed to cover the entire training data. However, if the lift is too high, the training procedure likely results in overly generic rules that predict the majority value for individual labels. Consequently, the learning algorithm fails to model the training data accurately. When using the peak lift function, the same effects can be observed. However, due to the shape of this particular lift function, the maximum number of labels that are included in the rule heads is typically limited. In addition to the sensitivity analysis in Figure 5.7, we provide additional information about the characteristics of models that have been learned on the dataset "Birds", using the optimal configuration of our approach according to parameter tuning, in Table 5.4. It allows for a comparison between the models that result from relaxed pruning or pruning by decomposability, respectively. Regardless of the heuristic employed for the evaluation of candidate rules, the use of relaxation lift functions tends to result in more multi-label heads being learned compared to the baseline. In accordance with Figure 5.7, this usually comes with a decrease in a model's complexity in terms of the number of rules and conditions. Moreover, we observe that models with more and larger multi-label heads typically include fewer label conditions. From this observation, we conclude that label-dependendent rules and rules with multiple predictions in their heads, at least partly, serve the same purpose, namely capturing local label dependencies. As the use of lift functions introduces a reward for the latter but not for the former, an algorithm that uses relaxed pruning favors the construction of multi-label heads over label conditions.

Characteristic	+		±	
Micro-averaged F-measure (with $\beta = 0.5$)				
Rules	140	140	132	92
Conditions	219	213	184	146
Label conditions	7	4	1	2
Multi-label heads	1	5	0	22
Labels per multi-label head	2.0	2.0	–	2.59
Label-wise averaged F-measure (with $\beta = 0.5$)				
Rules	140	129	132	113
Conditions	220	199	184	175
Label conditions	7	3	1	1
Multi-label heads	1	8	0	14
Labels per multi-label head	2.0	2.0	–	2.57
Hamming accuracy				
Rules	162	136	58	23
Conditions	254	204	58	29
Label conditions	3	0	1	0
Multi-label heads	1	30	0	12
Labels per multi-label head	2.0	2.1	–	17.0

Table 5.4: Characteristics of models that have been learned on the dataset "Birds". For each heuristic, we trained models where each rule is restricted to predict the relevance of labels (+) or is allowed to predict both positively and negatively (±). For each of these variants, the left and right column shows the values that result from pruning by decomposability and relaxed pruning, respectively.

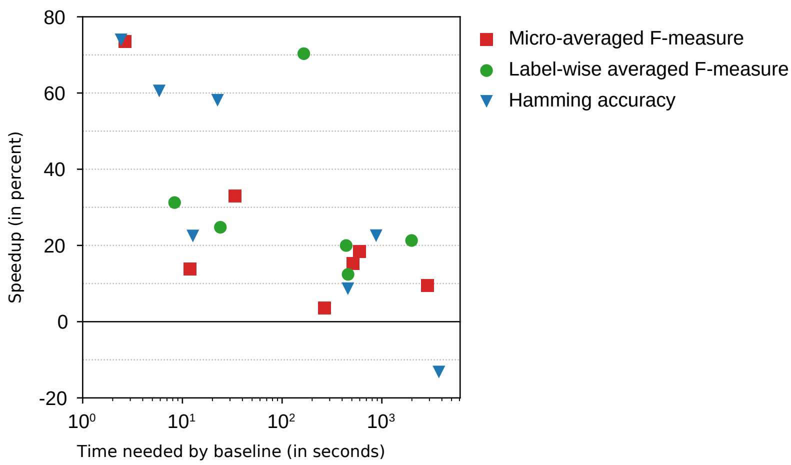


Figure 5.8: Comparison of the time needed for building models with respect to different heuristics when using relaxed pruning or pruning by decoposability. The training time of the latter (in seconds) is show on the horizontal axis, whereas the vertical axis corresponds to the speedup or slowdown (in percent) that results from relaxed pruning.

Computational Costs

In Figure 5.8, we compare the training time that results from the use of relaxed pruning to the time needed by the baseline, which relies on pruning by decomposability. The horizontal axis denotes the time required by the baseline for training when using different heuristics. The vertical axis denotes the relative speedup (or slowdown) that results from using relaxed pruning. Even though more candidates must potentially be evaluated when relaxing the search for multi-label heads, the proposed method is faster in most cases. Typically, it achieves a speedup between 10 and 25%. As we isolate the effects of relaxed pruning from other modifications of the learning algorithm, the speedup most likely results from fewer rules being learned due to their increased coverage. In addition, fewer candidate rules must be taken into account because the average number of conditions often decreases when using relaxed pruning.

Predictive Performance

In the following, we discuss the differences in predictive performance that result from the use of relaxed pruning compared to pruning by decomposability. We conclude from the number of wins, losses, and ties listed in Table 5.5 that relaxed pruning can achieve competitive results, despite learning more compact models. If the rules are restricted to positive predictions, we observe a decline in performance when using label-wise averaged F-measure or Hamming accuracy as the rule learning heuristic. If the rules are allowed to predict both positively and negatively the same heuristics — especially Hamming accuracy — result in an improvement instead. The wins, losses, and ties are distributed more

Table 5.5: Number of wins / losses / ties of different approaches that make use of relaxed pruning, compared to the predictive performance in terms of the micro-averaged and label-wise F1-measure, Hamming loss and Subset 0/1 loss that is achieved when using pruning by decomposability.

Approach	Micro F1	Label-wise F1	Hamming loss	Subset 0/1 loss
Micro-averaged F-measure (with $\beta = 0.5$)				
+	3/3/1	3/3/1	3/3/1	3/2/2
±	1/3/3	1/3/3	3/1/3	2/1/4
Label-wise averaged F-measure (with $\beta = 0.5$)				
+	1/5/1	3/3/1	1/5/1	2/3/2
±	2/1/4	2/1/4	2/1/4	2/1/4
Hamming accuracy				
+	1/5/1	2/4/1	2/4/1	0/4/3
±	4/2/1	2/4/1	5/1/1	5/0/2

Table 5.6: Exemplary rules that predict for selected labels of the datasets "Yeast", "Flags" and "Birds". Rules that result from pruning by decomposability are shown to the left, whereas rules that have been learned by relaxed pruning are shown to the right. In addition, the number of true positives and false positives that are covered by each rule are shown in parantheses (TP, FP). For brevity, the thresholds of conditions are omitted. Instead, we only show the names of the attributes they correspond to. In case of the dataset "Birds", we use the abbreviations "Red-breasted Nuthatch" (RBN), "Black-headed Grosbeak" (BHG), "MacGillivray's Warbler" (MGW), "Warbling Vireo" (WV), "Stellar's Jay" (SJ) and "audio-ssd" (ssd).

"Yeast" (using label-wise F-measure heuristic, restricted to positive rules)							
Class5	$\leftarrow$	*Att61*	(112, 50)	**Class4, Class5**	$\leftarrow$	*Att61*	(230, 94)
Class4	$\leftarrow$	**Class5**	(118, 44)				
Class3	$\leftarrow$	*Attr50*	(84, 50)	**Class2, Class3**	$\leftarrow$	*Att50*	(174, 111)
Class2	$\leftarrow$	**Class2**	(146, 141)				
"Flags" (using micro-averaged F-measure heuristic, allowing positive and negative rules)							
red	$\leftarrow$	$colours_1 \land area_1 \land bars \land crescent$	(85, 9)	**red, white**	$\leftarrow$	$colours_1$	(166, 38)
red	$\leftarrow$	$bars \land crescent \land colours_2 \land area_1 \land stripes$	(13, 6)	$\neg$**green, red**	$\leftarrow$	$colours_2$	(71, 35)
$\neg$**red**	$\leftarrow$	$area_2 \land circles$	(9, 1)	**green,** $\neg$**red**	$\leftarrow$	$\emptyset$	(2, 0)
red	$\leftarrow$	$sunstars_1$	(4, 0)				
$\neg$**red**	$\leftarrow$	$sunstars_2$	(1, 0)				
red	$\leftarrow$	$\emptyset$	(1, 0)				
"Birds" (using label-wise F-measure heuristic, allowing positive and negative rules)							
$\neg$**RBN**	$\leftarrow$	*ssd59*	(288, 0)	$\neg$**BHG,** $\neg$**WV,** $\neg$**MGW,** $\neg$**SJ,** $\neg$**RBN**	$\leftarrow$	*ssd64*	(1012, 3)
$\neg$**RBN**	$\leftarrow$	*ssd89*	(21, 0)	$\neg$**MGW,** $\neg$**RBN**	$\leftarrow$	*ssd56*	(141, 0)
RBN	$\leftarrow$	*ssd153*	(4, 0)	$\neg$**CN,** $\neg$**RBN**	$\leftarrow$	$ssd7 \land ssd145$	(190, 0)
$\neg$**RBN**	$\leftarrow$	$\emptyset$	(9, 0)	$\neg$**RBN**	$\leftarrow$	*ssd8*	(16, 0)
				RBN	$\leftarrow$	*ssd45*	(4, 0)
				$\neg$**RBN**	$\leftarrow$	$\emptyset$	(1, 0)

evenly among approaches that utilize the micro-averaged F-measure as a rule learning heuristic. This is despite the fact that relaxed pruning cannot guarantee optimal rule heads to be found in this particular setting. Among all considered approaches, the one that employs relaxed pruning for optimizing the Hamming accuracy heuristic and allows rules to predict positively and negatively ranks best for Hamming and Subset 0/1 loss. In terms of micro-averaged and label-wise F1-measure, the best rated models are obtained using relaxed pruning, relying on the micro-averaged F-measure heuristic and restricting the rules to positive predictions. This illustrates that the F1-measure puts greater emphasis on relevant labels than the Hamming loss. In conclusion, relaxing the pruning constraints and deliberately preferring rules with a slightly worse heuristic value in favor of increasing the coverage across multiple labels does not seem to harm the predictive performance of the models and even results in improvements in some cases. As mentioned earlier, we used parameter tuning to determine the best configuration of the KLN or peak relaxation lift function for a particular dataset. An analysis of the chosen configurations reveals that the latter is preferred to the former in most cases.

Exemplary Rules

In Table 5.6, we show exemplary rules that have been induced with and without the use of relaxed pruning. It can be seen that multi-label heads and label conditions are both suited to model label dependencies. Depending on the model, these different representations may even be equivalent in meaning (cf. Table 5.6, first row). Whereas the use of relaxed pruning likely causes fewer label conditions to be included in a model, it often results in more multi-label heads being constructed. This makes a quantitative analysis of the number of label dependencies discovered by different approaches more difficult. Nevertheless, our experimental results suggest that relaxed pruning helps to model label dependencies in the form of multi-label heads. Such heads often provide a more compact representation of the discovered correlations. In contrast to label conditions, rules with multi-label heads provide useful information on their own. They do not require to take the order of the rules into account and must not be interpreted in the context of other rules. Due to these advantages, we argue that multi-label heads are easier to understand in many cases.

5.5 Discussion

In this chapter, we presented a generalization of the separate-and-conquer paradigm, which is used by many traditional rule learning algorithms, to the multi-label classification setting. Because rules that are restricted to deterministic predictions are unlikely to provide accurate predictions for all labels in a dataset, we either rely on single-label rules, which predict for a single label, or partial multi-label rules, which are concerned with a subset of the available labels. The use of rules that focus on a certain label subset demands a strategy to deal with partially covered training examples. For this purpose, we introduced the notion of label weights, which allow us to keep track of regions in the label space that have not been addressed by a rule. In addition, we presented generalizations of commonly used rule learning heuristics to multiple labels, including decomposable and non-decomposable ones. Based on these foundations, we presented empirical results regarding two important aspects of a multi-label SeCo algorithm. On the one hand, we investigated the effects of different rule learning heuristics and pre-pruning strategies on the characteristics and predictive performance. Compared to single-label classification, the choice of the rule learning heuristic appears to be less straightforward in the multi-label setting, as different heuristics are needed depending on which evaluation measure should be optimized. On the other hand, we presented an approach that enables learning partial multi-label rules in an efficient way, despite the exponential number of possible label combinations that may be included in the head of a rule. Besides the aspects adressed in this chapter, several issues remain to be solved to provide a strong out-of-the-box learner for multi-label classification. Research on traditional rule learning approaches suggests that additional measures are needed to deal with the high variance of SeCo-based classifiers. For example, this includes pruning techniques that aim to improve the accuracy of individual rules In addition, stopping criteria or post-processing techniques can play an important role in overfitting avoidance. As all of these aspects should most probably be tailored to a particular multi-label evaluation measure that one seeks to optimize, it is unclear how existing approaches of this kind should be implemented in the multi-label setting. As illustrated by several examples in this chapter, SeCo algorithms are particularly well-suited for tackling MLC problems if one is interested in simple models that can be inspected and analyzed as a whole. In particular, this applies to models that meet the requirements of a DNF. Compared to models, where rules are allowed to provide positive and negative predictions, a DNF is easier to understand, as individual rules can be viewed in isolation, without the need to take the order of the rules into account. For the same reason, we argue that multi-label heads are often preferable to label-dependent rules. Unlike the latter, which must be interpreted in the context of their predecessors, the former allow a more compact representation of local co-occurrences and other types of label dependencies. In use cases where global interpretability is not a major requirement, more complex learning methods are often a better choice as they are likely to outperform the models that result from a SeCo algorithm. Hence, in the following chapter, we investigate ways to tackle MLC problems by means of rule-based ensembles. As will be seen, ensembles of probabilistic rules overcome several drawbacks of the methodology discussed in this chapter. In particular, learning approaches should ideally be able to deliver predictions that perform well for a certain measure of interest. However, due to their reliance on local heuristics, SeCo algorithms cannot optimize a given measure globally. As our experiments revealed, extensive parameter tuning is needed to tailor a SeCo-based learning algorithm to a particular multi-label evaluation measure. Its behavior is not only affected by a rule learning heuristic but is also subject to a large number of additional parameters. For example, this includes the configuration of a lift function, which heavily impacts the performance of a model and is prone to misconfiguration. Moreover, even though the relaxed pruning approach discussed in this chapter enables the efficient construction of multi-label heads, it does not guarantee finding optimal predictions for several labels if the heuristic function is non-decomposable. As the induction of multi-label heads is particularly relevant in cases where a non-decomposable evaluation measure should be optimized, this can be considered a significant limitation.

Learning Gradient Boosted Multi-label Rules

6

In this chapter, we discuss BOOMER — an algorithm for learning gradient boosted multi-label rules. It can be considered to be an instantiation of the framework presented in Chapter 4. Due to its modularity, individual aspects of the algorithm are interchangeable and can be tailored to different applications and datasets by choosing a suitable implementation. In this chapter, we focus on the most essential aspects of the algorithm. Possible optimizations and extensions are left for discussion in Chapter 7 and Chapter 8. Most importantly, we present a generalization of the popular and well-researched gradient boosting framework to multivariate problems that enables minimizing decomposable and non-decomposable loss functions. This generalized framework forms the basis of the BOOMER algorithm, which relies on gradient boosting for the induction of single-, as well as multi-label rules. The algorithmic details of the rule learning method and the formal framework it is built upon, were first published by Rapp, Loza Mencía, Fürnkranz, Nguyen, et al. (2020) . In the following, we recapitulate the methodology that is proposed in this work and revisit the experimental study that has been conducted to investigate its abilities and limitations.

Rapp, Loza Mencía, Fürnkranz, Nguyen, and Hüllermeier (2020): 'Learning Gradient Boosted Multi-label Classification Rules'

The BOOMER software package
The algorithm discussed in this chapter is implemented as part of an open source software package, which is publicly available under the MIT license. Its source code can be found online at `https://github.com/mrapp-ke/boomer`. A high-level description of the implementation is given by Rapp (2021) . In the remainder of this work, we provide information about its usage and configuration. Unless stated otherwise, the given remarks refer to version 0.8.2, which was the latest release at the time this thesis was published.

Rapp (2021): 'BOOMER – An Algorithm for Learning Gradient Boosted Multi-label Classification Rules'

6.1 Boosted Multi-label Rules

Ensembles of Additive Functions

The algorithm, which is discussed in the following, is concerned with learning ensembles of probabilistic rules. Like other ensemble methods (cf. Section 2.3), it focuses on the predictive performance of models at the expense of their simplicity and interpretability. In accordance with (4.1), we denote an ensemble that consists of T additive classification functions $f_t \in \mathcal{F}$, referred to as *ensemble members*, as

$$F = (f_1, \ldots, f_T)\,. \tag{6.1}$$

By $\mathcal{F}$ we denote the set of potential classification functions. Given an example $\boldsymbol{x}_n$, all of the ensemble members provide probabilistic predictions, given as a vector of real-valued confidence scores

$$\hat{\boldsymbol{p}}_n^t = f_t(\boldsymbol{x}_n) = \left(\hat{p}_{n1}^t, \ldots, \hat{p}_{nK}^t\right) \in \mathbb{R}^K, \tag{6.2}$$

where each score expresses a preference for predicting the label λ_k as irrelevant, if $\hat{p}_k < 0$, or relevant, if $\hat{p}_k > 0$. The scores provided by individual members of an ensemble can be aggregated into a single vector of confidence scores by calculating the element-wise vector sum

$$\hat{\boldsymbol{p}}_n = F(\boldsymbol{x}_n) = \hat{\boldsymbol{p}}_n^1 + \cdots + \hat{\boldsymbol{p}}_n^T \in \mathbb{R}^K, \tag{6.3}$$

which can subsequently be turned into the final prediction of the ensemble in the form of a binary label vector (cf. Section 6.5).

> The BOOMER algorithm allows to specify the number of rules to be included in a model via the parameter `--max-rules`. By default, 1000 rules are learned.

Rules as Classification Functions

As individual ensemble members, we use conjunctive classification rules as introduced in Section 4.1. Such rules can be viewed as classification functions, where the body is a mathematical function $b : \mathcal{X} \to \{0, 1\}$ that evaluates to 1 if a given example satisfies all conditions in the body or to 0 if at least one of its conditions is not met. An individual condition compares the value of the l-th attribute of an example to a constant, by using a relational operator, such as = and ≠, if the attribute A_l is nominal, or ≤ and >, if A_l is numerical or ordinal. In accordance with (6.2), the head of a rule assigns a numerical score to each label. If a given example $\boldsymbol{x}$ belongs to the axis-parallel region in the feature space $\mathcal{X}$ covered by the rule, i.e., if it satisfies all conditions in the rule's body, a vector $\hat{\boldsymbol{p}}$ is predicted. If the example is not covered, a null vector is predicted instead. Consequently, a probabilistic rule of this kind can be considered a mathematical function $f : \mathcal{X} \to \mathbb{R}^K$, defined as

$$f(\boldsymbol{x}) = b(\boldsymbol{x})\,\hat{\boldsymbol{p}}. \tag{6.4}$$

This is similar to the notation used by Dembczyński, Kotłowski, and Słowiński (2010) in the context of single-label classification. However, in the multi-label setting, we consider the head as a vector rather than a scalar to enable rules to predict for several labels. In this chapter, we restrict ourselves to rules with single-label or complete multi-label heads. Whereas the latter assign a non-zero confidence score to each available label, the vector predicted by the former provides a non-zero prediction for exactly one label and assigns zeros to the others. In general, the notation introduced above is also suitable for representing partial heads.

> The BOOMER algorithm allows to specify the preferred type of rule heads via the following parameter:
>
> ```
> --head-type [auto,single-label,complete]
> ```
>
> If set to `auto` (the default value), the most suitable type is chosen automatically, depending on the loss function.

6.2 Multivariate Boosting

Surrogate Loss Functions

An ensemble of additive functions $F = (f_1, \ldots, f_t)$, as introduced in Section 6.1, should be trained such that the expected empirical risk with respect to a certain (surrogate) *loss function* ℓ is minimized. As the members of an ensemble predict numerical confidence scores rather than binary label vectors, discrete functions, such as the evaluation measures introduced in Section 3.3, are not suited to assess the quality of potential ensemble members during training. Instead, continuous loss functions that can be minimized in place of the actual target measure shoudl be used as surrogates. For this purpose, we use multivariate loss functions $\ell : \{-1, +1\}^K \times \mathbb{R}^K \rightarrow \mathbb{R}$. They take two vectors $\boldsymbol{y}_n$ and $\hat{\boldsymbol{p}}_n$ as arguments. The former corresponds to the true labeling of an example $\boldsymbol{x}_n$. It specifies whether individual labels λ_k are relevant ($y_{nk} = +1$) or irrelevant ($y_{nk} = -1$) to the respective example.[1] The latter represents the predictions of the ensemble members according to (6.3). Section 6.4 presents surrogates for minimizing the Hamming loss and the subset 0/1 loss. They are used for the experiments in Section 6.6.

1: According to (3.1), ground truth labels are given in the form $y_{nk} \in \{0, 1\}$. Such a representation can easily be transformed into the form $y_{nk} \in \{-1, +1\}$ required here.

Stagewise Additive Modeling

Given a specific loss function ℓ to be optimized, we are concerned with the minimization of the regularized training objective

$$\mathcal{R}(F) = \sum_{n=1}^{N} \ell(\boldsymbol{y}_n, \hat{\boldsymbol{p}}_n) + \sum_{t=1}^{T} \Omega(f_t), \tag{6.5}$$

where Ω denotes an (optional) regularization term that may be used to penalize the complexity of the individual ensemble members. It may help avoid overfitting and ensure the convergence towards a global optimum if ℓ is not convex. Unfortunately, constructing an ensemble of additive functions that minimizes the objective given above is a hard optimization problem. In gradient boosting, this problem is tackled by training the model in a stagewise procedure, where the individual ensemble members are added one after the other, as originally proposed by J. H. Friedman, Hastie, and Tibshirani (2000). At each iteration t, the vector of scores that the existing ensemble members predict for an example $\boldsymbol{x}_n$ can be calculated based on the predictions of the previous iteration. We denote it as

$$F_t(\boldsymbol{x}_n) = F_{t-1}(\boldsymbol{x}_n) + f_t(\boldsymbol{x}_n) = \left(\hat{\boldsymbol{p}}_n^1 + \cdots + \hat{\boldsymbol{p}}_n^{t-1}\right) + \hat{\boldsymbol{p}}_n^t. \tag{6.6}$$

Substituting the additive calculation of the predictions into the objective function given in (6.5) yields the objective to be minimized by the ensemble member added at the t-th iteration. It calculates as

$$\mathcal{R}(f_t) = \sum_{n=1}^{N} \ell(\boldsymbol{y}_n, F_{t-1}(\boldsymbol{x}_n)) + \Omega(f_t). \tag{6.7}$$

Taylor Approximation

To efficiently minimize the training objective when about to be adding a new ensemble member f_t, we rewrite (6.7) in terms of the second-order multivariate Taylor approximation

$$\mathcal{R}(f_t) \approx \sum_{n=1}^{N} \left(\ell(\boldsymbol{y}_n, F_{t-1}(\boldsymbol{x}_n)) + \boldsymbol{g}_n \hat{\boldsymbol{p}}_n^t + \frac{1}{2} \hat{\boldsymbol{p}}_n^t H_n \hat{\boldsymbol{p}}_n^t \right) + \Omega(f_t), \tag{6.8}$$

where $\boldsymbol{g}_n = (g_{n1}, \ldots, g_{nK})$ denotes the vector of first-order partial derivatives of the loss function ℓ with respect to the existing ensemble members' predictions for a particular example $\boldsymbol{x}_n$ and individual labels λ_k. Accordingly, the Hessian matrix $H_n = ((h_{n11}, \dots, h_{n1K}), \dots, (h_{nK1}, \dots, h_{nKK}))$ consists of second-order partial derivatives corresponding to pairs of labels. We compute individual gradients and Hessians as

$$g_{ni} = \frac{\partial \ell}{\partial \hat{p}_{ni}} (\boldsymbol{y}_n, F_{t-1}(\boldsymbol{x}_n)) \text{ and } h_{nij} = \frac{\partial \ell}{\partial \hat{p}_{ni} \partial \hat{p}_{nj}} (\boldsymbol{y}_n, F_{t-1}(\boldsymbol{x}_n)). \quad (6.9)$$

Training Objective

By removing constant terms, (6.8) can be further simplified, resulting in the approximated training objective

$$\widetilde{\mathcal{R}}(f_t) = \sum_{n=1}^{N} \left(\boldsymbol{g}_n \hat{\boldsymbol{p}}_n^t + \frac{1}{2} \hat{\boldsymbol{p}}_n^t H_n \hat{\boldsymbol{p}}_n^t \right) + \Omega(f_t). \quad (6.10)$$

At each training iteration, the objective function $\widetilde{\mathcal{R}}$ can be used as a quality measure to decide which of the potential ensemble members improves the current model the most. This requires the predictions of the potential ensemble members for examples $\boldsymbol{x}_n$ to be known. How these predictions can be found depends on the type of ensemble members and the loss function at hand. Section 6.3 presents solutions to this problem when using classification rules as the additive functions of an ensemble.

6.3 Induction of Rules

Sequential Model Assemblage

In the following, we outline the algorithm used by BOOMER for learning an ensemble of gradient boosted single- or multi-label rules that minimize a given loss function in expectation. It is based on the mathematical foundations introduced in Section 6.2 and is implemented in adherence to the modular framework presented in Chapter 4. The basic structure of the iterative procedure used for assembling an ensemble is shown in Algorithm 6. As previously discussed in Section 4.2, rules only provide predictions for examples they cover. The first rule $f_1 : \hat{\boldsymbol{y}}_1 \leftarrow b_1$ in the ensemble is a default rule covering all examples, i.e., $b_1(\boldsymbol{x}) = 1, \forall \boldsymbol{x}$. In subsequent iterations of the algorithm, more specific rules are added. All rules f_t, including the default rule, contribute to the final predictions of the ensemble according to the probabilistic scores assigned to individual labels by their heads $\hat{\boldsymbol{p}}_t$. The scores are chosen such that the objective function in (6.10) is minimized. At each iteration t, this requires the label space statistics S, consisting of gradients and Hessians, to be (re-)calculated based on the scores $\hat{p}_{nk}$ predicted by the current model for each example $\boldsymbol{x}_n$ and label λ_k, as well as the corresponding true labels $y_{nk} \in \{-1, +1\}$ (cf. Algorithm 6, line 1 and line 5). At the first iteration, the predictions for all examples and labels are zero, i.e., $\hat{p}_{nk} = 0, \forall n, k$ if $t = 1$. While the default rule always provides confidence scores for each label, the remaining rules may either predict for a single label or all labels. As previously mentioned in Section 6.1, the type of rule heads can be specified via a hyper-parameter. Both variants are considered in the experimental study presented in Section 6.6. The computations that are necessary to obtain loss-minimizing predictions for the default rule and each of the remaining rules are presented in the section "Computation of Predictions" below.

Algorithm 6: Learning an ensemble of boosted classification rules

input : Training examples $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_n^N$, first and second derivative ℓ' and ℓ'' of the loss function, number of rules T, shrinkage parameter η

output: Ensemble of rules F

```
1  S = {(g_n, H_n)}_n^N = calculate gradients and Hessians w.r.t. ℓ' and ℓ''
2  w_1 = set weights for each example to 1
3  f_1 : ŷ_1 ← b_1 with b_1(x) = 1, ∀x and ŷ_1 = FIND_HEAD(D, w_1, S, b_1)
4  for t = 2 to T do
5      S = update gradients and Hessians of examples covered by f_{t-1}
6      w_t = obtain a weight for each example via instance sampling
7      f_t : ŷ_t ← b_t = REFINE_RULE(D, w_t, S)
8      ŷ_t = FIND_HEAD(D, w_t, S, b_t)
9      ŷ_t = η · ŷ_t
10 return ensemble of rules F = {f_1, ..., f_T}
```

Induction of Individual Rules

To learn the rules $f_2, \ldots, f_T$, we use a greedy top-down search, where the body is iteratively refined by adding new conditions, and the head is adjusted accordingly at each step. The algorithm used for the refinement of rules is outlined in the section "Refinement of Rules". As discussed in Section 4.3, instance sampling can be used to learn each rule on a different sample of the training examples (cf. Algorithm 6, line 6). As more diversified and less correlated rules are learned, this reduces the variance of the ensemble members. However, once the construction of a rule has finished, its predictions are recomputed with respect to the entire training data (cf. Algorithm 6, line 8), which we have found to be an effective countermeasure against overfitting the sample used for constructing the rule. As an additional measure to reduce the risk of fitting noise in the data, the scores predicted by a rule may be multiplied by a shrinkage parameter $\eta \in (0, 1]$ (cf. Algorithm 6, line 9). As discussed in Section 4.3, small values for η reduce the impact of individual rules on the overall model.

ⓘ The reassessment of predictions with respect to the entire training data can be turned off via the parameter `--recalculate-predictions`. The weight of individual rules can be controlled via the shrinkage parameter `--shrinkage`. Its default value is 0.3.

Computation of Predictions

In Algorithm 6, the function FIND_HEAD is used to find optimal confidence scores to be predicted by a particular rule f_t, i.e., scores that minimize the objective function $\widetilde{\mathcal{R}}$ introduced in (6.10). In addition, it provides an estimate of their quality. Because rules provide the same predictions for all examples they cover and abstain for others, the objective function in (6.10) can be further simplified. We can sum up the gradient vectors and Hessian matrices that correspond to the covered examples, resulting in the objective

$$\widetilde{\mathcal{R}}(f_t) = \boldsymbol{g}\hat{\boldsymbol{p}} + \frac{1}{2}\hat{\boldsymbol{p}}H\hat{\boldsymbol{p}} + \Omega(f_t), \tag{6.11}$$

where $\boldsymbol{g} = \sum_n (b(\boldsymbol{x}_n) w_n \boldsymbol{g}_n)$ denotes the element-wise weighted sum of the gradient vectors and $H = \sum_n (b(\boldsymbol{x}_n) w_n H_n)$ corresponds to the sum of the Hessian matrices. As shown in Algorithm 7, the sums of gradients and Hessians are provided to the function FIND_HEAD to determine a rule's predictions and a corresponding estimate of its quality.

L_2 Regularization

To penalize extreme predictions, we use the L_2 regularization term

$$\Omega_{L2}(f_t) = \frac{1}{2}\delta \left\|\hat{\boldsymbol{p}}^t\right\|_2^2, \tag{6.12}$$

where $\|\boldsymbol{x}\|_2$ denotes the Euclidean norm and $\delta \geq 0$ is a hyper-parameter that controls the weight of the regularization term.

> The weight of the L_2 regularization term can be specified via the parameter `--l2-regularization-weight`. Its default value is 1.0.

System of Linear Equations

To ensure that predictions $\hat{\boldsymbol{p}}$ minimize the regularized training objective $\widetilde{\mathcal{R}}$, we equate the first partial derivative of (6.11) with respect to $\hat{\boldsymbol{p}}$ with zero:

$$\begin{aligned} \frac{\partial \widetilde{\mathcal{R}}}{\partial \hat{\boldsymbol{p}}}(f_t) &= \boldsymbol{g} + H\hat{\boldsymbol{p}} + \delta\hat{\boldsymbol{p}} = \boldsymbol{g} + (H + R)\hat{\boldsymbol{p}} = 0 \\ &\Longleftrightarrow (H + R)\hat{\boldsymbol{p}} = -\boldsymbol{g}, \end{aligned} \tag{6.13}$$

where $R = \mathrm{diag}(\delta)$ is a diagonal matrix with the elements on the diagonal set to the value δ. (6.13) can be considered as a system of K linear equations, where $H + R$ is a matrix of coefficients, $-\boldsymbol{g}$ is a vector of ordinates and $\hat{\boldsymbol{p}}$ is the vector of unknowns to be determined. For commonly used loss functions, including those in Section 6.4, the sums of Hessians h_{ij} and h_{ji} are equal. Consequently, the matrix of coefficients is symmetrical.

Closed-form Solution

In the general case, i.e., if the loss function is non-decomposable, the linear system in (6.13) must be solved to determine the optimal multi-label head $\hat{\boldsymbol{p}}$. However, when dealing with a decomposable loss function, the first and second derivative with respect to a particular element $\hat{p}_i \in \hat{\boldsymbol{p}}$ is independent of any other element $\hat{p}_j \in \hat{\boldsymbol{p}}$. This causes the sums of

Algorithm 7: FIND_HEAD

input : Sums of gradients $\boldsymbol{g} = \sum_n (b(\boldsymbol{x}_n) w_n \boldsymbol{g}_n)$,
sums of Hessians $H = \sum_n (b(\boldsymbol{x}_n) w_n H_n)$
output: Single- or multi-label head $\hat{\boldsymbol{p}}$, quality q

$\boldsymbol{g} = \sum_n b(\boldsymbol{x}_n) w_n \boldsymbol{g}_n, H = \sum_n b(\boldsymbol{x}_n) w_n H_n$
if loss is decomposable **or** searching for a single-label head **then**
    $\hat{\boldsymbol{p}}$ = obtain $\hat{p}_k$ w.r.t. $\boldsymbol{g}$ and H for each label acc. to (6.14)
    if searching for a single-label head **then**
        $\hat{\boldsymbol{p}}$ = find best single-label prediction $\hat{p}_k \in \hat{\boldsymbol{p}}$ w.r.t. (6.10)
else
    $\hat{\boldsymbol{p}}$ = obtain $(\hat{p}_1, \dots, \hat{p}_K)$ w.r.t. $\boldsymbol{g}$ and H by solving (6.13)
q = evaluate (6.11) w.r.t. $\boldsymbol{g}$, H and $\hat{\boldsymbol{p}}$
return head $\hat{\boldsymbol{p}}$, quality q

Hessians h_{ij} that do not exclusively depend on $\hat{p}_i$, i.e., if $i \neq j$, to become zero. In such a case, the linear system reduces to K independent equations — one for each label. This enables to compute the optimal prediction $\hat{p}_i$ for the i-th label via the closed-form solution

$$\hat{p}_i = -\frac{g_i}{h_{ii} + \delta}. \tag{6.14}$$

Similarly, when interested in single-label rules that predict for the i-th label, the predictions $\hat{p}_j$ with $j \neq i$ are known to be zero because the rule will abstain for the corresponding labels. Consequently, (6.14) can be used to determine the predictions of single-label rules even if the loss function is non-decomposable.

Refinement of Rules

Top-down Hill Climbing

To learn a new rule, we use a greedy top-down search, also referred to as *top-down hill climbing* (Fürnkranz, Gamberger, and Lavrač, 2012), as previously mentioned in Section 4.3. Algorithm 8 is meant to outline the general procedure of such a search algorithm. For brevity, it does not include any algorithmic optimizations that can drastically improve the computational efficiency in practice. An efficient implementation, as well as possible extensions to the basic procedure, are discussed in Chapter 7. The search for a new rule starts with an empty body that is successively refined by adding additional conditions.

Adding conditions to its body causes a rule to become more specific and results in fewer examples being covered. The conditions, which may be used to refine an existing body, result from the feature values of the training examples in the case of nominal attributes or from averaging adjacent values in the case of numerical attributes. In addition to instance sampling, we use *feature sampling* to select a subset of the available attributes whenever a new condition should be added (cf. Algorithm 8,

Algorithm 8: REFINE_RULE

input : Training examples $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_n^N$, weights $\boldsymbol{w}$, statistics $S = \{(\boldsymbol{g}_n, \boldsymbol{H}_n)\}_n^N$, current rule f, its quality q (both optional)
output: Best rule f^*

best rule $f^* = f$, best quality q^* = q
$\mathcal{A}'$ = select a random subset of attributes from $\mathcal{D}$
foreach possible condition c on attributes $\mathcal{A}'$ and examples $\mathcal{D}$ **do**
  $f' : b' \rightarrow \hat{\boldsymbol{p}}'$ = copy of current rule f
  add condition c to body b'
  $\boldsymbol{g} = \sum_n (b(\boldsymbol{x}_n) w_n \boldsymbol{g}_n), \boldsymbol{H} = \sum_n (b(\boldsymbol{x}_n) w_n \boldsymbol{H}_n)$
  head $\hat{\boldsymbol{p}}'$, quality q' = FIND_HEAD $(\boldsymbol{g}, \boldsymbol{H})$
  if $q' < q^*$ **then**
    update best rule $f^* = f'$ and its quality $q^* = q'$
if $f^* \neq f$ **then**
  $\mathcal{D}'$ = subset of $\mathcal{D}$ covered by f^*
  return REFINE_RULE $(\mathcal{D}', \boldsymbol{w}, S, f^*, q^*)$
return best rule f^*

line 2). This leads to more diverse ensembles of rules and reduces the computational costs by limiting the number of potential candidate rules. For each condition that may be added to the current body at a particular iteration, the head of the rule is updated via the function FIND_-HEAD discussed in the previous section "Computation of Predictions" (cf. Algorithm 8, line 7). When learning single-label rules and if not configured otherwise, each refinement of the current rule is obliged to predict for the same label (omitted in Algorithm 8 for brevity). Among all refinements, the one that minimizes the regularized objective in (6.10) is chosen. If no refinement results in an improvement according to said objective, the refinement process stops. By default, no additional stopping criteria are used, and therefore they are omitted from Algorithm 8.

ⓘ The specificity of rules can optionally be restricted via the parameters `--max-conditions` and `--min-coverage`. The former specifies the maximum number of conditions to be included in a rule's body, whereas the latter allows stopping the refinement of rules as soon as they cover fewer examples than required (no restrictions apply by default). In addition, the parameter `--max-head-refinements` specifies the number of successive refinements that are allowed to predict for different labels (defaults to 1).

6.4 Surrogate Loss Functions

Logistic Surrogate Losses

As mentioned in Section 6.2, we use continuous and differentiable surrogate loss functions in place of the actual target measure to assess the quality of individual candidate rules during training. As surrogates for the Hamming loss and the subset 0/1 loss (cf. Section 3.3), we use different variants of the *logistic loss*. This loss function, which is equivalent to *cross entropy*, is the basis for logistic regression and is commonly used in boosting approaches to single-label classification (early uses go back to J. H. Friedman, Hastie, and Tibshirani, 2000).

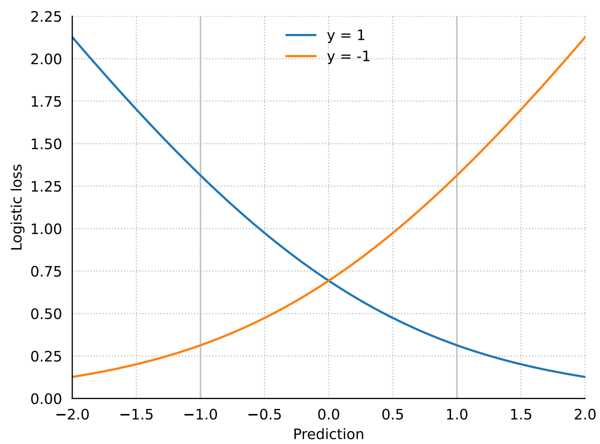


Figure 6.1: Value of the logistic loss function, given a real-valued prediction $\hat{p}$ for a single label with ground truth $y = -1$ and $y = 1$, respectively.

Label-wise Logistic Loss

Function Definition

To cater for the characteristics of the Hamming loss, the *label-wise logistic loss* applies the logistic loss function in Figure 6.1 to each label individually:

$$\ell_{l.w.\text{-}log.}(\boldsymbol{y}_n, \hat{\boldsymbol{p}}_n) \coloneqq \sum_{k=1}^{K} \log\left(1 + \exp\left(-y_{nk}\hat{p}_{nk}\right)\right). \tag{6.15}$$

Following the formulation of this objective, $\hat{p}_{nk}$ can be considered as log-odds, which estimate the probability of $y_{nk} = 1$ as

$$\text{logistic}\left(\hat{p}_{nk}\right) = \frac{1}{1 + \exp\left(-\hat{p}_{nk}\right)}. \tag{6.16}$$

Under the assumption of label independence, the logistic loss has been shown to be a consistent surrogate loss for the Hamming loss (Dembczyński, Kotłowski, and Hüllermeier, 2012; Gao and Z.-H. Zhou, 2013).

First and Second-order Derivatives

To compute the gradients and Hessians that guide the training process, the first and second partial derivative of the loss function are required. Given an example $\boldsymbol{x}_n$, the gradient with respect to the current prediction for its i-th label is

$$g_{ni} = \frac{-y_{ni}}{1 + \exp\left(y_{ni}\hat{p}_{ni}\right)}, \tag{6.17}$$

whereas the corresponding Hessian calculates as

$$h_{nii} = \frac{1}{1 + \exp\left(y_{ni}\hat{p}_{ni}\right)} - \frac{1}{\left(1 + \exp\left(y_{ni}\hat{p}_{ni}\right)\right)^2}. \tag{6.18}$$

Numerical Stability

Computing (6.17) and (6.18) in a naive way is likely to cause numerical problems due to the limited precision of floating point operations. As the exponential function $\exp(x)$ grows very fast, overflows may occur even for moderately large values x. To implement the calculation of gradients in a numerically stable way, we make use of a numerically stable formulation of the logistic function in (6.16), defined as

$$\text{logistic}\left(\hat{p}_{ni}\right) = \begin{cases} 1/\left(1 + \exp\left(-\hat{p}_{ni}\right)\right) & \text{if } \hat{p}_{ni} \geq 0 \\ \exp\left(\hat{p}_{ni}\right)/\left(1 + \exp\left(\hat{p}_{ni}\right)\right) & \text{otherwise.} \end{cases} \tag{6.19}$$

It can be used to rewrite the calculation of gradients in (6.17) as

$$g_{ni} = \begin{cases} \text{logistic}\left(\hat{p}_{ni}\right) - 1 & \text{if } y_{ni} = 1 \\ \text{logistic}\left(\hat{p}_{ni}\right) & \text{otherwise.} \end{cases} \tag{6.20}$$

The logistic function in (6.19) is also used to rewrite the calculation of Hessians. In addition, the reformulation uses the function

$$\text{logistic}^2\left(\hat{p}_{ni}\right) = \begin{cases} 1/\left(1 + \exp\left(-\hat{p}_{ni}\right)\right)^2 & \text{if } \hat{p}_{ni} \geq 0 \\ \exp\left(\hat{p}_{ni}\right)^2/\left(1 + \exp\left(\hat{p}_{ni}\right)\right)^2 & \text{otherwise.} \end{cases} \tag{6.21}$$

By making use of these two functions, the calculation of Hessians in (6.18) can be computed in a numerically stable way as

$$h_{ni} = \text{logistic}\left(\hat{p}_{ni}\right) - \text{logistic}^2\left(\hat{p}_{ni}\right). \tag{6.22}$$

Example-wise Logistic Loss

Function Definition

As the label-wise logistic loss can be calculated by aggregating the values that result from applying the logistic loss function to each label individually, it is label-wise decomposable. In contrast, the *example-wise logistic loss*

$$\ell_{ex.w.\text{-}log.}(\boldsymbol{y}_n, \hat{\boldsymbol{p}}_n) \coloneqq \log\left(1 + \sum_{k=1}^{K} \exp\left(-y_{nk}\hat{p}_{nk}\right)\right) \tag{6.23}$$

is non-decomposable, as it cannot be computed via label-wise aggregation. This smooth and convex surrogate is proposed by Amit, Dekel, and Singer (2007), who show that it provides an upper bound of the subset 0/1 loss.

First and Second-order Derivatives

The gradients with respect to the current prediction for the i-th label of a particular example $\boldsymbol{x}_n$ can be computed according to the first partial derivative of the example-wise logistic loss as

$$g_{ni} = \frac{-y_{ni}\exp\left(-y_{ni}\hat{p}_{ni}\right)}{1 + \sum_{k=1}^{K}\exp\left(-y_{nk}\hat{p}_{nk}\right)}. \tag{6.24}$$

The Hessian with respect to the prediction for a single label λ_i, i.e., an element that corresponds to the diagonal of the Hessian matrix, computes as

$$h_{nii} = \frac{\exp\left(-y_{ni}\hat{p}_{ni}\right)}{1 + \sum_{k=1}^{K}\exp\left(-y_{nk}\hat{p}_{nk}\right)} \cdot \left(1 - \frac{\exp\left(-y_{ni}\hat{p}_{ni}\right)}{1 + \sum_{k=1}^{K}\exp\left(-y_{nk}\hat{p}_{nk}\right)}\right), \tag{6.25}$$

whereas the Hessian with respect to two different labels λ_i and λ_j, i.e., an element that corresponds to the upper (or lower) triangle of the Hessian matrix, calculates as

$$h_{nij} = \frac{-y_{ni}y_{nj}\exp\left(-y_{ni}\hat{p}_{ni} - y_{nj}\hat{p}_{nj}\right)}{1 + \sum_{k=1}^{K}\exp\left(-y_{nk}\hat{p}_{nk}\right)} \cdot \frac{1}{1 + \sum_{k=1}^{K}\exp\left(-y_{nk}\hat{p}_{nk}\right)}. \tag{6.26}$$

Numerical Stability

To compute the gradients and Hessians with respect to the example-wise logistic loss in a numerically stable manner, we exploit the equality

$$\frac{\exp\left(-y_{ni}\hat{p}_{ni}\right)}{1 + \sum_{k=1}^{K}\exp\left(-y_{nk}\hat{p}_{nk}\right)} = \frac{\exp\left(-y_{ni}\hat{p}_{ni} - m\right)}{\exp\left(-m\right) + \sum_{k=1}^{K}\exp\left(-y_{nk}\hat{p}_{nk} - m\right)}, \tag{6.27}$$

where $m = \max\left(0, \max_{k=1}^{K}\left(y_{nk}\hat{p}_{nk}\right)\right)$. It can be used to rewrite the computation of gradients in (6.24) as

$$g_{ni} = \frac{-y_{ni}\exp\left(-y_{ni}\hat{p}_{ni} - m\right)}{\exp\left(-m\right) + \sum_{k=1}^{K}\exp\left(-y_{nk}\hat{p}_{nk} - m\right)}. \tag{6.28}$$

Moreover, it enables to rewrite the computation of Hessians, previously introduced in (6.25) and (6.26), as

$$\begin{aligned} h_{nii} =& \frac{\exp\left(-y_{ni}\hat{p}_{ni} - m\right)}{\exp\left(-m\right) + \sum_{k=1}^{K} \exp\left(-y_{nk}\hat{p}_{nk} - m\right)} \\ &\cdot \left(1 - \frac{\exp\left(-y_{ni}\hat{p}_{ni} - m\right)}{\exp\left(-m\right) + \sum_{k=1}^{K} \exp\left(-y_{nk}\hat{p}_{nk} - m\right)}\right), \end{aligned} \tag{6.29}$$

when it comes to gradients that corresponds to the diagonal of the Hessian matrix, and

$$\begin{aligned} h_{nij} =& \frac{-y_{ni}\hat{p}_{ni} \exp\left(-y_{ni}\hat{p}_{ni} - y_{nj}\hat{p}_{nj} - m\right)}{\exp\left(-m\right) + \sum_{k=1}^{K} \exp\left(-y_{nk}\hat{p}_{nk} - m\right)} \\ &\cdot \frac{\exp\left(-m\right)}{\exp\left(-m\right) + \sum_{k=1}^{K} \exp\left(-y_{nk}\hat{p}_{nk} - m\right)}, \end{aligned} \tag{6.30}$$

when Hessians that correspond to the upper (or lower) triangle of the Hessian matrix should be computed.

> The BOOMER algorithm can be instructed to use one of the mentioned losses by specifying the following parameter:
>
> ```
> --loss [logistic-label-wise,logistic-example-wise]
> ```
>
> A description of additional loss functions can be found in the documentation.

6.5 Loss-Minimizing Prediction

Predicting for an example $\boldsymbol{x}_n$ involves two steps: First, the scores that are provided by individual rules are aggregated into a vector of confidence scores $\hat{\boldsymbol{p}}_n$ according to (6.3). Second, the vector $\hat{\boldsymbol{p}}_n$ must be turned into a binary label vector $\hat{\boldsymbol{y}}_n \in \mathcal{Y}$. It should minimize the expected risk with respect to the used loss function, such that

$$\hat{\boldsymbol{y}}_n = \arg\min_{\hat{\boldsymbol{y}}_n} \ell\left(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n\right).$$

Label-wise Prediction

In case of the label-wise logistic loss in (6.15), we compute the binary label vector in a label-wise manner as

$$\hat{\boldsymbol{y}}_n = \left(\operatorname{sgn}\left(\hat{p}_{n1}\right), \dots, \operatorname{sgn}\left(\hat{p}_{nK}\right)\right), \tag{6.31}$$

where $\operatorname{sgn}(x)$ evaluates to 1 if $x > 0$ and 0 otherwise.

Example-wise Prediction

To predict the label vector that minimizes the example-wise logistic loss given in (6.23), we return the label vector among the vectors in the training data, which minimizes the loss, i.e.,

$$\hat{\boldsymbol{y}}_n = \arg\min_{\boldsymbol{y} \in \mathcal{D}} \ell_{\text{ex.w.-log.}}\left(\boldsymbol{y}, \hat{\boldsymbol{p}}_n\right). \tag{6.32}$$

Similar to the label powerset transformation method (cf. Section 3.6), the prediction is restricted to the label vectors that are present in the training data. Senge, Coz, and Hüllermeier (2013) argue that often only a small fraction of the 2^K possible label combinations are observed in practice. For this reason, the correctness of an unseen label combination becomes increasingly unlikely for large datasets. According to preliminary experiments, predicting one of the known label vectors usually results in an improvement of predictive performance in terms of the subset 0/1 loss.

> The prediction scheme to be used by the BOOMER algorithm can be specified via the following parameter:
>
> ```
> --predictor [auto,label-wise,example-wise]
> ```
>
> If set to `auto` (the default value), the most suitable scheme is chosen automatically depending on the loss function.

6.6 Experimental Evaluation

Experimental Setup

In an experimental study, we evaluated the ability of the BOOMER algorithm[2] to minimize the Hamming and subset 0/1 loss, using the label-wise and example-wise logistic loss as surrogates. In each case, we considered both, single- and multi-label rules, resulting in four different variants (*l.w.-log. single*, *l.w.-log. multi*, *ex.w.-log. single*, and *ex.w.-log. multi*). For each of them, we tuned the shrinkage parameter $\eta \in \{0.1, 0.3, 0.5\}$ and the regularization weight $\delta \in \{0.0, 0.25, 1.0, 4.0, 16.0, 64.0\}$, using *bootstrap bias corrected cross validation (BBC-CV)* (Tsamardinos, Greasidou, and Borboudakis, 2018), which allows incorporating parameter tuning and evaluation in a computationally efficient manner.

2: The experiments that are discussed in this section were conducted using version 0.1.0 of the BOOMER algorithm. It is available at `https://github.com/mrapp-ke/Boomer/releases/tag/0.1.0`.

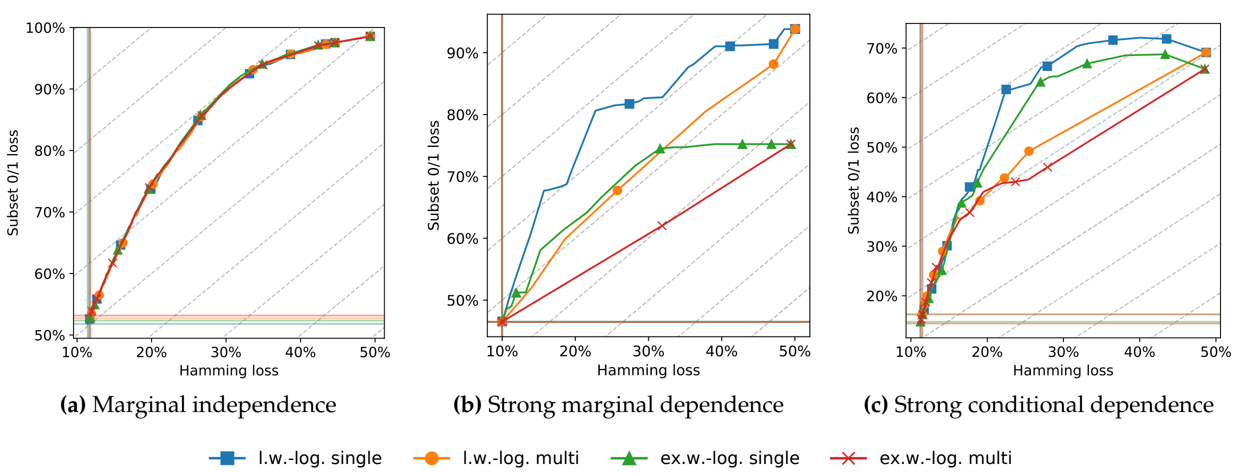
Bootstrap Bias Corrected Cross Validation

The idea of BBC-CV is to train a model for each parameter configuration using regular cross validation as described in Section 3.3. When performing a CV, each example in the given dataset is included exactly once in the test set. As a result, one obtains a prediction for all available examples and each parameter setting. These predictions serve as the basis for the randomized procedure that is outlined in the following:

I. **Parameter Tuning.** First, a subset of the available examples is drawn with replacement. Then, for each parameter setting, the quality of the predictions provided for the selected examples is assessed in terms of a particular target measure. This allows for determining the best configuration for the selected examples.

II. **Evaluation.** To obtain an independent estimate of the best configuration's quality, its out-of-sample predictions are evaluated with respect to different evaluation measures.

The previously described procedure should be repeated several times (we used 100 iterations). The evaluation results obtained at each iteration must be averaged to obtain a final estimate of an algorithm's strength, given the option to pick optimal parameters from predefined configurations.



(a) Marginal independence **(b)** Strong marginal dependence **(c)** Strong conditional dependence

Figure 6.2: Predictive performance of different approaches with respect to the Hamming loss and the subset 0/1 loss on three synthetic datasets. Starting at the top right, each curve shows the performance as more rules are added. The slope of the isometrics refers to an even improvement of Hamming and subset 0/1 loss. The symbols indicate $1, 2, 4, 8, \ldots, 512, 1000$ rules in the model.

Synthetic Data

Properties of the Data

We used three synthetic datasets, each containing 10, 000 examples and six labels, as proposed by Dembczyński, Waegeman, et al. (2012). Our goal was to analyze the effects of using single- or multi-label rules given different types of label dependencies. The attributes correspond to two-dimensional points drawn randomly from the unit circle and the labels are assigned according to linear decision boundaries through its origin. The results are shown in Figure 6.2.

Marginal Independence

In the case of marginal independence, points are sampled for each label independently. Noise is introduced by randomly flipping 10% of the labels, such that the scores achievable by the Bayes-optimal classifier on an independent test set are 10% for Hamming loss and $\approx$ 47% for subset 0/1 loss. As it is not possible — by the construction of the dataset — to find a rule body that is suited to predict for more than one label, multi-label rules cannot be expected to provide any advantage. In fact, despite very similar trajectories, the single-label variants achieve slightly better losses in the limit. This indicates that, most probably due to the number of uninformative features, the approaches that aim at learning multi-label rules struggle to induce pure single-label rules.

Strong Marginal Dependence

For modeling a strong marginal dependence between the labels, a small angle between the linear boundaries of two labels is chosen, such that they mostly coincide for the respective examples. Starting from different default rules, depending on the loss function they minimize, the algorithms converge to the same performances, which indicates that all variants can similarly deal with marginal dependencies. However, when using multi-label rules, the final subset 0/1 score is approached faster and more steadily, as a single rule provides predictions for several labels at once. In fact, it is remarkable that the ex.w.-log. multi variants already converge after two rules, whereas ex.w.-log. single needs six, one for each label, and optimizing for label-wise logistic loss takes much longer.

Strong Conditional Dependence

Finally, strong conditional dependence is obtained by randomly switching all labels for 10% of the examples. As a result, the score that is achievable by the Bayes-optimal classifier is 10% for both losses. While the trajectories are similar to before, the variants that optimize the non-decomposable loss achieve better results in the limit. Unlike the approaches that consider the labels independently, they seem to be less prone to the noise introduced at the example-level.

Real-world Benchmark Data

Baselines

We also conducted experiments on eight benchmark datasets from the Mulan and MEKA projects and compared the results to those of different baselines. A description of the datasets can be found in Section 3.2. As baselines, we considered the problem transformation methods binary relevance (BR), label powerset (LP), and classifier chains (CC), using XGBoost (T. Chen and Guestrin, 2016) as the base classifier. A detailed discussion of these transformation methods is given in Section 3.6. For BR and CC, we used the logistic loss. For LP, we used the softmax objective.

Table 6.1: Predictive performance (in percent) of different approaches with respect to Hamming loss, subset 0/1 loss and the example-wise F1 measure. For each evaluation measure and dataset, we report the ranks of the different approaches (small numbers) and highlight the best approach (bold text).

Dataset	l.w.-log.				ex.w.-log.				XGBoost					
	single		multi		single		multi		BR		LP		CC	
Subset 0/1 loss														
Birds	38.72	2	40.43	6	39.57	4.5	39.15	3	39.57	4.5	40.85	7	**38.30**	1
Emotions	73.33	6.5	73.33	6.5	70.48	3.5	65.24	2	70.48	3.5	**60.00**	1	71.43	5
Enron	87.81	7	86.82	6	84.35	4	83.53	2	85.34	5	**82.70**	1	83.86	3
Langlog	78.85	5	78.85	5	78.27	3	76.35	2	80.96	7	**70.38**	1	78.85	5
Medical	28.45	3	30.16	4	23.90	2	**23.04**	1	56.90	7	41.11	5	44.95	6
Scene	39.33	7	33.93	5	25.17	3	**23.26**	1	34.72	6	24.16	2	30.11	4
Slashdot	65.29	5	62.89	4	53.30	3	**49.75**	1	72.04	7	52.94	2	66.96	6
Yeast	83.72	7	82.27	5	78.60	4	75.81	2	82.72	6	**75.70**	1	76.59	3
Avg. rank		5.31		5.19		3.38		**1.75**		5.75		2.50		4.13
Hamming loss														
Birds	3.52	5	**3.16**	1	3.58	6	3.23	2	3.43	3	4.03	7	3.45	4
Emotions	18.81	5	20.00	7	18.17	2	18.41	3	18.73	4	**18.02**	1	19.29	6
Enron	4.56	3	**4.49**	1	4.90	5	4.91	6	4.52	2	5.75	7	4.63	4
Langlog	1.48	2.5	1.48	2.5	1.49	4.5	**1.45**	1	1.49	4.5	2.13	7	1.60	6
Medical	0.84	2.5	0.86	4	0.84	2.5	**0.79**	1	1.69	7	1.55	6	1.36	5
Scene	8.16	7	7.42	6	7.21	4	**6.63**	1	7.19	3	7.27	5	6.85	2
Slashdot	**4.15**	1	4.17	2	5.03	6	4.64	5	4.61	4	5.16	7	4.54	3
Yeast	19.27	2	19.84	5	19.44	4	19.29	3	**19.13**	1	21.22	7	20.11	6
Avg. rank		3.56		3.56		4.19		**2.75**		3.56		5.88		4.50
Example-wise F1 measure														
Birds	70.38	4	**72.42**	1	68.00	7	71.18	2	69.68	6	70.06	5	70.96	3
Emotions	58.19	7	59.06	6	64.56	3	65.75	2	61.90	5	**69.24**	1	62.59	4
Enron	52.86	6	53.87	4	53.73	5	53.91	3	54.76	2	46.61	7	**56.46**	1
Langlog	22.51	6	23.21	5	23.28	4	26.30	2	19.36	7	**33.17**	1	24.16	3
Medical	81.68	3	80.07	4	85.51	2	**85.97**	1	53.14	7	71.34	5	64.72	6
Scene	65.90	7	71.69	5	80.30	2	**81.69**	1	70.30	6	79.72	3	74.23	4
Slashdot	40.94	5	44.04	4	54.78	2	**58.62**	1	32.57	7	54.01	3	39.15	6
Yeast	61.56	6	61.03	7	62.78	3	**63.41**	1	62.54	4	61.71	5	62.92	2
Avg. rank		5.50		4.50		3.50		**1.63**		5.50		3.75		3.63

While we tuned the number of rules $T \in \{50, 100, \ldots, 10000\}$ for our algorithm, XGBoost comes with an integrated method for determining the number of trees. The learning rate and the L_2 regularization weight were tuned in the same value ranges for both. Moreover, we configured the BOOMER algorithm to use sampling with replacement to obtain a subset of the available training examples whenever a new rule should be learned and choose from $\lfloor \log_2 (L-1) + 1 \rfloor$ random attributes at each refinement iteration. For a fair comparison, we instructed XGBoost to sample 66% of the training examples at each iteration and select the same number of attributes at each split. As classifier chains are sensitive to the order of labels, we chose the best order among ten random permutations. According to their respective objectives, the BR baseline was tuned with respect to Hamming loss, LP and CC were tuned with respect to subset $0/1$ loss.

Experimental Results

In Table 6.1, we report the predictive performance of our methods and their competitors in terms of Hamming and subset $0/1$ loss. For completeness, we also report the example-wise F1 score (cf. Section 3.3). The Friedman test (cf. Section 3.4) indicates significant differences for all measures but the Hamming loss. The Nemenyi post-hoc test yields critical distances between the average ranks of $2.91/3.19$ for $\alpha = 0.1/0.05$. On average, ex.w.-log. multi ranks best in terms of subset $0/1$ loss. It is followed by LP, its counterpart ex.w.-log. single, and CC. As all of them aim to minimize the subset $0/1$ loss, it is expected that they rank better than their competitors directed at the Hamming loss. Most notably, since example-wise optimized multi-label rules achieve better results than single-label rules on all datasets (statistically significant with $\alpha = 0.1$), we conclude that the ability to induce such rules, which is a novelty of the proposed method, is crucial for minimizing subset $0/1$ loss. On the other hand, in terms of Hamming loss, rules that minimize the label-wise logistic loss are competitive to the BR baseline, without a clear preference for single- or multi-label rules. Interestingly, although the example-wise logistic loss aims at minimizing subset $0/1$ loss, when using multi-label rules, it also achieves remarkable results for the Hamming loss on some datasets and consequently even ranks best on average.

Dependence-Awareness

Goals of the Study

The Hamming loss and the subset $0/1$ loss, used in the previous experiments, are often considered prototypical examples of decomposable and non-decomposable multi-label evaluation measures. Whereas the former can principally be optimized by considering each label individually, the latter requires taking correlations between labels into account and therefore is often used to quantify a classifier's ability to model label dependencies. However, as argued in Section 3.3, the Hamming loss and the subset $0/1$ loss can both be criticized for being rather extreme. On the one hand, since the distribution between relevant and irrelevant examples for each label is typically quite unbalanced and because it does not take any dependencies between labels into account, the Hamming loss is often close to 0. On the other hand, since an entirely correct prediction becomes more difficult with an increasing number of labels, the subset $0/1$ loss is usually close to 1. For the following experiments, we employ a family of "dependence-aware" multi-label loss functions that allow for a more detailed analysis of a classifier's ability to capture label dependencies. First, we briefly recapitulate the formulation of these loss functions as proposed by Hüllermeier, Wever, et al. (2022) . Afterward, we discuss the experimental results that are presented in said paper, focusing on a comparison of the BOOMER algorithm to a variety of established MLC approaches.

Hüllermeier, Wever, Loza Mencía, Fürnkranz, and Rapp (2022): 'A flexible class of dependence-aware multi-label loss functions'

Dependence-Aware Evaluation Measures

Hüllermeier, Wever, et al. (2022) leverage the mathematical framework of *non-additive measures* (Sugeno, 1974) and *Choquet integrals* (Choquet, 1954) to formulate a class of multi-label evaluation measures that model the importance of correct predictions for label subsets rather than individual labels in a flexible way. In the following, we focus on two parameterized loss functions belonging to this class, namely the *polynomial loss* and the *binomial loss*. These loss functions comprise the Hamming loss and the subset 0/1 loss as special cases, where full importance is given to single labels or the entire labelset, and allow to interpolate between these two extremes. Using an *ordered weighted averaging* (OWA) aggregation (Yager and Filev, 1999; Yager and Kacprzyk, 2012), the polynomial and binomial loss can be written as evaluation measures of the form

$$\mathcal{M}_{OWA}(Y, \hat{Y}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{K} w_i u_{n\tau(i)} \text{ with } u_{n\tau(i)} = |y_{n\tau(i)} - \hat{y}_{n\tau(i)}|, \quad (6.33)$$

where the permutation function τ is such that $0 \leq u_{n\tau(1)} \leq u_{n\tau(2)} \leq \cdots \leq u_{n\tau(K)} \leq 1$ and the weights $w_1 + \cdots + w_K = 1$ specify the importance of a correct prediction for the $\tau(i)$-th label. Both losses are so-called *counting measures*, where the impact of a label subset on the overall performance is not affected by the labels in the set but only depends on its cardinality. This kind of symmetry property is certainly meaningful in MLC, where different labels are typically considered equally important, and performance metrics are usually invariant to the permutation of labels. In the case of counting measures, the weight of the $\tau(i)$-th label on the overall performance can be assessed in terms of a function $v : \mathbb{R} \rightarrow \mathbb{R}$ as

$$w_i = v\left(\frac{K - i + 1}{K}\right) - v\left(\frac{K - i}{K}\right). \quad (6.34)$$

Polynomial Loss

Depending on a parameter α, the polynomial loss assesses the importance of individual labels in terms of the convex function

$$v(x) = x^{\alpha} \quad \text{with} \quad \alpha \geq 1. \quad (6.35)$$

If $\alpha = 1$, the polynomial loss is equivalent to the Hamming loss. With increasing values for the parameter α, more emphasis is put on the correct prediction of larger label subsets. In the limit, i.e., if $\alpha \rightarrow \infty$, the polynomial loss becomes equivalent to the subset 0/1 loss.

Binomial Loss

The binomial loss focuses on label subsets with a predefined number of labels k. It assesses the weights of individual labels in terms of the function

$$v\left(\frac{x}{K}\right) = \frac{\binom{x}{k}}{\binom{K}{k}}. \quad (6.36)$$

Similar to the polynomial loss, the binomial loss comprises the Hamming loss and the subset 0/1 loss as special settings, where $k = 1$ and $k = K$, respectively. Again, it is possible to interpolate between the two losses by setting the parameter k to values in-between. However, in contrast to the polynomial loss, the binomial loss does not reward predictions for small labelsets with less than k elements.

Experimental Setup

To demonstrate how the polynomial and binomial loss can be used to analyze the "dependency awareness" of different multi-label classifiers, i.e., their ability to capture label dependencies, Hüllermeier, Wever, et al. (2022) evaluate the predictive performance of several MLC methods in terms of these losses. Their empirical study is based on a selection of nine benchmark datasets (cf. Section 3.2) and relies on 10-fold cross validation (cf. Section 3.3) to obtain unbiased estimates of a classifier's predictive performance. It includes a wide variety of commonly used problem transformation approaches and adaptation methods discussed in Section 3.6 and Section 3.7. The considered problem transformation approaches include the binary relevance (BR) and label powerset (LP) method, classifier chains (CC), and RAkEL. The latter was configured to use label subsets of varying sizes. We denote a first variant that deals with pairs of labels as RAkEL-2. A second variant that uses labelsets with five labels is referred to as RAkEL-5. The problem adaptation methods include ensembles of predictive clustering trees (EPCT) and BOOMER.[3] Similar to previous experiments, the label- and example-wise logistic loss in (6.15) and (6.23) were both used as the latter's training objective. We refer to these variants as BOOMER-l.w.-log. and BOOMER-ex.w.-log., respectively. The experiments can be reproduced by using the publicly available source code.[4] It employs the problem transformation methods provided by the MEKA (Read, Reutemann, et al., 2016) project, using decision trees as base learners, and relies on CLUS[5] for the implementation of PCT-based baselines. For all experiments, the default configurations of the different algorithms were used. This also applies to the BOOMER algorithm, which was configured to learn 1000 rules and use default values for shrinkage ($\eta = 0.3$) and L_2 regularization ($\delta = 1.0$). Each of the rules was constructed on a subset of the training examples, drawn with replacement. The refinements of a rule were restricted to a random subset of the available attributes. Whereas single-label rules were induced by the BOOMER-l.w.-log. approach, complete rules were used by the BOOMER-ex.w.-log. method. In addition, the latter also made use of gradient-based label binning (cf. Chapter 8).

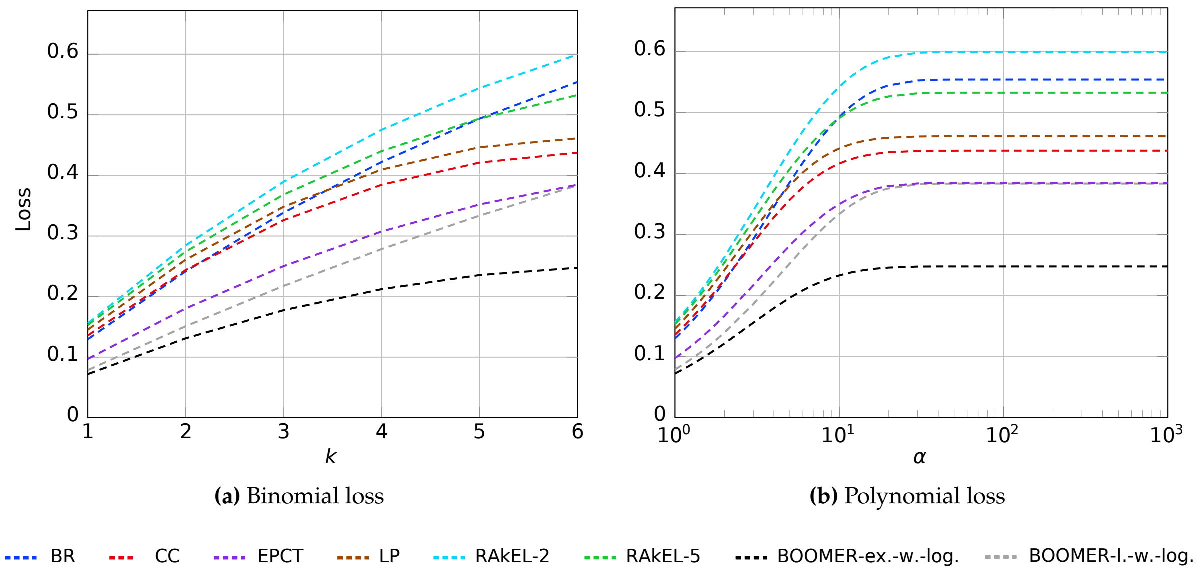
3: The experiments in this section are based on version 0.4.0 of BOOMER, available at `https://github.com/mrapp-ke/Boomer/releases/tag/0.4.0`

4: `https://github.com/KIuML/DependenceAwareMLCLoss`

5: `https://dtai.cs.kuleuven.be/clus`

Analysis on a Single Dataset

All considered MLC methods were evaluated in terms of different instantiations of the binomial and polynomial loss. Whereas the parameter k of the former was set to values in $\{1, \dots, K\}$, the parameter α of the latter was varied between 1 and 1000. In both cases, the lowest parameter value 1 corresponds to the Hamming loss and the highest value to the subset 0/1 loss. Intermediate values interpolate between these two extremes. To illustrate the properties of the dependence-aware loss functions, we start our analysis with a comparison of the evaluated algorithms on the dataset "Scene". Figure 6.3 shows the value of the parameters k and α on the x-axis. The y-axis indicates the loss that is achieved by the different methods. Naturally, the curves are monotonically increasing because the losses become more demanding and difficult to minimize with an increasing dependence-awareness. The intersections between two curves are particularly interesting, as they indicate that the ranking of the respective approaches has changed. On close examination, we can observe that some algorithms tend to work better than their



(a) Binomial loss

(b) Polynomial loss

Figure 6.3: Comparison of different multi-label classification methods with respect to parameterized instantiations of the polynomial loss and the binomial loss on the dataset "Scene".

competitors for small k or α. However, the ranking of the different approaches may change as the values of these parameters increase, and the losses demand greater dependence-awareness. For example, we can see that BR performs favorably to LP for small k or α, but LP catches up with increasing parameter values until it finally outperforms BR. In general, the dependence-awareness of a method is reflected by the slope of the corresponding curve — the flatter the curve, the better the method's dependence-awareness. For example, even though the approaches RAkEL-2 and RAkEL-5 are competitive for small parameter values, the latter has greater dependence-awareness and therefore outperforms its counterpart for larger values. Similar observations can be made for the BOOMER algorithm, where the variant BOOMER-ex.w.-log. outperforms BOOMER-l.w.-log. for large parameter values.

Comparison of BOOMER Approaches

For a more focused comparison between two methods over several datasets, Hüllermeier, Wever, et al. (2022) suggest using a different visualization of the experimental results than in Figure 6.3. For each dataset, they plot the parameter k or α of a dependence-aware loss function (on the x-axis) against the ratio of the losses achieved by two learners (on the y-axis). A point above the horizontal $y = 1$ indicates better performance of the first method, whereas a point below the horizontal indicates that the second method performs better. Thus, specifically interesting are the intersections of curves with the horizontal. Additional information is provided by the derivative of individual curves. A monotonically decreasing (increasing) shape indicates better dependence-awareness of the first (second) method, as it improves relative to the second (first) method with an increasing demand for dependence-awareness. In contrast, a parallel trajectory indicates a similar behavior regarding dependence-awareness. A visualization of this kind is shown in Figure 6.4, where we compare the dependence-awareness of the different BOOMER approaches to each other. As the parameter α of the polynomial loss allows for a more fine-grained analysis of dependence-awareness than the k parameter of the binomial loss, we restrict ourselves to the former for the pair-wise comparison of competing approaches.

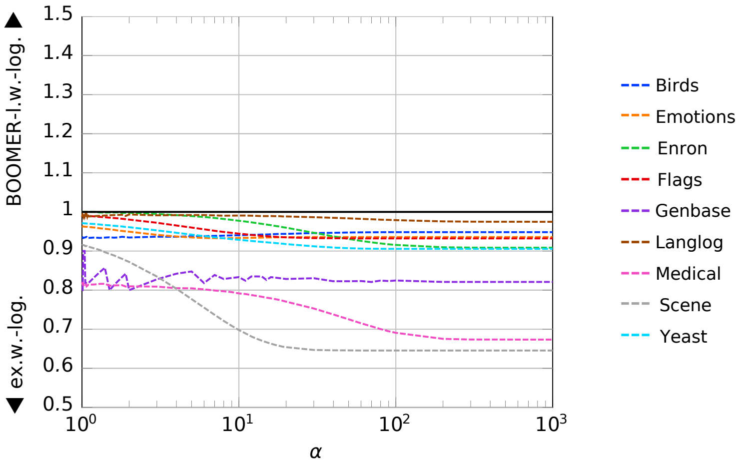


Figure 6.4: Comparison of the BOOMER-l.w.-log. and BOOMER-ex.w.-log. approaches in terms of parameterized instantiations of the polynomial loss on different datasets. The value of the α parameter is shown on the x-axis. The y-axis corresponds to the ratio between the losses that are achieved by the former and the latter approach. Points above the horizontal $y = 1$ indicate better performance of the BOOMER-l.w.-log. method. Points below the horizontal line indicate that BOOMER-ex.w.-log. performs better.

Figure 6.4 shows that the method BOOMER-ex.w.-log., which aims to optimize a non-decomposable surrogate of the subset 0/1 loss, achieves strong results and tends to outperform the BOOMER-l.w.-log. approach, which uses a surrogate for the Hamming loss. In accordance with our previous findings, it performs surprisingly strongly for small values of the parameter α (all points are below the horizontal). This advantage usually increases for instantiations of the polynomial loss that demand greater dependence-awareness (indicated by monotonically decreasing curves). However, the shape of the individual curves in Figure 6.4 depends not only on the models being compared but also on the dataset. This is expected because the performance of a method differs from dataset to dataset, just like the label-dependence. Thus, while dependence-awareness might be an advantage for one dataset, it could be a disadvantage for another one. For example, in the complete absence of label dependencies, an approach that deals with labels in an isolated manner can be very effective, whereas a method that deals with combinations of labels is unnecessarily complicated.

Comparison with Baselines

Figure 6.4 compares the BOOMER-l.w.-log. approach to the BR, LP, RAkEL, CC, and EPCT baselines. For small values of the parameter α, it tends to outperform its competitors (indicated by points above the horizontal), even though it performs worse than RAkEL-5, CC, and EPCT on the dataset "Medical". For increasing values of α, the differences between BOOMER-l.w.-log. and methods capable of modeling label dependencies, i.e., LP, RAkEL, CC, and EPCT, tend to become smaller (indicated by monotonically decreasing curves). This suggests that the use of a decomposable loss function and the restriction to single-label rules results in a lack of dependence-awareness. This is also emphasized by the results in Figure 6.6, where the BOOMER-ex.w.-log. method is compared to the different baselines. Unlike the BOOMER-l.w.-log. approach, it can generally outperform its competitors, regardless of the parameter α. Its strong performance with regard to parameter settings that demand great dependence-awareness suggests that the use of a non-decomposable loss function and the ability to use multi-label heads benefit the dependence-awareness of the BOOMER algorithm.

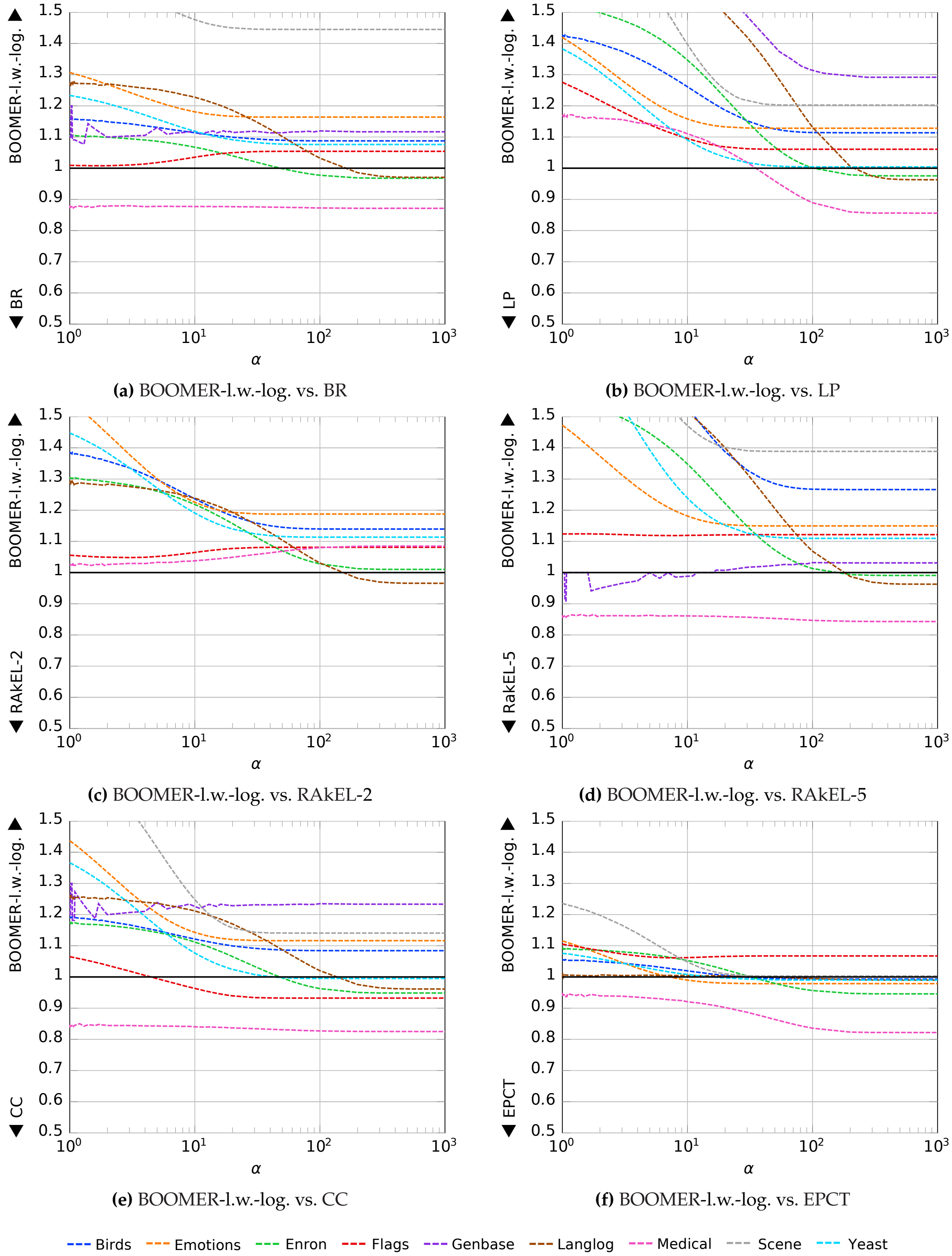
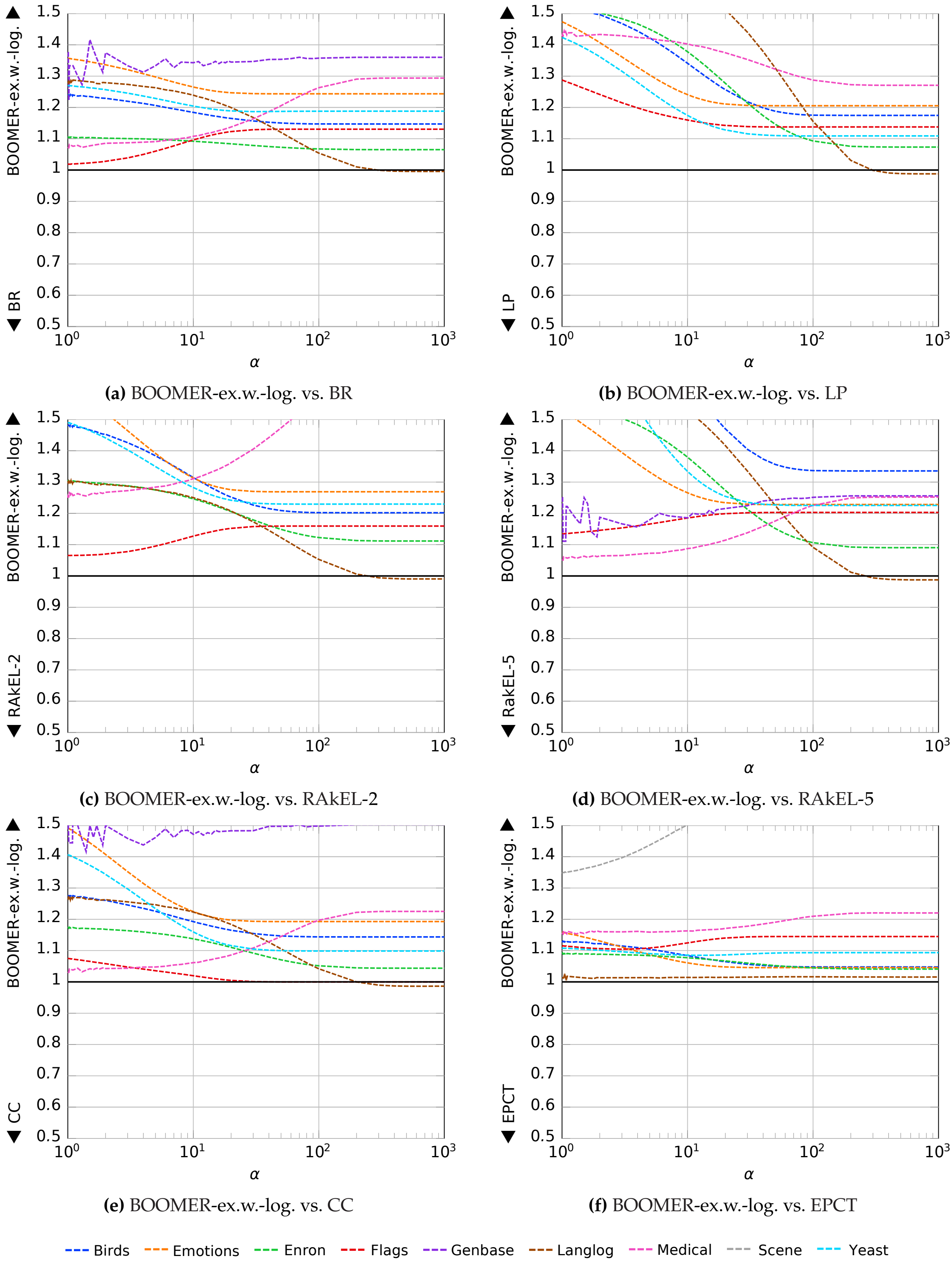


Figure 6.5: Pair-wise comparison of the BOOMER-l.w.-log. approach and competing MLC methods in terms of parameterized instantiations of the polynomial loss on different datasets. The value of the α parameter is shown on the x-axis. The y-axis corresponds to the ratio between the losses that are achieved by two methods. Points above the horizontal $y = 1$ indicate better performance of the BOOMER method. Points below the horizontal line indicate that the respective competitor performs better.



(a) BOOMER-ex.w.-log. vs. BR

(b) BOOMER-ex.w.-log. vs. LP

(c) BOOMER-ex.w.-log. vs. RAkEL-2

(d) BOOMER-ex.w.-log. vs. RAkEL-5

(e) BOOMER-ex.w.-log. vs. CC

(f) BOOMER-ex.w.-log. vs. EPCT

Figure 6.6: Pair-wise comparison of the BOOMER-ex.w.-log. approach and competing MLC methods in terms of parameterized instantiations of the polynomial loss on different datasets. The value of the α parameter is shown on the x-axis. The y-axis corresponds to the ratio between the losses that are achieved by two methods. Points above the horizontal $y = 1$ indicate better performance of the BOOMER method. Points below the horizontal line indicate that the respective competitor performs better.

6.7 Discussion

Our experiments on synthetic and real-world data confirm that the proposed algorithm for learning ensembles of single- or multi-label classification rules can successfully target different losses. It is able to outperform conventional state-of-the-art boosting methods, as well as tree-based problem transformation and adaptation approaches, on datasets of moderate size. Moreover, an analysis of the algorithm's dependence-awareness suggests that the ability to minimize non-decomposable surrogate loss functions, as well as the use of multi-label rules that allow to capture local label dependencies, is crucial for the effective optimization of non-decomposable evaluation measures like the subset 0/1 loss. The main drawback of the proposed method, as used in this first series of experiments, are the computational demands that come with large datasets. In particular, the minimization of non-decomposable loss functions comes with high computational costs due to the need to solve a linear equation system for calculating the predictions of potential candidate rules. To compensate for this, in Chapter 7, we discuss various algorithmic enhancements to the basic methodology introduced in this chapter. In Chapter 8, we further introduce an approximation technique that aims to address the computational bottleneck of the algorithm, which is the need to solve a linear system in the non-decomposable case. Additional enhancements, such as the possibility to exploit sparsity in the label space, are left for future work. We elaborate on these ideas in Section 9.2.

Efficient Induction of Rules | 7

In Section 4.3, an overview of the different components that are involved in the induction of individual rules is provided. Most importantly, this includes components for the enumeration of candidate bodies and the construction of corresponding rule heads. This chapter focuses on the former and discusses an algorithm for the efficient conduction of a greedy top-down search, as used by the BOOMER algorithm previously introduced in Chapter 6. The methodology discussed in this chapter can also be applied to SeCo-based algorithms, as discussed in Chapter 5. Compared to the naive implementation in Algorithm 8, computational efficiency may vastly benefit from implementing a "pre-sorted" search algorithm. It is based on the idea to avoid redundant computations by evaluating the possible candidate rules in a predetermined order. This eliminates the need to identify the examples covered by different candidates and aggregate the corresponding label space statistics for each of them individually. Implementations of rule learning methods that employ a pre-sorted search algorithm, e.g., *JRip*, an implementation of RIPPER (Cohen, 1995) that is part of the WEKA (Hall et al., 2009) software package, can be found to drastically outperform rule learners that do not come with such an optimization. This is illustrated by Table 7.1, which compares the training times of JRip and *Wittgenstein*[1] , another implementation of RIPPER that does not rely on a predetermined order for the evaluation of candidate rules. In the following, we introduce the basic structure of the pre-sorted search algorithm used by the BOOMER algorithm for the efficient induction of rules. We further discuss how sparsity in the feature space, which is characteristic of many multi-label datasets (cf. Section 3.2), can be exploited to further reduce computational costs and describe how the algorithm deals with nominal attributes and missing feature values. In addition, we discuss a histogram-based search algorithm for the efficient induction of rules in domains with many numerical features, where the ability to exploit sparsity in the feature space does not provide any significant advantages in terms of scalability. Finally, we elaborate on the BOOMER algorithm's ability to use parallelization to further speed up the training of predictive models.

1: The training times in Table 7.1 result from version 0.2.3 of the Wittgenstein algorithm, available at `https://github.com/imoscovitz/wittgenstein`.

Table 7.1: Training times (in seconds) of JRIP and Wittgenstein, averaged across 10 cross validation folds, when applied to different multi-label datasets using the binary relevance transformation approach (cf. Section 3.6).

Dataset	JRip	Wittgenstein
Birds	10.0	877.3
CAL500	28.5	776.0
Emotions	1.6	13.6
Flags	0.6	1.3
Image	13.9	820.2
Ohsumed	771.4	2820.9
Scene	16.1	605.1
Slashdot	371.3	11424.2
Yeast	19.7	521.6

7.1 Pre-Sorted Search Algorithm

The idea of using a pre-sorted algorithm goes back to early work on the efficient construction of decision trees (Mehta, Rakesh Agrawal, and Rissanen, 1996; Shafer, Rakesh Agrawal, and Mehta, 1996). Due to the conceptual similarities between tree- and rule-based models, it can easily be generalized to rule learning methods. Both learning approaches require enumerating the thresholds that may be used to make up nodes in a decision tree or conditions in a rule, respectively. These thresholds result from the feature values of the training examples, given in the form of a feature matrix $X \in \mathbb{R}^{N \times L}$. For each training example $\boldsymbol{x}_n$, it assigns a feature value x_{nl} to each of the available attributes A_l. In this section, we restrict ourselves to numerical attributes. Means to deal with nominal attributes or missing feature values are discussed in the sections "Nominal Attributes" and "Missing Feature Values" below. When searching for the best condition that may be added to a rule, the available attributes are dealt with independently. For each attribute A_l to be considered by the algorithm, the thresholds that may be used by the first condition of a rule result from a vector of feature values $(x_{1l}, \dots, x_{nl})$ that corresponds to the l-th column of the feature matrix.[2] As different attributes are dealt with in isolation, we omit the index of the respective attribute for brevity. To enumerate the thresholds for a particular attribute, the elements in the corresponding column vector must be sorted in increasing order. For this purpose, we use a bijective permutation function $\tau : \mathbb{N}^+ \rightarrow \mathbb{N}^+$, where $\tau(i)$ specifies the index of the example that corresponds to the i-th element in the sorted vector

2: To facilitate column-wise access to the feature matrix, it should be given in the *Fortran-contiguous* memory layout (see Figure 7.4 for an example).

$$\left(x_{\tau(1)}, \dots, x_{\tau(N)}\right) \text{ with } \tau(i) \leq \tau(i+1), \forall i \in [1, N). \tag{7.1}$$

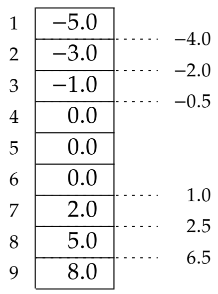


Figure 7.1: Exemplary vector of numerical feature values for a particular attribute, sorted in increasing order. The thresholds that result from averaging two adjacent feature values are shown to the right.

An exemplary vector of sorted feature values, together with the corresponding thresholds, is shown in Figure 7.1. Each of the thresholds is computed by averaging two adjacent feature values. Because these values do not change as additional conditions or rules should be learned, sorting the values that correspond to a particular attribute is necessary only once during training, and previously sorted vectors can be kept in memory for repeated access. If an existing rule should be refined by adding a condition to its body, only a subset of the feature values must be considered to make up potential thresholds. The subset corresponds to the examples that satisfy the existing conditions. We use an indicator function $\mathbb{1}_X : \mathbb{N}^+ \rightarrow \{0, 1\}$ to check whether individual examples should be taken into account by the search algorithm:

$$\mathbb{1}_X(n) = \begin{cases} 1 & \text{if example } \boldsymbol{x}_n \text{ is currently covered} \\ 0 & \text{otherwise.} \end{cases} \tag{7.2}$$

If an example is not covered, its feature value may not be used to make up thresholds for additional conditions. A data structure that helps to keep track of the covered examples efficiently, rather than comparing the feature values of each example to the existing conditions, is presented in the following section "Exploiting Feature Sparsity". It also facilitates dealing with sparse feature values.

Enumeration of Conditions

As shown in Algorithm 9, the feature values that correspond to a particular attribute are processed in sorted order to enumerate the thresholds of potential conditions. When dealing with numerical attributes, the thresholds result from averaging adjacent feature values (cf. Algorithm 9, line 9). The calculation of thresholds is restricted to the feature values of examples that are covered according to the indicator function $\mathbb{1}_X$ and have non-zero weights according to a weight vector $\boldsymbol{w}$. The weights result applying an (optional) instance sampling method, as described in Section 4.3 (cf. Algorithm 9, line 3 and line 7). When dealing with numerical attributes, each threshold can be used to form two distinct conditions, using the relational operator $\leq$ or $>$, respectively. As can be seen in Figure 7.2, when a condition that uses the former operator is added to a rule, it results in all examples that correspond to the previously processed feature values being covered. In contrast, a condition that uses the latter operator covers all of the other examples.

Aggregation of Statistics

As previously discussed in Section 6.3 and illustrated in Algorithm 7, it is necessary to construct a head for each candidate rule that results from adding a new condition to a rule. In addition, the quality of the resulting rule must be assessed in terms of a numerical score. Both the predictions provided by a head and the estimated quality depend on the aggregated label space statistics of the covered examples. We exploit the fact that conditions using the $\leq$ operator, when evaluated in sorted order by increasing thresholds, are satisfied by a superset of the examples covered by the previous condition using the same operator but a smaller threshold. Processing the possible conditions in the aforementioned order enables the pre-sorted search algorithm to compute the aggregated statistics ($\boldsymbol{g}'$ and $\boldsymbol{H}'$) incrementally (cf. Algorithm 9, line 8). For the efficient evaluation of conditions that use the $>$ operator, the search algorithm is provided with the statistics that result from the aggregation over all training examples that are currently covered and have a non-zero weight ($\boldsymbol{g}$ and $\boldsymbol{H}$). The difference between the globally aggregated statistics and the previously aggregated ones ($\boldsymbol{g} - \boldsymbol{g}'$ and $\boldsymbol{H} - \boldsymbol{H}'$) yields the aggregated statistics of the examples covered by such a condition (cf. Algorithm 9, line 13). As the global aggregation of statistics does not depend on a particular attribute, this operation must be performed only once, even when searching for a rule's best refinement across multiple attributes. Figure 7.3 provides an example of how the aggregated statistics are computed for the conditions that can be created from a single threshold.

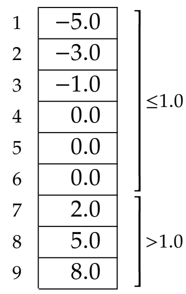


Figure 7.2: Coverage of numerical conditions that can be created from a single threshold using the $\leq$ or $>$ operator.

Algorithm 9: Pre-sorted algorithm for the evaluation of refinements

input : Vector of sorted feature attributes $\left(x_{\tau(n)}\right)_n^N$, quality of the current rule q, indicator function $\mathbb{1}_X$, weights of training examples $\boldsymbol{w}$, statistics $S = \{(\boldsymbol{g}_n, \boldsymbol{H}_n)\}_n^N$, globally aggregated statistics $\boldsymbol{g} = \sum_n (\mathbb{1}_X(n)\, w_n \boldsymbol{g}_n)$ and $\boldsymbol{H} = \sum_n (\mathbb{1}_X(n)\, w_n \boldsymbol{H}_n)$

output: Best refinement r^*, quality of the refinement q^*

1 best refinement $r^* = \emptyset$, best quality $q^* = q$
2 **for** $i = 1$ **to** N **do**
3   **if** $\mathbb{1}_X(\tau(i)) = 1$ **and** $w_{\tau(i)} > 0$ **then**
4   **break**
5 initialize sum of gradients $\boldsymbol{g}' = \boldsymbol{g}_{\tau(i)}$ and Hessians $\boldsymbol{H}' = \boldsymbol{H}_{\tau(i)}$
6 **for** $j = i + 1$ **to** N **do**
7   **if** $\mathbb{1}_X(\tau(j)) = 1$ **and** $w_{\tau(j)} > 0$ **then**
8   update sum $\boldsymbol{g}' = \boldsymbol{g}' + \boldsymbol{g}_{\tau(j)}$ and $\boldsymbol{H}' = \boldsymbol{H}' + \boldsymbol{H}_{\tau(j)}$
9   threshold $\mu = \text{avg}\left(x_{\tau(i)}, x_{\tau(j)}\right)$
10   updated head $\hat{\boldsymbol{p}}'$, quality $q' = \text{FIND\_HEAD}(\boldsymbol{g}', \boldsymbol{H}')$
11   **if** $q' < q^*$ **then**
12   update refinement $r^* = \{t, \leq, \hat{\boldsymbol{p}}'\}$ and its quality $q^* = q'$
13         $\hat{\boldsymbol{p}}', q' = \text{FIND\_HEAD}(\boldsymbol{g} - \boldsymbol{g}', \boldsymbol{H} - \boldsymbol{H}')$
14   **if** $q' < q^*$ **then**
15   update refinement $r^* = \{t, >, \hat{\boldsymbol{p}}'\}$ and its quality $q^* = q'$
16         $i = j$
17 **return** best refinement r^*, its quality q^*

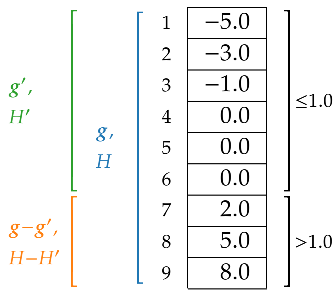


Figure 7.3: Aggregation of statistics depending on the coverage of conditions that use the $\leq$ or $>$ operator and a single threshold. In case of the $>$ operator, the statistics are obtained by computing the difference (orange) between previously aggregated statistics (green), which correspond to already processed feature values, and globally aggregated statistics that are computed beforehand (blue).

Exploiting Feature Sparsity

As shown in Table 3.2, high feature sparsity is a common property of multi-label datasets, especially those used in text classification. When dealing with training data where most feature values are equal to zero, using a sparse representation of the feature matrix reduces the amount of memory required for storing its elements and facilitates the implementation of algorithms that can deal with such data in a computationally efficient way. In the following, we discuss a variant of the pre-sorted search algorithm introduced above that allows to search for potential refinements of rules using both dense and sparse feature matrices. As shown experimentally, the use of sparse input data, where feature values are provided in the *compressed sparse column* (CSC) format (see Figure 7.4 for an example), may drastically reduce training times.

3×3 matrix:

3	4	2
0	0	6
1	0	0

Fortran-contiguous:

3	0	1	4	0	0	2	6	0

CSC format:

values:	3	1	4	2	6
row_indices:	0	2	0	0	1
column_indices:	0	2	3		

Figure 7.4: Representation of a 3×3 matrix in the Fortran-contiguous and compressed sparse column (CSC) format. The former uses a single one-dimensional array to store all values in column-wise order, whereas the latter uses the following three arrays: (1) The array *values* stores all non-zero values in column-wise order. (2) For each value in *values*, *row_indices* stores the index of the corresponding row, starting at zero. (3) The i-th element in *column_indices* specifies the index of the first element in *values* and *row_indices* that belongs to the i-th column.

> The BOOMER algorithm allows to specify the preferred format of the feature matrix via the following parameter:
>
> ```
> --feature-format [auto,dense,sparse]
> ```
>
> If set to `auto` (the default value), the most suitable format is chosen automatically, depending on which representation is expected to require less memory.

When dealing with a sparse representation of the feature matrix, the sorted column vector for each attribute, which serves as a basis for enumerating possible thresholds, only contains non-zero feature values. An example is given in Figure 7.5. On the one hand, because only non-zero values must be processed, this reduces the algorithm's computational complexity. However, on the other hand, the algorithm cannot identify the examples with zero feature values.

Enumeration of Thresholds

To enumerate all thresholds that result from a sparse vector, including those that result from examples with zero feature values, a "sparsity-aware" search algorithm must follow three phases:

I. It starts by processing the sorted feature values in increasing order as before. Traversal of the feature values must be stopped as soon as a positive value or a value equal to zero is encountered.

II. Afterward, it processes the sorted feature values in decreasing order until a negative value or a value equal to zero is encountered.

III. After all elements in a given vector have been processed, it is possible to deal with the thresholds that eventually result from examples with zero feature values. To determine whether such examples exist, the number of elements with non-zero weights processed so far can be compared to the total number of examples in a dataset or a sample thereof. If all available examples have already been processed, a single threshold can be formed by averaging the largest negative feature value and the smallest positive one that has been encountered in the previous phases. Otherwise, two thresholds between zero and each of these values can be made up.

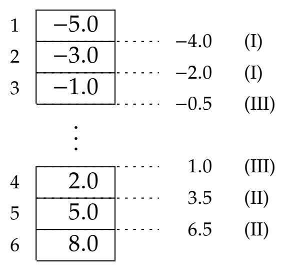


Figure 7.5: Sparse representation of the vector of numerical features in Figure 7.1, where values that are equal to zero are omitted. The thresholds that result from averaging two adjacent feature values are shown to the right. The numbers in parentheses (I, II, III) specify the phases of the sparsity-aware search algorithm that are responsible for the evaluation of individual thresholds.

An algorithm that follows the aforementioned procedure is not only able to deal with dense and sparse vector representations, but also enumerates all thresholds that are considered by Algorithm 9. However, if a large fraction of the feature values are equal to zero, it involves far less computational steps.

Aggregation of Statistics (Phase I)

The first phase of the sparsity-aware search algorithm, where examples with negative feature values are processed, is identical to Algorithm 9. During this initial phase, the aggregated statistics of examples that satisfy potential conditions are obtained as illustrated in Figure 7.3. If a condition uses the $\leq$ operator, the statistics of the examples it covers correspond to the previously aggregated statistics (g' and H') of already processed examples. To obtain the statistics of examples covered by a condition using the $>$ operator, the difference ($g - g'$ and $H - H'$) between the globally aggregated statistics (g, H) and the previously aggregated ones are computed. The first phase ends as soon as an example with a positive or zero feature value is encountered. The statistics that have been aggregated until this point (g' and H') include all examples with negative feature values and must be retained for later use during the third phase of the algorithm.

Aggregation of Statistics (Phase II)

The second phase, where examples with positive feature values are considered, follows the same principles as the previous phase. However, the examples are processed in decreasing order of their corresponding feature values. Consequently, the incrementally aggregated statistics (denoted as g'' and H'' to distinguish them from the variables used in the first phase) updated at each step correspond to the examples that cover a condition using the $>$ operator. To obtain the aggregated statistics of examples that satisfy a condition using the $\leq$ operator, the difference ($g - g''$ and $H - H''$) between the globally aggregated statistics and the previously aggregated ones must be computed. The end of the second phase is reached as soon as an example with a negative or zero feature value is encountered. The statistics aggregated during this phase (g'' and H'') include the statistics of all examples with positive feature values. They are kept in memory for use during the third and final phase.

Aggregation of Statistics (Phase III)

After the second phase has finished, the algorithm is able to decide whether any examples with zero feature values, which are neither stored by a sparse representation of the feature values nor can explicitly be accessed by the rule induction algorithm, are available. This is the case if the sum of the weights of all examples processed until this point is smaller than the total sum of weights of all examples in a dataset or a sample thereof. In any case, it is necessary to evaluate potential conditions that separate the examples with positive feature values from the remaining ones (possibly including examples with zero feature values). As shown in Figure 7.6, this requires considering two conditions with the operator $\leq$ and $>$, respectively. The statistics of examples that satisfy the latter correspond to the statistics aggregated during the algorithm's second phase (g'' and H''). To obtain the statistics of examples that are covered by the former, the difference ($g - g''$ and $H - H''$) between the statistics aggregated during the second phase and the globally aggregated ones must be computed. In addition, if any examples with zero feature values are available, additional conditions using the operators $\leq$ and $>$ that separate the examples with negative feature values from the remaining ones must be considered. The statistics of examples that are covered by the former correspond to the statistics that have been aggregated during the first phase of the algorithm (g' and H''). In contrast, the

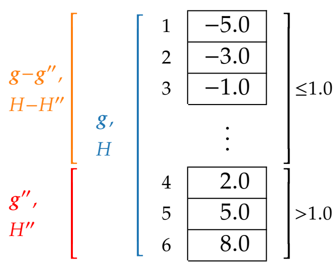


Figure 7.6: Evaluation of conditions that separate examples with positive feature values from the remaining ones. The condition that uses the $\leq$ operator requires to compute the difference (orange) between the statistics of examples with positive feature values (red) and the globally aggregated ones (blue).

statistics of examples that satisfy the latter must again be computed by taking the globally aggregated statistics into account. They calculate as the difference ($g - g'$ and $H - H'$) between the statistics corresponding to examples with negative feature values, which have been processed during the first phase of the algorithm, and the globally aggregated ones that have been computed beforehand. An example of how the aggregated statistics for the evaluation of these conditions are obtained is given in Figure 7.7. If no examples with zero feature values are available, the evaluation of additional conditions, as depicted in Figure 7.7, can be omitted.

Keeping Track of Covered Examples

Once the best condition among all available candidates has been found and has been added to a rule, it is necessary to keep track of the examples that are covered by the modified rule. This is crucial because additional conditions that may be added during later refinement iterations must only be created from the feature values of examples that satisfy the existing conditions. When dealing with a dense representation of feature values, as shown in Figure 7.1, the feature values of all examples can directly be accessed. Keeping track of the covered examples is straightforward in this case. However, given a sparse representation, as shown in Figure 7.5, it becomes a non-trivial task since the algorithm does not know which examples come with zero feature values. To overcome this problem, the BOOMER algorithm employs a data structure suited to keep track of the covered examples in both cases, regardless of the feature representation used. It maintains a vector that stores a value for each example in a dataset, as well as an *indicator value*. If the value that corresponds to a certain example is equal to the indicator value, it is considered to be covered. This enables to answer queries to the indicator function $\mathbb{1}_X(n)$, as defined in (7.2), in constant time by comparing the value of the n-th example to the indicator value. Initially, when a new rule does not contain any conditions yet, the indicator value and the values in the vector are all set to zero, i.e., all examples are considered to be covered by the rule. An example of such a data structure for nine examples that correspond to the feature values in Figure 7.5 is shown in Figure 7.8. In order to update the data structure after a new condition has been found, the range of examples it covers must be taken into account. If only examples with negative (or positive) feature values satisfy a condition, i.e., if the condition's threshold is less than (greater than or equal to) zero and it uses

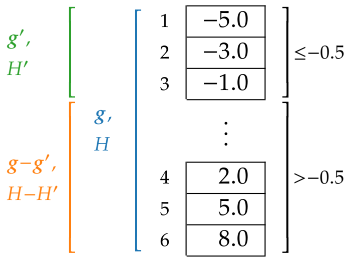


Figure 7.7: Evaluation of conditions that separate examples with negative feature values from the remaining ones. The difference (orange) between the statistics of examples with negative feature values (green) and the globally aggregated ones (blue) is required in case of the ≤ operator.

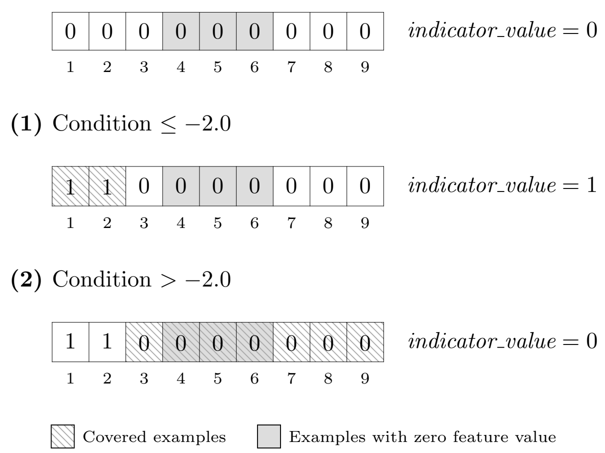


Figure 7.8: Visualization of the data structure that is used to keep track of the examples that are covered by a rule. For each example, it stores a value that indicates whether the example is covered or not. An example is considered to be covered if its value is equal to an *indicator_value*. Initially, all examples are marked as covered (top). When a new condition is added to the rule, the data structure is updated by following one of the following strategies: (1) If the examples that satisfy the condition do not have zero feature values, the corresponding elements and the *indicator_value* are both set to the total number of conditions. (2) Otherwise, the elements that correspond to uncovered examples are updated, whereas the *indicator_value* remains unchanged.

the ≤ (>) operator, the values that are maintained by the data structure to specify whether the respective examples are covered can be updated directly. In such a case, the values of covered examples and the indicator value are set to the number of conditions that are currently contained in the rule's body, marking them as covered. If a condition is satisfied by examples with zero feature values, for which the corresponding indices are unknown, the values that correspond to the uncovered examples are updated instead by setting them to the current number of conditions. However, the indicator value remains unchanged, which renders the examples that correspond to the updated values uncovered, whereas examples with unmodified values remain covered if they have already satisfied the previous conditions.

Nominal Attributes

One-Hot-Encoding

An advantage of rule learning algorithms is their ability to deal with nominal attributes by using operators like = or ≠ for the conditions in a rule's body. This is in contrast to many statistical machine learning methods, such as logistic regression, support vector machines, or neural networks, that cannot deal with nominal attributes directly. Instead, they require to apply preprocessing techniques to the data before training. Most commonly, *one-hot-encoding* is used to convert nominal attributes to numerical ones. It replaces a single nominal attribute with a fixed number of discrete values with several binary attributes that specify for each of the original values whether it is set to an example or not. Such a conversion may drastically increase the number of attributes in a dataset and therefore can negatively affect the complexity of a learning task.

ⓘ The BOOMER algorithm provides the following parameter to specify whether one-hot-encoding should be used to deal with nominal attributes:

```
--one-hot-encoding [true,false]
```

By default, the algorithm relies on its native support for this particular type of attributes.

Enumeration of Thresholds

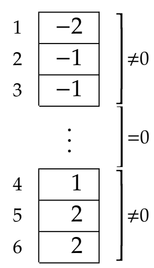


Figure 7.9: Coverage of nominal conditions that can be created from a single threshold using the = or ≠ operator.

To handle nominal attributes, the BOOMER algorithm relies on the same principles used by its pre-sorted search algorithm to deal with numerical attributes, including the ability to use sparse representations of feature values. In the case of a nominal attribute, the feature values associated with the individual training examples are not arbitrary real numbers but are limited to a predefined set of discrete values that do not necessarily correspond to a continuous range and possibly include negative values. As a result, the thresholds that potential conditions may use are not formed by averaging adjacent feature values but correspond to the discrete values associated with the available training examples. Two conditions must be evaluated for each of the values encountered by the algorithm in a sorted vector of feature values. As shown in Figure 7.9, they use the operator = and ≠, respectively. Whereas a condition that uses the former operator covers neighboring examples with the same value, the examples that satisfy a condition with the latter operator do not correspond to a continuous range in a sorted vector of feature values. This requires adjustments to the algorithm when it comes to aggregating statistics that correspond to examples that are covered by nominal conditions.

Aggregation of Statistics (Phase I and II)

The algorithm follows the same order for processing the sorted feature values as outlined in the previous section "Exploiting Feature Sparsity" to facilitate the use of sparse feature representations when dealing with nominal attributes. At first, it processes the examples associated with negative feature values. Afterward, it evaluates the conditions that result from positive feature values, and finally, in a third phase, potential conditions with zero thresholds are considered. During the first and second phase, the statistics of examples with the same feature value are aggregated individually. In accordance with the notation in Figure 7.10, we denote the aggregated statistics for different feature values as $g', g'', \ldots$ and $H', H'', \ldots$. This is in contrast to the aggregation of statistics in the case of numerical attributes, where the statistics of all examples with negative and positive feature values are aggregated. As illustrated in Figure 7.10, the globally aggregated statistics (g and H) that are provided to the algorithm beforehand are used to obtain the statistics corresponding to examples that satisfy conditions using the $\neq$ operator. This requires to compute the difference between the globally aggregated statistics and the aggregated statistics of all examples associated with a particular discrete value.

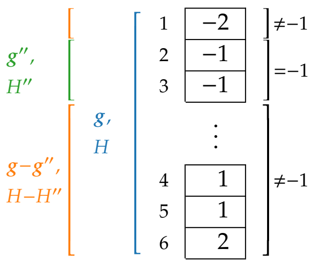


Figure 7.10: Evaluation of conditions that separate examples with a particular feature value from the remaining ones. The difference (orange) between the statistics of the covered examples (green) and the globally aggregated ones (blue) is used to evaluate conditions with the $\neq$ operator.

Aggregation of Statistics (Phase III)

During the third phase of the algorithm, special treatment is required to evaluate conditions with zero thresholds if any examples with zero feature values are available. To determine whether such examples exist, the sum of the weights of all examples that have previously been processed in the first and second phases of the algorithm is compared to the weights of all examples in a dataset or a sample thereof, as described earlier. To obtain the aggregated statistics that correspond to the examples with zero feature values, the statistics $g', g'', \ldots$ and $H', H'', \ldots$ that have been computed during the previous phases must be aggregated. We denote the resulting accumulated statistics as $g^* = g' + g'' + \ldots$ and $H^* = H' + H'' + \ldots$, respectively. As shown in Figure 7.11, they correspond to the examples with non-zero feature values covered by a condition that uses the $\neq$ operator. To evaluate a condition that uses the $=$ operator and covers all examples with zero feature values, inaccessible by the algorithm when using a sparse feature representation, the difference between the globally aggregated statistics (g and H) and the accumulated ones are computed.

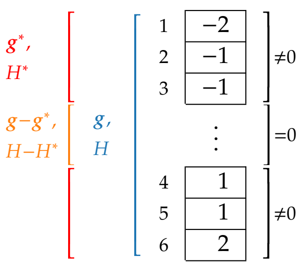


Figure 7.11: Evaluation of nominal conditions that separate examples with zero feature values from the remaining ones. The aggregated statistics of the former are computed as the difference (orange) between the globally aggregated statistics (blue) and the ones that correspond to all previously processed examples (red).

Missing Feature Values

Possible Strategies

Although none of the benchmark datasets introduced in Section 3.2 comes with missing feature values, the ability to deal with training data, where the feature values of individual examples are partly unknown, is a common requirement of many real-world classification problems. We therefore elaborate on how the BOOMER algorithm deals with unknown feature values in the following. Different strategies for handling missing values can be found in the rule learning literature (see, e.g., Wohlrab and Fürnkranz, 2011, for an overview). BOOMER ignores examples with missing values when evaluating the conditions that can be added to a rule's body with respect to a certain attribute. Consequently, rules that contain conditions on an attribute A_l in their body can never be satisfied by examples for which the feature value x_l is missing.[3] For example, this strategy is also used by RIPPER (Cohen, 1995).

Adjustments to the Algorithm

Only minor adjustments are necessary to apply the pre-sorted search algorithm, which is used by BOOMER for the evaluation of conditions, to an attribute for which some examples may lack a value. First of all, the examples that do not assign a value to the respective attribute are excluded from the sorted feature vector in (7.1) and therefore are ignored when enumerating the possible thresholds of conditions and aggregating the statistics of examples they cover. Instead, the algorithm keeps track of the examples that do not assign a value to a particular attribute in a separate data structure. When searching for possible conditions concerned with the respective attribute, the statistics of the examples that are known to lack a value for the attribute must be subtracted from the globally aggregated statistics (previously denoted as $\boldsymbol{g}$ and H). This ensures that the statistics that are aggregated while processing a vector of sorted feature values, as well as the statistics that are computed as the difference between previously aggregated statistics and the globally aggregated ones, do not include examples with missing values, the considered refinements of a rule cannot cover.

3: Other variants include the opposite strategy, i.e., conditions on missing feature values are always satisfied, as well as the possibility to learn conditions that explicitly test for unknown values. Alternatively, some learning algorithms come with strategies to impute the values that are unknown for an example or ignore examples with missing values entirely (Fürnkranz, Gamberger, and Lavrač, 2012).

Experimental Evaluation

Experimental Setup

To evaluate the efficiency of the pre-sorted search algorithm employed by BOOMER and, in particular, the advantages that result from using its sparsity-aware variant, we applied the algorithm to several benchmark datasets introduced in Section 3.2. For a complete picture, we included datasets with varying degrees of feature sparsity in the experimental study and considered both dense and sparse feature representations for the experiments. Unlike in Section 6.6, we did not use any parameter tuning for the experiments but relied on default parameters that have been found to work well on average in preliminary experiments.[4] To deal with nominal attributes that are included in some of the datasets, we relied on the BOOMER algorithm's native support for this kind

4: For all experiments that are discussed in this chapter we used version 0.6.2 of the BOOMER algorithm, which is available at `https://github.com/mrapp-ke/Boomer/releases/tag/0.6.2`.

Table 7.2: Average training times (in seconds; rounded to one decimal place) per cross validation fold on different datasets (the feature sparsity is given in parentheses). The small numbers indicate the speedup that results from using sparse feature representations, compared to dense feature representations. By default, BOOMER prefers a sparse representation for datasets that are marked with an asterisk.

Dataset		Dense	Sparse	
20NG*	(96.81)	9.7	**1.1**	8.82
Bibtex*	(96.26)	12.1	**1.9**	6.37
Birds	(38.64)	**0.6**	**0.6**	1.00
Bookmarks*	(94.16)	308.4	**44.0**	7.01
Delicious*	(96.34)	159.6	**20.0**	7.98
Emotions	(0.33)	**0.5**	**0.5**	1.00
Enron*	(91.60)	1.0	**0.2**	5.00
EukaryoteGO*	(99.86)	5.1	**0.4**	12.75
EukaryotePseAAC	(43.37)	6.5	**4.6**	1.41
EUR-Lex-SM*	(95.26)	70.9	**9.3**	7.62
Image	(0.22)	**2.3**	**2.3**	1.00
IMDB*	(98.06)	126.0	**12.2**	10.33
Langlog*	(81.38)	0.9	**0.3**	3.00

Dataset		Dense	Sparse	
Mediamill	(0.00)	**235.2**	235.5	1.00
Medical*	(99.32)	0.9	**0.2**	4.50
Ohsumed*	(96.03)	7.4	**0.9**	8.22
Reuters-K500*	(98.41)	4.5	**1.0**	4.50
Scene	(1.15)	3.2	**3.0**	1.07
Slashdot*	(99.46)	2.6	**0.2**	13.00
TMC2007*	(99.79)	33.9	**1.3**	26.08
Yahoo-Computers*	(99.62)	19.6	**0.9**	21.78
Yahoo-Reference*	(99.59)	9.9	**0.6**	16.50
Yahoo-Science*	(99.53)	8.8	**0.5**	17.60
Yahoo-Social*	(99.71)	17.3	**0.9**	19.22
Yeast	(0.00)	2.7	**2.6**	1.04

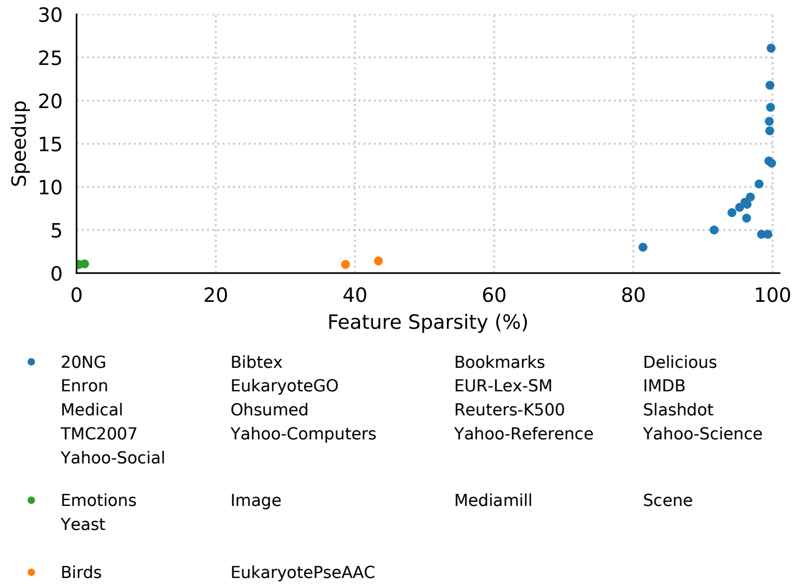


Figure 7.12: The average speedup in training time per cross validation fold that results from a search algorithm that is capable of exploiting sparsity in the feature space, compared to an algorithm that uses dense data structures. The considered datasets are grouped by small (green), medium (orange) and high (blue) feature sparsity.

of attribute rather than applying one-hot-encoding to the data. For each dataset, we trained models that consist of $T = 1000$ rules. The L_2 regularization weight was set to $\delta = 1.0$, whereas the shrinkage parameter was set to $\eta = 0.3$. The algorithm was configured to obtain a subset of $\lfloor \log_2 (L-1) + 1 \rfloor$ random attributes whenever a rule should be refined. To learn individual rules, all available training examples were used rather than employing a sampling method as in Section 6.6. We further restricted ourselves to the minimization of the label-wise logistic loss function in (6.15), which serves as a surrogate for the Hamming loss, using rules with single-label heads. The BOOMER algorithm's ability to utilize multiple computational threads was not used in the experiments. A discussion of different possibilities to speed up training via multi-threading, accompanied by performance benchmarks, can be found in Section 7.3 below.

Experimental Results

Table 7.2 provides an overview of the BOOMER algorithm's training times, averaged across the folds of a 10-fold cross validation, when applied to different datasets and using dense or sparse representations of feature values. In addition, it also shows the speedups in training time that result from the exploitation of sparsity in the feature space when using the latter type of feature representation. As expected, when applied to datasets with low feature sparsity ("Emotions", "Image", "Mediamill", "Scene" and "Yeast"), the training times remain mostly unaffected by using sparse data structures. A minor speedup can be observed on datasets with a feature sparsity between 30 and 50% ("Birds", "EukaryotePseAAC"). On the remaining datasets, where at least 80% of the feature values are equal to zero, the sparsity-aware rule induction algorithm clearly outperforms the baseline and significantly reduces training time. This is also illustrated by the graphical representation of the experimental results in Figure 7.12, where the speed up that results from using a sparse feature representation is related to the feature sparsity of the considered datasets.

7.2 Histogram-based Search

Limitations of the Sparsity-aware Algorithm

The exploitation of feature sparsity, as discussed in the previous section, helps reduce training times on many benchmark datasets used in this work, as they typically come with high feature sparsity. However, it does not provide significant advantages on datasets with low feature sparsity. The BOOMER algorithm provides an alternative to the pre-sorted search algorithm introduced in Section 7.1 to efficiently deal with the latter type of datasets. It is based on assigning examples with similar feature values for a particular attribute to a predefined number of bins and using an aggregated representation of their corresponding label space statistics, referred to as *histograms*. Depending on how many bins are used, this approach drastically reduces the number of candidate rules the rule induction algorithm must consider. Histogram-based approaches have previosuly been used to deal with complex single-label classification tasks in modern implementations of gradient-boosted decision trees, such as XGBoost (T. Chen and Guestrin, 2016) or LightGBM (Ke et al., 2017). In the following, we discuss a generalization of the underlying concept, which has evolved from prior research on decision tree learning (see, e.g., Alsabti, Ranka, and V. Singh, 1998; Jin and G. Agrawal, 2003; Kamath, Cantú-Paz, and Littau, 2002; P. Li, Qiang Wu, and Burges, 2007), to rule learning methods and investigate its impact on predictive performance and training efficiency in an empirical study.

ⓘ The BOOMER algorithm allows to use histogram-based rule induction by specifying the following parameter:

```
--feature-binning [equal-width,equal-frequency]
```

Information on how to configure the number of bins can be found in the documentation.

Assigning Examples to Bins

Feature Binning Methods

The histogram-based rule induction algorithm requires grouping the available training examples into a predefined number of bins. Different approaches can principally be used to determine such a mapping (see, e.g., Kotsiantis and Kanellopoulos, 2006, for a survey on existing techniques). We restrict ourselves to unsupervised binning methods, where the assignment is solely based on the feature values of the training examples. This is in contrast to supervised methods, such as the *weighted quantile sketch* approach that originates from the XGBoost classification system (T. Chen and Guestrin, 2016), where information about the ground truth labels of individual examples, or even their label space statistics, are taken into account. Compared to approaches that utilize the label space statistics to map from examples to bins, unsupervised binning methods can usually be implemented more efficiently. This is because a mapping solely based on feature values remains unchanged for the entire training process, whereas the statistics for individual examples are subject to change and require adjusting the mapping whenever a model is refined.

Equal-width Feature Binning

The first binning method that is supported by the BOOMER algorithm is referred to as *equal-width binning*. This method, which is commonly used to discretize numerical feature values, is based on dividing the range of values for a particular attribute into equally-sized intervals, such that the absolute difference between the smallest and largest value in each bin are the same. Given a predefined number of bins M, the maximum difference between the values that are assigned to an interval calculates as

$$\omega = \frac{\max - \min}{M}, \tag{7.3}$$

where min and max denote the largest and smallest value in a bin, respectively. Based on the value ω, a mapping $\sigma : \mathbb{R} \rightarrow \mathbb{N}^+$ from individual feature values x_n to the index of the corresponding bin can be obtained as

$$\sigma_{eq.\text{-}width}(x_n) = \min\left(\lfloor \frac{x_n - \min}{\omega} \rfloor + 1, M\right). \tag{7.4}$$

Equal-frequency Feature Binning

Another well-known method to discretize numerical features implemented by the BOOMER algorithm is *equal-frequency binning*. Unlike equal-width binning, which is supposed to result in bins with values close to each other, this particular discretization method aims to obtain bins that contain approximately the same number of values. The available examples are first sorted in ascending order by their respective feature values to obtain the bins for a particular attribute. This results in a sorted vector of feature values $\left(x_{\tau(1)}, \ldots, x_{\tau(N)}\right)$, where the permutation function $\tau(i)$ specifies the index of the example that corresponds to the i-th element in the sorted vector, as previously defined in (7.1). Afterward, the sorted values are divided into a predefined number of intervals, such that each bin contains the same number of values. Given an individual feature value x_n, the index of the corresponding bin calculates as

$$\sigma_{eq.\text{-}freq.}(x_n) = \lfloor \tau(n) - 1 \rfloor + 1. \tag{7.5}$$

In practice, examples with identical feature values should be prevented from being assigned to different bins. However, for reasons of brevity, this is omitted from the above formula.

Dealing with Nominal Attributes

To handle datasets that do not only include numerical feature values, but also come with nominal attributes, we use an appropriate binning to deal with the latter. It creates a bin for each discrete value encountered in the training data and assigns examples with identical values to the same bin.

Enumeration of Thresholds

Numerical Thresholds

We denote the set of example indices that have been assigned to the m-th bin via a mapping function σ as

$$\mathcal{B}_m = \{n \in \{1, \ldots, N\} \mid \sigma(x_n) = m\}. \tag{7.6}$$

Given M bins previously created for a particular attribute, one can obtain $M - 1$ thresholds that the conditions of potential candidate rules may use. Depending on whether the $\leq$ or $>$ operator is used by a condition, the m-th threshold separates the examples that correspond to the bins $\mathcal{B}_1, \ldots, \mathcal{B}_m$ from the examples that have been assigned to the bins

$\mathcal{B}_{m+1}, \ldots, \mathcal{B}_M$. The individual thresholds $\mu_1, \ldots, \mu_{M-1}$ calculate as the average of the largest and smallest feature value in two neighboring bins $\mathcal{B}_m$ and $\mathcal{B}_{m+1}$. Depending on the characteristics of the binning method at hand, some bins may remain empty. For the enumeration of potential thresholds, bins that are not associated with any examples should be ignored.

Nominal Thresholds

When dealing with bins that have been created from nominal feature values, all examples in a particular bin have the same feature value. In this case, a threshold can be obtained from each available bin, resulting in M thresholds overall. Each of these thresholds corresponds to the discrete feature values assigned to one of the available bins.

Creation of Histograms

Caching of Histograms

When using unsupervised binning methods, the mapping of examples to bins and the thresholds resulting from individual bins must only be determined once during training. They are obtained when a particular attribute is considered by the rule induction algorithm for the first time and are kept in memory for repeated access. In contrast, the histograms that serve as a basis for evaluating candidate rules must be created from scratch whenever a rule should be refined. As shown in Algorithm 10, they result from aggregating the label space statistics of examples that have been assigned to the same bin.

Keeping Track of Covered Examples

Examples that do not satisfy the conditions that have previosuly been added to the body of a rule must be ignored. As defined in (7.2), we use an indicator function $\mathbb{1}_X$ to keep track of the examples that are covered by a rule. In addition, the extent to which the statistics of individual training examples contribute to a histogram depends on their respective weights. This enables the histogram-based search algorithm to use different samples of the available training examples to induce individual rules.

Algorithm 10: Creation of histograms from label space statistics

input : Bins $(\mathcal{B}_m)_m^M$, statistics $S = \{(\boldsymbol{g}_n, \boldsymbol{H}_n)\}_n^N$, indicator function $\mathbb{1}_X$,
weights of training examples $\boldsymbol{w}$
output: Histogram S'

initialize empty histogram $S' = \{(\boldsymbol{g}'_m, \boldsymbol{H}'_m)\}_m^M$, where all elements of $\boldsymbol{g}'_m$ and $\boldsymbol{H}'_m$ are set to zero
for $n = 1$ **to** N **do**
  if $\mathbb{1}_X(n) = 1$ **and** $w_n > 0$ **then**
    obtain bin index $m = \sigma(x_n)$
    update $\boldsymbol{g}'_m = \boldsymbol{g}'_m + w_n \boldsymbol{g}_n$ and $\boldsymbol{H}'_m = \boldsymbol{H}'_m + w_n \boldsymbol{H}_n$
return histogram S'

Evaluation of Refinements

Creating Conditions from Bins

When using the histogram-based search algorithm, evaluating candidate rules in terms of a given loss function follows the same principles as its pre-sorted counterpart in Algorithm 9. However, instead of taking the feature values of individual training examples into account for making up the conditions that can be added to a rule's body, the conditions to be considered by the histogram-based algorithm result from the predetermined thresholds that correspond to bins for a particular attribute. Even when an existing rule should be refined, i.e., when an existing rule covers only a subset of the training examples, the thresholds remain unchanged to increase the algorithm's efficiency.

Aggregation of Statistics

Similar to the pre-sorted rule induction algorithm, the histogram-based approach is based on incrementally aggregating the statistics of training examples that are covered by the considered refinements. However, instead of aggregating statistics at the level of individual training examples, it relies on the statistics that correspond to the individual bins of a histogram. For the efficient evaluation of conditions that use the $>$ operator in case of numerical attributes, or the $\neq$ operator in case of nominal attributes, the algorithm is provided with globally aggregated statistics that are determined beforehand and computes the difference between previously processed statistics that correspond to individual bins and the globally aggregated ones as illustrated in Figure 7.3.

Missing Feature Values

The statistics of examples with missing feature values are excluded from the globally aggregated statistics, as previously described in the context of the pre-sorted algorithm. In addition, the respective examples are ignored when determining the mapping to individual bins. Consequently, the histogram-based rule induction method can handle missing feature values.

Experimental Evaluation

Experimental Setup

To compare the histogram-based rule induction algorithm to its pre-sorted counterpart, we investigated the predictive performance and training times of both approaches in an experimental study. We restricted our experiments to large datasets with many examples and low feature sparsity, as the histogram-based algorithm aims to reduce the time needed for training in such use cases. Among the benchmark datasets that are used in this work (cf. Section 3.2), only two datasets, namely "Mediamill" and "Nus-Wide cVLADplus", meet these criteria. For our experiments, we configured the BOOMER algorithm in the same way as described in Section 7.1, i.e., we learned models of single-label rules that minimize the label-wise logistic loss function in (6.15) and relied on the algorithm's default parameters otherwise. However, the algorithm's ability to use multi-threading was again not used. It is elaborated on in Section 7.3 below. The main goal of our experiments was to investigate how the training time and predictive performance in terms of the Hamming loss, the subset 0/1 loss, and the example-wise F1-measure are affected by varying numbers of bins. Hence, we set the number of bins to be used by the histogram-based approach to 64, 32, 16, 8, or 4% of the distinct feature values available for a particular attribute. In addition, we also included a rather extreme setting, where the number of bins was limited to 8 bins. To assign the available training examples to the available bins, we further tested both binning methods supported by BOOMER, i.e., the equal-width and equal-frequency method.

Table 7.3: Predictive performance of the pre-sorted and histogram-based rule induction algorithm in terms of Hamming loss, Subset 0/1 loss and example-wise F1 measure, as well as the average training times (in seconds) per cross validation fold. The number of bins to be used by the histogram-based approach was set to 64, 32, 16, 8 or 4% of the distinct feature values that are available for an attribute, or to 8 bins.

Dataset	Pre-sorted algorithm	Equal-frequency binning					
		64%	32%	16%	8%	4%	8 bins
Hamming loss							
Mediamill	**3.16**	3.22	3.23	3.22	3.23	3.22	3.24
Nus-Wide cVLADplus	**2.19**	2.23	2.23	2.23	2.23	2.23	2.24
Subset 0/1 loss							
Mediamill	**93.01**	93.89	93.93	93.84	93.89	93.89	93.99
Nus-Wide cVLADplus	**77.03**	77.52	77.53	77.54	77.52	77.55	77.55
Example-wise F1 measure							
Mediamill	**50.61**	49.67	49.67	49.75	49.69	49.65	49.27
Nus-Wide cVLADplus	**27.54**	25.52	25.53	25.57	25.54	25.57	25.10
Training time							
Mediamill	238.6	167.0	109.8	71.9	52.8	46.1	**39.6**
Nus-Wide cVLADplus	4623.7	760.3	641.1	514.4	402.3	281.9	**208.3**

Dataset	Pre-sorted algorithm	Equal-width binning					
		64%	32%	16%	8%	4%	8 bins
Hamming loss							
Mediamill	**3.16**	3.22	3.22	3.22	3.22	3.22	3.27
Nus-Wide cVLADplus	**2.19**	2.23	2.23	2.23	2.23	2.23	2.24
Subset 0/1 loss							
Mediamill	**93.01**	93.91	93.82	93.78	93.81	93.81	94.10
Nus-Wide cVLADplus	**77.03**	77.53	77.54	77.53	77.52	77.56	77.58
Example-wise F1 measure							
Mediamill	**50.61**	49.64	49.70	49.75	49.68	49.74	48.79
Nus-Wide cVLADplus	**27.54**	25.51	25.52	25.54	25.52	25.59	25.09
Training time							
Mediamill	238.6	106.9	75.9	58.6	50.1	45.8	**40.0**
Nus-Wide cVLADplus	4623.7	634.3	513.6	391.5	284.6	241.5	**206.7**

Experimental Results

Table 7.3 shows the predictive performances and training times that result from applying the pre-sorted and histogram-based algorithm to the considered datasets. It can be seen that the latter can reduce the time needed for training, regardless of the dataset and the configuration. As expected, the speedup in training time that results from the histogram-based approach increases when fewer bins are used. The equal-width binning method tends to be slightly more efficient than the equal-frequency method. This is most probably because the former does not require sorting the training examples based on their feature values and therefore comes with linear instead of logarithmic complexity. On the dataset "Mediamill", equal-width binning reduces the training time up to a factor of ≈ 4.8, whereas the training algorithm finishes up to ≈ 22 times faster on the dataset "Nus-Wise-cVLADplus". Regardless of the number of bins, we observe a minor deterioration in predictive performance for all reported evaluation measures compared to the pre-sorted rule induction algorithm. Even though the performance of the histogram-based approach appears to be very resilient against a limitation of the available bins, the variant that is limited to 8 bins always comes with the most significant drop in predictive performance. On the considered datasets, the evaluation scores that are achieved by the equal-width and equal-frequency binning methods are close to each other and do not differ to a degree that is statistically significant.

7.3 Multi-Threading

Possibilties for Parallelization

To make use of the multi-core architecture of today's CPUs, modern implementations of classification algorithms, such as XGBoost (T. Chen and Guestrin, 2016) or LightGBM (Ke et al., 2017), which put great emphasis on scalability, often allow to speed up training by executing certain algorithmic aspects in parallel rather than sequentially. Unlike ensemble methods, where individual members are independent of each other, e.g., in random forests (Breiman, 2001), boosting-based rule learning methods or approaches based on the separate-and-conquer paradigm do not allow to construct individual rules in parallel due to the sequential nature of their training procedure, where each rule is built with respect to its predecessors. Instead, in a multi-label setting, the following possibilities exist to parallelize computational steps that are involved in the induction of a single rule:

- **Multi-threaded Evaluation of Refinement Candidates.** The evaluation of conditions that can possibly be added to a rule's body requires enumerating the feature values of the training examples for each available attribute, aggregating the label space statistics of examples they cover, and computing the predictions and quality of the resulting candidates. Multi-threading can be used to evaluate refinements for different attributes in parallel.
- **Parallel Computation of Predictions and Quality Scores.** For each candidate considered during the construction of a single rule, the predictions for different labels and an estimate of their quality must be obtained. These operations are particularly costly when interactions between labels should be considered. In such a case, the parallelization of these operations across several labels may help reduce training times.
- **Distributed Update of Label Space Statistics.** After a rule has been learned, the label space statistics of all examples it covers must be updated. The complexity of this operation depends on how many examples are covered and is affected by the number of labels for which a rule predicts. Moreover, the update becomes more costly if statistics are not only provided for individual labels but also for pairs of labels or entire labelsets. Depending on the methodology used by a particular rule learning approach, training times may be reduced by updating the statistics for different examples in parallel.

Possible Limitations

The benefits of using the different possibilities for parallelization heavily depend on the characteristics of a particular dataset and the configuration of a learning algorithm. In some cases, the overhead of managing and synchronizing multiple threads outweighs the speedup that the parallel execution of computations may achieve. Consequently, the use of multi-threading may even have a negative effect on the time that is needed for training. Due to the large variety of possible configurations of the BOOMER algorithm, we restrict ourselves to two common use cases: First, we investigate a setting where single-label rules are used to minimize a label-wise decomposable loss function. Second, we consider optimizing a non-decomposable loss function using complete multi-label rules. Similar to Section 7.1, we use default settings for the remaining parameters, including the ability to use sparse feature representations if appropriate (cf. Table 7.2).

The Decomposable Case

Experimental Setup

In the first experiment, we investigated how the time needed by the BOOMER algorithm for learning single-label rules is affected by the use of multi-threading. According to preliminary experiments, training efficiency is unlikely to benefit from updating the label space statistics in parallel in this particular setting. This is because such rules only affect the statistics that correspond to a single label. Due to the small costs of such an update operation, there is only a small potential for runtime improvements. Similarly, we have observed that the benefits of using multiple threads for computing predictions and quality scores tend to be small in the decomposable case. During the first iteration of top-down hill climbing, where candidate rules that contain a single condition in their body are considered, the algorithm must yet decide on a label to predict for. During this initial phase of rule construction, the aforementioned operations come with linear complexity. When evaluating the possible refinements of an existing rule, after the algorithm has decided on a particular label, they even reduce to constant-time operations that cannot be parallelized. Based on these findings, we restrict our study to the multi-threaded evaluation of refinement candidates across different attributes.

> The following parameter allows to specify whether the evaluation of candidate rules should be parallelized across different attributes:
>
> ```
> --parallel-rule-refinement [auto,true,false]
> ```
>
> If set to `auto` (the default value), multi-threading is only used if expected to result in a runtime improvement.

Experimental Results

Table 7.4 reports the training times that result from using a single- or multi-threaded implementation to evaluate candidate rules. By default, BOOMER randomly selects a subset of the available attributes when a rule should be refined. This ensures that the resulting model consists of diverse rules that achieve high predictive accuracy in combination and results in a significant reduction of training time. With such a method for complexity reduction in place, the degree to which multi-threading can be expected to result in runtime improvements mostly depends on the feature sparsity of the training data. Whereas training can be three times faster on datasets with low feature sparsity, multi-threading tends to negatively affect training times if feature sparsity is very high. To investigate how runtimes are affected by the number of attributes that the rule induction algorithm must consider, we also conducted experiments with feature sampling disabled. In such a setting, the multi-threaded implementation outperforms the single-threaded baseline on all considered datasets. As shown in Figure 7.13, the parallelization generally has greater potential for significant speedups (up to a factor of 7) when the training algorithm must process more attributes.

The Non-Decomposable Case

Experimental Setup

A key functionality of BOOMER is its capability to minimize non-decomposable loss functions. In addition to previous section's experiments, we investigate the use of multi-threading to speed up training in this particular scenario. As the training objective, we use the example-wise logistic loss function in (6.23). According to the experimental results in Section 6.6, the use of multi-label rules is crucial for the successful minimization of this loss function that acts as a surrogate for the subset 0/1 loss. We learn complete rules that provide predictions for all available labels in the following study to cater to these results. As discussed in Section 6.3 and shown in (6.13), calculating loss-minimizing predictions for different candidate rules requires solving a linear system in the non-decomposable case. Moreover, a matrix-vector multiplication must be performed to obtain estimate a candidate's quality by substituting its predictions into the objective function in (6.11). The BOOMER algorithm relies on the software libraries *LAPACK* (Anderson et al., 1999) and *BLAS* (Blackford et al., 2002) to implement these linear algebra operations. Depending on the implementations of these libraries, multiple computational threads may be utilized to solve the aforementioned operations.[5] In addition, multi-threading can optionally be used to update the statistics that correspond to different examples in parallel whenever a new rule is added to a model. When dealing with a non-decomposable loss function, the number of Hessians that must be maintained for each example grows exponentially with the number of available labels. Compared to the decomposable case, this makes the update operation more complex and offers potential for runtime improvements via parallelization. Finally, we also consider using multi-threading to parallelize the search for refinements across different attributes. As the evaluation of candidate rules involves the previously mentioned linear algebra operations, this particular parallelization strategy cannot be used in combination with the multi-threading capabilities offered by BLAS and LAPACK, which must be disabled to avoid problems due to nested multi-threading.[6] However, if the number of labels for which a rule may predict is reasonably small and depending on the feature sparsity of a particular dataset, a parallel search for refinements may exhibit greater speedups than achievable by using multi-threaded linear algebra operations. As recent versions of BOOMER come with an approximation technique, which imposes an upper bound on the number of distinct predictions that may be provided by a rule, the former strategy for parallelization appears to be promising in many use cases. In our experiments, we limited the number of distinct predictions to 4% of the available labels. A detailed discussion of the mentioned approximation method is provided in Chapter 8.

5: We used OpenBLAS for all experiments in this work (see `https://www.openblas.net`).

6: When using OpenBLAS, this can be achieved by setting the enviroment variable `OPENBLAS_NUM_THREADS`.

Experimental Results

The training times that result from the use of different multi-threading strategies in the non-decomposable case are shown in Table 7.5. Compared to a configuration where multiple computational threads are only used for linear algebra operations, the additional use of parallelization to update the statistics of different examples reduces training times on 15 out of the 21 considered datasets. Whether a speedup (up to a factor of

Table 7.4: Average training times (in seconds; rounded to one decimal place) per cross validation fold on different datasets (the feature sparsity is given in parantheses) when minimizing a decomposable loss function. The small numbers indicate the speedup that results from the use of multi-threading to evaluate the potential refinements of rules with respect to different attributes in parallel, compared to a single-threaded implementation. By default, the BOOMER algorithm uses multi-threading for experiments that are marked with an asterisk.

Dataset		Single	Multi	
With Feature Sampling				
20NG	(96.81)	**1.1**	1.6	0.69
Bibtex	(96.26)	**1.9**	2.5	0.76
Birds*	(38.64)	0.6	**0.3**	2.00
Bookmarks	(94.16)	44.0	**35.3**	1.25
Delicious	(96.34)	20.0	**19.6**	1.02
Emotions*	(0.33)	0.5	**0.2**	2.50
Enron	(91.60)	**0.2**	0.3	0.67
EukaryoteGO	(99.86)	**0.4**	0.6	0.67
EukaryotePseAAC*	(43.37)	6.5	**2.6**	2.50
EUR-Lex-SM	(95.26)	**9.3**	9.4	0.99
Image*	(0.22)	2.3	**0.8**	2.88
IMDB	(98.06)	12.2	**12.0**	1.02
Langlog	(81.38)	**0.3**	0.4	0.75
Medical	(99.32)	**0.2**	0.3	0.67
Ohsumed	(96.03)	**0.9**	1.2	0.75
Reuters-K500	(98.41)	**1.0**	1.5	0.67
Scene*	(1.15)	3.2	**1.0**	3.20
Slashdot	(99.46)	**0.2**	0.3	0.62
Yahoo-Computers	(99.62)	**0.9**	1.4	0.64
Yahoo-Reference	(99.59)	**0.6**	0.9	0.67
Yahoo-Science	(99.53)	**0.5**	0.8	0.63
Yahoo-Social	(99.71)	**0.9**	1.4	0.64
Yeast*	(0.00)	2.7	**0.9**	3.00

Dataset	Single	Multi	
Without Feature Sampling			
20NG*	39.0	**14.6**	2.67
Bibtex*	86.1	**20.2**	4.26
Birds*	15.7	**2.8**	5.61
Bookmarks*	3729.1	**1571.0**	2.37
Delicious*	261.4	**129.9**	2.01
Emotions*	4.5	**1.0**	4.50
Enron*	11.7	**5.2**	2.25
EukaryoteGO*	71.3	**62.2**	1.15
EukaryotePseAAC*	258.0	**36.6**	7.05
EUR-Lex-SM*	1677.0	**445.0**	3.77
Image*	60.3	**9.7**	6.22
IMDB*	500.0	**121.8**	4.11
Langlog*	17.7	**4.5**	3.93
Medical*	6.4	**5.4**	1.19
Ohsumed*	40.7	**12.2**	3.34
Reuters-K500*	36.2	**8.8**	4.11
Scene*	83.4	**13.7**	6.09
Slashdot*	21.0	**15.7**	1.34
Yahoo-Computers*	1080.3	**392.7**	2.75
Yahoo-Reference*	759.8	**282.7**	2.69
Yahoo-Science*	733.4	**274.0**	2.68
Yahoo-Social*	1608.5	**576.7**	2.79
Yeast*	32.9	**5.3**	6.21

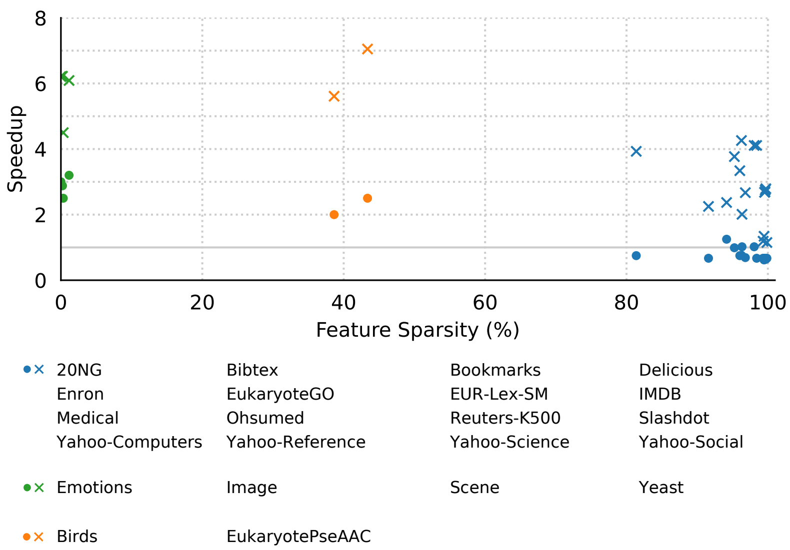


Figure 7.13: The average speedup (or slowdown) in training time per cross validation fold that results from the use of multi-threading to evaluate the potential refinements of rules with respect to different attributes in parallel, compared to a single-threaded implementation. Regardless of whether feature sampling is used (circles) or not (crosses), the speedup depends on whether a dataset has low (green), medium (orange) or high (blue) feature sparsity.

Table 7.5: Average training times (in seconds; rounded to one decimal place) per cross validation fold on different datasets (the feature sparsity and the number of labels are given in parentheses) when minimizing a non-decomposable loss function. Three different configurations that use multi-threading to parallelize different aspects of the BOOMER algorithm are considered. Multi-threading can be used for linear algebra operations, to update the statistics of different examples in parallel or to evaluate the potential refinements of rules with respect to different attributes in parallel. By default, BOOMER only uses multi-threading for linear algebra operators. On datasets with 20 labels or more (marked with an asterisk) multi-threading is used to update statistics as well.

Dataset			**Linear Algebra**	**Linear Algebra & Statistic Update**		**Statistic Update & Refinements**	
20NG*	(96.81,	20)	6.3	5.6	1.13	**4.1**	1.54
Bibtex*	(96.26,	159)	171.7	99.1	1.73	**86.4**	1.99
Birds	(38.64,	19)	**32.8**	34.0	0.97	46.1	0.71
Emotions	(0.33,	6)	**16.5**	17.5	0.94	28.2	0.59
Enron*	(91.60,	53)	5.8	4.6	1.26	**2.6**	2.23
EukaryoteGO*	(99.86,	22)	3.9	**2.5**	1.56	2.6	1.50
EukaryotePseAAC*	(43.37,	22)	**236.7**	237.7	1.00	335.4	0.71
EUR-Lex-SM*	(95.26,	201)	2483.8	2062.7	1.20	**1309.5**	1.90
Image	(0.22,	5)	**71.7**	74.4	0.96	195.1	0.37
IMDB*	(98.06,	28)	80.5	48.3	1.67	**41.3**	1.95
Langlog*	(81.38,	75)	14.9	11.1	1.34	**6.6**	2.26
Medical*	(99.32,	45)	4.7	2.1	2.24	**1.9**	2.47
Ohsumed*	(96.03,	23)	6.3	5.5	1.15	**4.0**	1.58
Reuters-K500*	(98.41,	103)	130.2	104.1	1.25	**88.6**	1.47
Scene	(1.15,	6)	**187.8**	189.8	0.99	519.7	0.36
Slashdot*	(99.46,	22)	2.9	**2.8**	1.04	3.5	0.83
Yahoo-Computers*	(99.62,	33)	9.7	8.2	1.18	**7.7**	1.26
Yahoo-Reference*	(99.59,	33)	7.5	6.3	1.19	**5.8**	1.29
Yahoo-Science*	(99.53,	40)	8.5	7.2	1.18	**6.3**	1.35
Yahoo-Social*	(99.71,	39)	13.0	9.7	1.34	**9.3**	1.40
Yeast	(0.00,	14)	**87.4**	88.3	0.99	128.7	0.68

≈ 2) can be achieved primarily depends on the number of labels. On datasets with less than 20 labels ("Birds", "Emotions", "Image", "Scene" and "Yeast"), the overhead that is introduced by the additional use of multiple threads negatively affects the time needed for training. Similarly, the benefits of using multi-threading to evaluate possible refinements of rules across different attributes in parallel, rather than utilizing multiple threads for linear algebra operations, depend on the dataset. Unlike in the decomposable case, where this particular parallelization strategy is particularly efficient on datasets with small feature sparsity, the opposite can be observed in the non-decomposable setting. To understand this behavior, it is necessary to recall the principles of the sparsity-aware rule induction algorithm in Section 7.1. When dealing with datasets that come with highly sparse feature values, the number of candidate rules that must be considered by the algorithm drastically reduces. Therefore, compared to a dataset with small feature sparsity, the amount of training time spent on linear algebra operations is significantly smaller in such a case. Consequently, there is less potential to speed up these operations by using multi-threading. For this reason, multi-threading should be used to parallelize the search for refinements across multiple attributes on datasets with high feature sparsity (resulting in speedups up to a factor of ≈ 2.5). In contrast, the available processor cores are better utilized for parallelizing linear algebra operations if the feature sparsity is small.

Parallelized Prediction

Predicting for Different Examples in Parallel

In addition to the use of multi-threading to speed up training, parallelization can also be used when predictions for several examples should be obtained. Delivering predictions for a given set of examples requires first enumerating the rules in a model and identifying those rules that cover each of the provided examples. Second, the predictions provided by the heads of these rules must be aggregated to obtain an overall prediction. As both of these steps may be carried out independently for each example, multi-threading can be used to predict for different examples in parallel.

> ⓘ By default, the BOOMER algorithm obtains predictions for different examples in parallel. This behavior may be adjusted via the parameter `--parallel-prediction`.

However, prediction time is usually not a limiting factor when dealing with datasets comparable to those presented in Section 3.2 in terms of dimensionality. Hence, we leave it at the mention of this possibility and forego a closer examination of its advantages and disadvantages.

7.4 Discussion

This chapter provided a detailed discussion of the pre-sorted search algorithm used for the efficient induction of rules by successful rule learning algorithms like RIPPER. We also discussed extensions to the basic algorithm that enable dealing with both nominal attributes and missing feature values. Furthermore, we demonstrated how sparse data structures can be used to represent feature values. Our experiments suggest that the exploitation of feature sparsity drastically improves training efficiency in many cases, especially when dealing with text classification datasets. These results are complemented by a study on the histogram-based induction of rules. We showed that the latter approach helps to reduce training times on datasets with low feature sparsity, where the ability to use sparse feature representations does not provide any benefits. Although the basic principles that are discussed in this chapter are widely adopted by existing algorithms for the construction of rule- or tree-based models, publications on the topic are typically restricted to a high-level view of the discussed techniques. Hence, the goal of this chapter is to supplement existing literature and provide an extensive and unified view of commonly used optimizations and approximations. Moreover, an empirical investigation using real-world benchmark datasets provided us with valuable insights for the practical use of our algorithms. In particular, this applies to our study regarding the possibilities to speed up different aspects of a multi-label rule learning algorithm via parallelization. We observed that even though parallelization helps to reduce training times in many cases, its potential benefits heavily depend on the characteristics of a dataset and the configuration of an algorithm. The experimental results presented in this chapter helped us provide sane defaults for the publicly available BOOMER algorithm that can be expected to work well in practice.

Gradient-based Label Binning | 8

As argued in Section 3.5, the ability to model dependencies between labels is crucial to effectively optimize non-decomposable evaluation measures if such patterns exist in the data. As this is often the case in real-world scenarios, research on multi-label classification is heavily driven by the requirement to capture correlations in the label space. Following this common trend and motivated by the need for MLC methods that can flexibly be adjusted to different evaluation measures, including non-decomposable ones, we presented the BOOMER algorithm in Chapter 6. It is based on the gradient boosting framework that provides a well-studied foundation for learning approaches specifically tailored to a particular loss function. Furthermore, the experiments in Section 6.6 show its ability to achieve high predictive accuracy and demonstrate that multi-label rules that capture information about multiple labels in their heads are well-suited to minimize non-decomposable losses, such as the subset 0/1 loss. Our results suggest that the utilization of second-order derivatives, as used by many recent boosting approaches (e.g., T. Chen and Guestrin, 2016; Ke et al., 2017; Z. Zhang and C. Jung, 2020), helps to minimize non-decomposable losses due to the information about pairs of labels it incorporates into the optimization process. On the downside, this comes with high computational costs, even if the number of labels is small. In this chapter, we address the computational bottleneck of such an approach — the need to solve a system of linear equations — by integrating a novel approximation technique, referred to as *gradient-based label binning* (GBLB), into the boosting procedure. Based on the derivatives computed during training, it dynamically groups the labels into a predefined number of bins to impose an upper bound on the dimensionality of the linear system. The proposed methodology, together with experiments using the BOOMER algorithm, was first published by Rapp, Loza Mencía, Fürnkranz, and Hüllermeier (2021) . The empirical results presented in said work show that the use of GBLB may boost the speed of training without any significant loss in predictive performance.

Rapp, Loza Mencía, Fürnkranz, and Hüllermeier (2021): 'Gradient-based Label Binning in Multi-label Classification'

8.1 Complexity Analysis

Aggregation of Gradients and Hessians

The objective function in (6.11), the BOOMER algorithm aims to minimize at each training iteration, depends on gradient vectors and Hessian matrices that correspond to individual training examples. Given K labels, the former consist of K elements, whereas the latter are symmetric matrices with $K(K+1)\,/\,2$ non-zero elements, one for each label and each pair of labels. The induction of a new rule, using a search algorithm as described in Chapter 7, requires summing up the gradient vectors and Hessian matrices of the covered examples to form the linear system in (6.13). Instead of computing the sums for each candidate rule individually, the candidates are processed in a predetermined order, such that each one

Algorithm 11: Candidate evaluation without (left) / with GBLB (right)

input : Gradient vector $\boldsymbol{g}$, Hessian matrix H, L_2 regularization weight δ

output : Predictions $\hat{\boldsymbol{p}}$, quality score q, mapping $\boldsymbol{m}$ (if GBLB is used)

	Without GBLB	With GBLB
1	$R = \operatorname{diag}(\delta)$	$\boldsymbol{m} = \textsc{map\_to\_bins}(\boldsymbol{g}, H, \delta)$ $\boldsymbol{g}, H, R = \textsc{aggregate}(\boldsymbol{m}, \boldsymbol{g}, H, \delta)$
2	$\hat{\boldsymbol{p}} = \text{DSYSV}(-\boldsymbol{g}, H + R)$	|
3	$q = \text{DDOT}(\hat{\boldsymbol{p}}, \boldsymbol{g}) +$ $(0.5 \cdot \textsc{ddot}(\hat{\boldsymbol{p}}, \text{DSPMV}(\hat{\boldsymbol{p}}, H)))$	*same as left* |
4	**return** $\hat{\boldsymbol{p}}, q$	**return** $\hat{\boldsymbol{p}}, q, \boldsymbol{m}$

covers one or several additional examples compared to its predecessor. As a result, an update of the sums with complexity $\mathcal{O}(K^2)$ must be performed for each example and attribute that is considered for making up new candidates.

Complexity of Linear Algebra Operations

Algorithm 11 shows the steps that are necessary to compute the confidence scores to be predicted by an individual candidate rule, as well as a score that assesses its quality, if the loss function is non-decomposable. The modifications that are necessary to implement GBLB are shown to the right of the original lines of code. Originally, the given gradient vector and Hessian matrix, which result from summation over the covered examples, are used as a basis to solve the system of linear equations in (6.13). For solving such a linear system with a symmetric coefficient matrix, the BOOMER algorithm's implementation uses the routine DSYSV provided by the LAPACK (Anderson et al., 1999) software library (cf. Algorithm 11, line 2). The computation of a corresponding quality score requires substituting the calculated scores into (6.11). It involves invocations of the BLAS (Blackford et al., 2002) operations DDOT for vector-vector multiplication and DSPMV for vector-matrix multiplication (cf. Algorithm 11, line 3). Whereas the operation DDOT comes with linear costs, the DSPMV and DSYSV routines have quadratic and cubic complexity, i.e., $\mathcal{O}(K^2)$ and $\mathcal{O}(K^3)$, respectively.[1] As Algorithm 11 must be executed for each candidate rule, it is the computationally most expensive operation taking part in a multivariate boosting algorithm that minimizes a non-decomposable loss function.

1: Information on the complexity of the BLAS and LAPACK routines referred to in this chapter can be found online at `http://www.netlib.org/lapack/lawnspdf/lawn41.pdf`.

Dimensionality Reduction

GBLB addresses the computational complexity of Algorithm 11 by mapping the available labels to a predefined number of bins M and aggregating the elements of the gradient vector and Hessian matrix accordingly. If $M \ll K$, this significantly reduces their dimensionality and limits the costs of the BLAS and LAPACK routines. As a result, given that the overhead introduced by the mapping and aggregation functions is small, we expect an overall reduction in training time.

Limitations

In this chapter, we do not address the computational costs of summing up the gradients and Hessians that correspond to individual examples. However, the proposed method is designed such that it can be combined with methods dedicated to this aspect. Albeit restricting themselves to decomposable losses, Si et al. (2017) have proposed a promising method that ensures many gradients evaluate to zero. This approach, which was partly adopted by Z. Zhang and C. Jung (2020), restricts the labels that must be considered to those with non-zero gradients. However, to maintain sparsity among the gradients, strict requirements must be fulfilled by the loss function. Among many others, the logistic loss function in (6.23) does not meet these requirements. In contrast, the approach that is investigated in the following does not impose any restrictions on the loss function. Even though it can potentially be used with any loss function, it is intended for use cases where a non-decomposable loss should be minimized. This is because it explicitly addresses the computational bottleneck of such a training procedure, i.e., the need to solve the linear system in (6.13), which reduces to an operation with linear complexity in the decomposable case.

8.2 Mapping Labels to Bins

Gradient-based Mapping

GBLB evolves around the idea of assigning the available labels to a predefined number of bins whenever a potential ensemble member is evaluated during training (cf. MAP_TO_BINS in Algorithm 11). This is similar to problem transformation methods like LP or RAkEL, where subsets of labels are dealt with jointly by a learning algorithm (cf. Section 3.6). However, as these transformation methods require training a model for each considered labelset, they usually come with high computational demands. To compensate for this, the label space transformation approaches introduced in Section 3.8 aim to reduce the complexity of the label space to be dealt with by multi-label classifiers. Notwithstanding that such a reduction in complexity is indispensable in cases where thousands or even millions of labels should be handled, it often remains unclear what measure such dimensionality reduction methods aim to optimize. Unlike the reduction methods mentioned above, GBLB is integrated into the gradient boosting framework and therefore is inherently tailored to a particular loss function. Such a tight integration also allows to dynamically adjust to different regions in input space for which an ensemble member may predict. By taking into account the derivatives guiding the optimization process to determine the mapping from labels to bins, the impact of the approximation is kept at a minimum.

Mapping Criteria

The goal of GBLB is to map the available labels $\lambda_1, \ldots, \lambda_K$ to a predefined number of bins $\mathcal{B}_1, \ldots, \mathcal{B}_M$. To obtain the index of the bin, a particular label λ_k should be assigned to, we use a mapping function $\sigma : \mathbb{R} \rightarrow \mathbb{N}^+$ that depends on a given criterion $\theta_k \in \mathbb{R}$. In this work, we use the criterion

$$\theta_k = -\frac{g_k}{h_{kk} + \delta}, \tag{8.1}$$

which takes the gradient and Hessian for the respective label, as well as the L_2 regularization weight, into account. Per (6.14), it corresponds to the optimal prediction when considering the label in isolation, i.e., when assuming that the predictions for other labels will be zero. The criterion can be obtained for each label individually, so the computational overhead is kept at a minimum. Based on the assignments that are provided by a mapping function σ, we denote the set of label indices that belong to the m-th bin as

$$\mathcal{B}_m = \{k \in \{1, \ldots, K\} \mid \sigma(\theta_k) = m\}. \tag{8.2}$$

Binned Predictions

Labels should be assigned to the same bin if the corresponding confidence scores, which will be presumably be predicted by an ensemble member, are close to each other. If the optimal scores to be predicted for certain labels are very different in absolute size or even differ in their sign, the respective labels should be mapped to different bins. Based on this premise, we limit the number of distinct scores an ensemble member may predict by enforcing the restriction

$$\hat{p}_i = \hat{p}_j, \forall i, j \in \mathcal{B}_m. \tag{8.3}$$

It requires that a single score is predicted for all labels that have been assigned to the same bin. Given that the mentioned prerequisites are met, we expect the difference between the scores that are predicted for a bin and those that are optimal with respect to the original labels to be reasonably small.

Equal-Width Label Binning

Properties of Equal-width Binning

Principally, different approaches for implementing the mapping function σ are conceivable. We use *equal-width* binning, as this well-known method provides two advantages: First, unlike other methods, such as equal-frequency binning, it does not involve sorting and can therefore be applied in linear time. Second, the boundaries of the bins are chosen such that the absolute difference between their smallest and largest value, referred to as the *width*, is the same for all bins. As argued in the previous section, this is a desirable property in our particular use case.

Negative and Positive Bins

Furthermore, we want to prevent labels for which the predicted score should be negative from being assigned to the same bin as labels for which the prediction should be positive. Otherwise, the predictions would be suboptimal for some of these labels. Therefore, we strictly separate between *negative* and *positive bins*. Given $B_\ominus$ negative and $B_\oplus$ positive bins, the width calculates as

$$\omega_\ominus = \frac{\max_\ominus - \min_\ominus}{B_\ominus} \quad \text{and} \quad \omega_\oplus = \frac{\max_\oplus - \min_\oplus}{B_\oplus}, \tag{8.4}$$

for the positive and negative bins, respectively. By $\max_\ominus$ and $\max_\oplus$ we denote the largest value in $\{\theta_1, \dots, \theta_K\}$ with negative and positive sign, respectively. Accordingly, $\min_\ominus$ and $\min_\oplus$ correspond to the smallest value with the respective sign. Labels for which $\theta_k = 0$, i.e., labels with zero gradients, can be ignored. As no improvement in terms of the loss function can be expected, we explicitly set the prediction to zero in such a case.

Mapping Function

Once the width of the negative and positive bins has been determined, the mapping from individual labels to one of the $M = B_\ominus + B_\oplus$ bins can be obtained via the function

$$\sigma_{\textit{eq.-width}}(\theta_k) = \begin{cases} \min\left(\lfloor \frac{\theta_k - \min_\ominus}{\omega_\ominus} \rfloor + 1, B_\ominus\right), & \text{if } \theta_k < 0 \\ \min\left(\lfloor \frac{\theta_k - \min_\oplus}{\omega_\oplus} \rfloor + 1, B_\oplus\right) + B_\ominus, & \text{if } \theta_k > 0. \end{cases} \tag{8.5}$$

Aggregation of Statistics

Reformulizations

By exploiting the restriction introduced in (8.3), the gradients and Hessians that correspond to labels in the same bin can be aggregated to

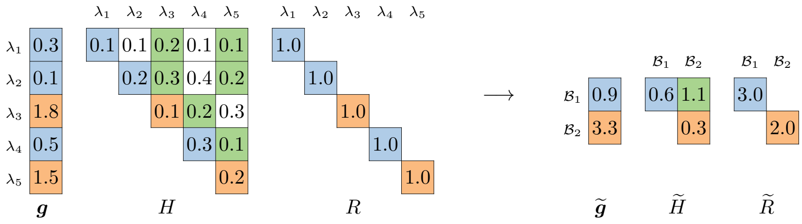


Figure 8.1: Illustration of how a gradient vector, a Hessian matrix, and a regularization matrix for five labels $\lambda_1, \dots, \lambda_5$ are aggregated with respect to two bins $\mathcal{B}_1 = \{1, 2, 4\}$ and $\mathcal{B}_2 = \{3, 5\}$ when using L_2 regularization with $\delta = 1$. Elements with the same color are added up for aggregation.

obtain a gradient vector and a Hessian matrix with reduced dimensions (cf. AGGREGATE in Algorithm 11). To derive a formal description of this aggregation, we first rewrite the objective function (6.11) in terms of sums instead of using vector and matrix multiplications. This results in the formula

$$\widetilde{\mathcal{R}}(f_t) = \sum_{n=1}^{N} \sum_{i=1}^{K} \left(g_i^n \hat{p}_i + \frac{1}{2} \hat{p}_i \left(h_{ii}^n \hat{p}_i + \sum_{\substack{j=1, \\ j \neq i}}^{K} h_{ij}^n \hat{p}_j \right) \right) + \Omega(f_t). \tag{8.6}$$

Aggregation of Gradients and Hessians

Based on the constraint given in (8.3) and due to the distribution property of the multiplication, the equality

$$\sum_{i=1}^{K} x_i \hat{p}_i = \sum_{j=1}^{M} \left(\hat{p}_j \sum_{i \in \mathcal{B}_j} x_i \right), \tag{8.7}$$

where x_i is any term dependent on i, holds. It can be used to rewrite (8.6) in terms of sums over the bins, instead of sums over the individual labels. For brevity, we denote the sum of the gradients, as well as the sum of the elements on the diagonal of the Hessian matrix, that correspond to the labels in bin $\mathcal{B}_m$ as

$$\widetilde{g}_m = \sum_{i \in \mathcal{B}_m} g_i \quad \text{and} \quad \widetilde{h}_{mm} = \sum_{i \in \mathcal{B}_m} h_{ii}. \tag{8.8}$$

To abbreviate the sum of Hessians that correspond to a pair of labels that have been assigned to different bins $\mathcal{B}_m$ and $\mathcal{B}_q$, we use the short-hand notation

$$\widetilde{h}_{mq} = \sum_{i \in \mathcal{B}_m} \sum_{j \in \mathcal{B}_q} h_{ij}. \tag{8.9}$$

By exploiting (8.7) and using the abbreviations introduced above, the objective function in (8.6) can be rewritten as

$$\widetilde{\mathcal{R}}(f_t) = \sum_{n=1}^{N} \sum_{m=1}^{M} \left(\hat{p}_m \widetilde{g}_m^n + \frac{1}{2} \hat{p}_m \left(\hat{p}_m \widetilde{h}_m^n + \sum_{\substack{q=1, \\ q \neq m}}^{M} \hat{p}_q \widetilde{h}_{mq}^n \right) \right) + \Omega(f_t), \tag{8.10}$$

which can afterwards be turned into the original notation based on vector and matrix multiplications. The resulting formula

$$\widetilde{\mathcal{R}}(f_t) = \sum_{n=1}^{N} \left(\widetilde{\boldsymbol{g}}_n \hat{\boldsymbol{p}}_n^t + \frac{1}{2} \hat{\boldsymbol{p}}_n^t \widetilde{H}_n \hat{\boldsymbol{p}}_n^t \right) + \Omega(f_t) \tag{8.11}$$

has the same structure as shown initially in (6.11). However, the gradient vector $\boldsymbol{g}$ and the Hessian matrix H have been replaced by $\widetilde{\boldsymbol{g}}$ and $\widetilde{H}$, respectively. Consequently, when calculating the scores to be predicted by an ensemble member by solving (6.13), the coefficients and ordinates that take part in the linear system do not correspond to individual labels but result from the sums in (8.8) and (8.9). As a result, the number of linear equations has been reduced from the number of labels K to the number of non-empty bins, which is at most M.

Conversion of the Regularization Matrix

An example that illustrates the aggregation of a gradient vector and a Hessian matrix is given in Figure 8.1. It also shows how the regularization matrix R is affected. When dealing with bins instead of individual labels, the L_2 regularization term in (6.12) becomes

$$\Omega_{L2}(f_t) = \frac{1}{2}\delta \sum_{m=1}^{M} \left(|\mathcal{B}_m| \, \hat{p}_m^2 \right), \tag{8.12}$$

where $|\mathcal{B}_m|$ denotes the number of labels that belong to a particular bin. As a consequence, the regularization matrix becomes

$$\widetilde{R} = \operatorname{diag}\left(\delta\,|\mathcal{B}_1|, \dots, \delta\,|\mathcal{B}_M|\right). \tag{8.13}$$

8.3 Experimental Evaluation

Experimental Setup

To investigate the effects GBLB has on predictive performance and training time in isolation, we chose a single configuration of the BOOMER algorithm as the basis for our experiments.[2] We used 10-fold cross validation (cf. Section 3.3) to train models on 15 benchmark datasets (cf. Section 3.2) using the logistic loss function in (6.23) as a surrogate for minimizing the subset $0/1$ loss. Each model consists of $5,000$ rules that have been learned on varying subsets of the training examples, drawn with replacement. Feature sampling has been used to restrict the refinement of rules to random subsets of the available attributes. As for the learning rate and the L_2 regularization weight, we used the default values 0.3 and 1.0, respectively. Besides the original algorithm, as presented in Chapter 6, we tested an implementation that uses GBLB. For a broad analysis, we set the maximum number of bins to 32, 16, 8, and 4% of the available labels. In addition, we investigated an extreme setting with two bins, where all labels with positive and negative criteria are assigned to the same bin, respectively.

2: The experiments that are discussed in this section were conducted using version 0.5.0 of the BOOMER algorithm. It is available at `https://github.com/mrapp-ke/Boomer/releases/tag/0.5.0`

> The BOOMER algorithm provides the following parameter to specify whether GBLB should be used:
>
> ```
> --label-binning [auto,equal-width,None]
> ```
>
> If set to `auto` (the default value), GBLB is enabled automatically for the minimization of non-decomposable losses. Information on how to configure the number of bins can be found in the documentation.

Speedup in Training Time

Table 8.1 shows the average time per cross validation fold needed by the considered approaches for training. Compared to the baseline that

Table 8.1: Average training times (in seconds) per cross validation fold on different datasets (the number of labels is given in parentheses). The small numbers specify the speedup that results from using GBLB with the number of bins set to 32, 16, 8, and 4% of the labels or using two bins. Variants that are equivalent to two bins are omitted.

Dataset		**No GBLB**	**GBLB**									
			32%		**16%**		**8%**		**4%**		**2 bins**	
EUR-Lex-SM	(201)	46947	54985	0.85	44872	1.05	38222	1.23	33658	1.39	**21703**	2.16
EukaryotePseAAC	(22)	16033	3593	4.46	2492	6.43	2195	7.30	—		**1534**	10.45
Reuters-K500	(103)	12093	6930	1.75	4197	2.88	3353	3.61	2803	4.31	**2743**	4.41
Bibtex	(159)	2507	2599	0.96	2765	0.91	2649	0.95	2456	1.02	**2125**	1.18
Yeast	(14)	2338	998	2.34	761	3.07	525	4.45	—		**521**	4.49
Birds	(19)	2027	701	2.89	505	4.01	337	6.01	—		**336**	6.03
Yahoo-Social	(39)	1193	261	4.57	217	5.50	192	6.21	**139**	8.58	175	6.82
Yahoo-Computers	(33)	874	172	5.08	134	6.52	126	6.94	**101**	8.65	123	7.11
Yahoo-Science	(40)	735	200	3.67	160	4.59	135	5.44	**106**	6.93	136	5.40
Yahoo-Reference	(33)	571	174	3.28	141	4.05	129	4.43	**110**	5.19	137	4.17
Slashdot	(20)	518	154	3.36	117	4.43	**86**	6.02	—		119	4.35
EukaryoteGO	(22)	191	79	2.42	74	2.58	**60**	3.18	—		64	2.98
Enron	(53)	181	69	2.62	52	3.48	48	3.77	47	3.85	**44**	4.11
Medical	(45)	170	60	2.83	57	2.98	55	3.09	**50**	3.40	51	3.33
Langlog	(75)	132	126	1.05	112	1.18	105	1.26	**101**	1.31	102	1.29
Avg. Speedup				2.81		3.58		4.61		**4.86**		4.00

does not use GBLB, the training time can always be reduced by utilizing GBLB with a suitable number of bins. Using fewer bins tends to speed up the training process, although approaches that use the fewest bins are not always the fastest. On average, limiting the number of bins to 4% of the labels results in the most significant speedup (by factor 5). However, the possible speedup depends on the dataset at hand. For example, on the dataset "EukaryotePseAac", the average training time is reduced by factor 10, whereas no significant speedup is achieved for "Bibtex".

Effects on Predictive Performance

To be useful in practice, the speedup resulting from GBLB should not come with a significant deterioration in terms of the target loss. Therefore, we report the predictive performance of the considered approaches in Table 8.2. Besides the subset 0/1 loss, which we aim to minimize in this chapter, we also include the Hamming loss as a commonly used representative of decomposable loss functions (cf. Section 3.3). When focusing on the subset 0/1 loss, we observe that the baseline algorithm without

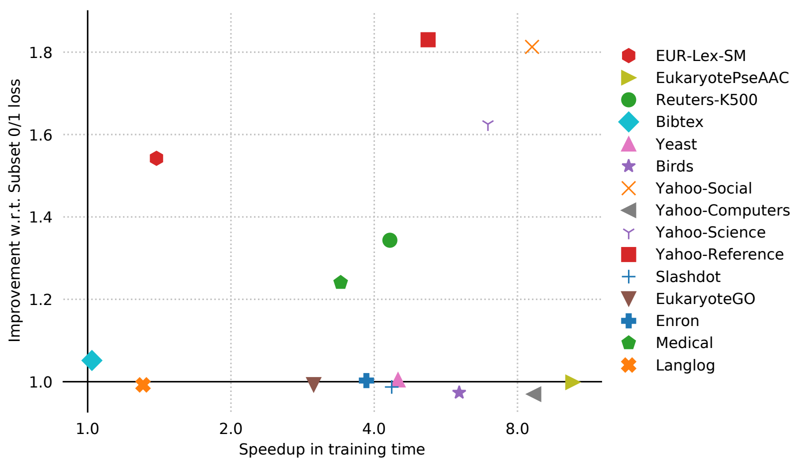


Figure 8.2: Relative difference in training time and Subset 0/1 loss (both calculated as the baseline's value divided by the value of the respective approach) per cross validation fold that results from using GBLB with the number of bins set to 4% of the labels.

Table 8.2: Predictive performance of different approaches in terms of the subset 0/1 loss and the Hamming loss (smaller values are better).

Dataset	Label-wise	No GBLB	GBLB				
			32%	16%	8%	4%	2 bins
Subset 0/1 loss							
EUR-Lex-SM	61.63	69.53	45.07	**45.03**	45.30	45.08	47.32
EukaryotePseAAC	85.09	65.43	65.37	**65.28**	65.68	—	65.52
Reuters-K500	71.37	71.07	53.70	53.22	53.07	**52.90**	53.40
Bibtex	85.99	81.31	**77.28**	77.44	77.55	77.32	78.95
Yeast	84.94	76.54	76.91	76.42	76.87	—	**76.21**
Birds	45.29	45.30	**45.14**	45.45	45.60	—	46.53
Yahoo-Social	50.65	64.30	34.49	**34.40**	34.84	35.47	35.37
Yahoo-Computers	58.04	**46.26**	46.65	47.09	46.94	47.70	47.42
Yahoo-Science	74.00	85.80	**50.89**	51.20	52.07	52.79	52.04
Yahoo-Reference	58.19	74.14	**39.82**	40.16	40.73	40.51	40.48
Slashdot	63.88	**46.62**	46.64	46.64	47.73	—	47.22
EukaryoteGO	30.63	28.35	28.39	28.24	**28.10**	—	28.55
Enron	88.19	83.14	83.32	83.32	83.38	**82.91**	82.97
Medical	28.25	28.82	23.13	**22.62**	23.08	23.23	22.77
Langlog	79.59	78.84	79.11	79.25	**78.63**	79.45	79.45
Hamming loss							
EUR-Lex-SM	0.55	0.91	0.40	**0.39**	0.40	0.40	0.42
EukaryotePseAAC	**5.02**	5.65	5.64	5.63	5.67	—	5.66
Reuters-K500	1.11	1.71	1.11	**1.09**	**1.09**	**1.09**	1.10
Bibtex	**1.25**	1.45	1.27	1.27	1.27	1.28	1.31
Yeast	19.75	19.01	18.87	19.08	19.01	—	**18.80**
Birds	3.91	3.79	3.80	3.79	**3.73**	—	3.87
Yahoo-Social	1.90	3.81	**1.79**	1.80	1.83	1.87	1.87
Yahoo-Computers	3.10	**2.97**	3.00	3.02	3.03	3.08	3.06
Yahoo-Science	2.83	5.85	**2.74**	2.75	2.81	2.84	2.79
Yahoo-Reference	2.30	4.95	**2.28**	2.30	2.34	2.33	2.32
Slashdot	**4.02**	4.24	4.24	4.25	4.37	—	4.30
EukaryoteGO	**1.89**	1.95	1.95	1.94	1.92	—	1.98
Enron	**4.53**	4.72	4.77	4.77	4.72	4.72	4.73
Medical	0.84	1.05	0.80	**0.77**	0.79	0.81	0.79
Langlog	1.52	1.52	**1.50**	1.51	**1.50**	1.52	1.52

GBLB exhibits subpar performance on some datasets, namely "EUR-Lex-SM", "Reuters-K500", "Bibtex", "Yahoo-Social", "Yahoo-Science", "Yahoo-Reference", and "Medical". This becomes especially evident when compared to an instantiation of the algorithm that targets the Hamming loss via minimization of the label-wise decomposable logistic loss function in (6.15). In said cases, the latter approach performs better even though it is not tailored to the subset 0/1 loss. Although the baseline performance could probably be improved by tuning the regularization weight, we decided against parameter tuning, as it exposes an interesting property of GBLB. On the mentioned datasets, approaches that use GBLB appear to be less prone to converge towards local minima. Regardless of the number of bins, they clearly outperform the baseline. According to the Friedman test (cf. Section 3.4), these differences are significant with $\alpha = 0.01$. The Nemenyi post-hoc test yields critical distances for each of the GBLB-based approaches when compared to the baseline. On the remaining datasets, where the baseline without GBLB already performs well, the use of GBLB produces competitive results. In these cases, the

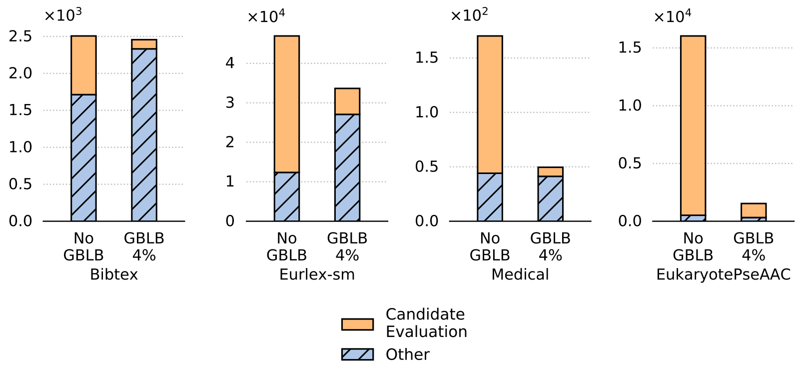


Figure 8.3: Average proportion of training time per cross validation fold that is used for the evaluation of candidate rules without GBLB and when using GBLB with the number of bins set to 4% of the labels.

Friedman test confirms the null hypothesis with $\alpha = 0.1$. In Figure 8.2, an overview of how the training time and the predictive performance in terms of the subset 0/1 loss are affected when restricting the number of bins to 4% of the labels.

Analysis of Training Time

To better understand the differences in speedups that may be achieved using GBLB, a detailed analysis is given in the following for four datasets with varying characteristics. Figure 8.3 depicts the training time needed by the baseline approach and a GBLB-based approach with the number of bins set to 4% of the labels. Besides the total training time, we also show the amount of time spent on the evaluation of candidate rules (cf. Algorithm 11), which is the algorithmic aspect addressed by GBLB. For all given scenarios, it can be seen that the time needed for candidate evaluation could successfully be reduced. Nevertheless, the effects on the overall training time vastly differ. On the datasets "Bibtex" and "EUR-Lex-SM", the time spent on parts of the algorithm other than the candidate evaluation increased when using GBLB, resulting from more specific rules being learned. On the one hand, this required more candidates to be evaluated and therefore hindered the overall speedup. On the other hand, the resulting rules clearly outperformed the baseline, according to Table 8.2. On the dataset "Bibtex", even without GBLB, the candidate evaluation was not the most expensive aspect of training. Due to its binary attributes, the number of potential candidates is small compared to the large number of examples. As a result, most of the computation time is spent on summing up the gradients and Hessians of individual examples (cf. Section 8.1). The impact of speeding up the candidate evaluation is therefore limited. On the datasets "Medical" and "EukaryotePseAAC", where the candidate evaluation was originally the most expensive aspect, a significant reduction of training time could be achieved by making that operation more efficient. The time spent on other parts of the algorithm remained mostly unaffected in these cases. As mentioned earlier, this includes the summation of gradients and Hessians, which becomes the most time-consuming operation when using GBLB. Addressing this aspect holds the greatest potential for further performance improvements.

8.4 Discussion

In this chapter, we discussed an approximation technique that dynamically assigns the available labels to a predefined number of bins. The mapping from labels to bins is based on the gradients and Hessians that guide the training process of a multivariate boosting algorithm. Our experiments confirm that this dimensionality reduction successfully reduces the training time needed for minimizing non-decomposable loss functions, such as the subset 0/1 loss. According to our results, this speedup does not come with any significant loss in predictive performance. In several cases, the proposed method even outperforms the baseline by a large extent due to its ability to overcome local minima without the necessity for extensive parameter tuning. Despite the promising results, the use of non-decomposable loss functions in the boosting framework remains computationally challenging. Ideas to address the remaining limitations, such as additional measures that allow exploiting sparsity in the label space, are left for future work and are discussed in Section 9.2.

Conclusions | 9

The large variety of different rule learning techniques found in the literature provides a solid foundation for designing rule-based approaches to multi-label classification. Different implementations of the individual aspects that make up an effective multi-label rule learning algorithm allow the training of models that differ quite drastically in their characteristics and come with their own advantages and disadvantages. On the one hand, rules can be considered a versatile tool for tackling multi-label classification problems due to this flexibility. On the other hand, as there is no rule-based solution to this problem domain that can be expected to behave optimal in any given use case, a multi-label rule learning algorithm should carefully be tailored to the requirements at hand. Therefore, two fundamentally different directions with varying goals have been pursued in this work. In addition to a rule learning approach for the induction of compact models that consist of discrete rules, a second method for learning large ensembles of probabilistic rules was investigated. In Section 9.1, we summarize of the results that have been obtained with regard to these approaches and discuss to what extent they address the research challenges outlined in Section 1.1, including the interpretability of predictive models, the importance of capturing label correlations, the desire to cater to different target measures, and the need for computationally efficient algorithms. Beyond the algorithms and concepts that have been discussed in this work, the large number of possibilities to tackle the multi-label classification setting by means of rules leaves room for future research. In Section 9.2, we outline several extensions to the methodologies that have been discussed in this work. They may help to enhance the capabilities of the proposed methods or overcome some of their limitations in the future.

9.1 Summary of Results

Multi-label Rule Learning Framwork

As the first contribution of this work, in Chapter 4, a unified framework for implementing rule-based multi-label classification methods was presented. It was designed in a modular fashion, which enables to implement certain aspects of the framework in different ways, depending on the goals a particular approach should achieve. For example, the methods proposed in Chapter 5 and Chapter 6 mostly rely on a greedy top-down search for the generation of candidate rules, whereas an alternative approach that extracts rules from ensembles of decision trees is used for the empirical study in Section 5.3. Besides other techniques that may be considered for candidate generation, such as association rule mining or branch-and-bound algorithms, the framework also provides freedoms when it comes to the construction of rule heads. Whereas rules that provide deterministic predictions should probably be preferred in applications where great emphasis is put on the interpretability of models, probabilistic rules appear to be a natural choice in rule-based ensemble methods. In both cases, depending on the multi-label evaluation measure one is interested in, the rules may either be restricted to single labels or provide information about several labels in their heads. Our experiments suggest that single-label heads are often better if decomposable evaluation measures should be optimized. In contrast, the effective optimization of non-decomposable target measures demands complete or partial multi-label rules that can capture local label dependencies. Compared to ensemble methods that use probabilistic rules, approaches that focus on compact models with deterministic rules come with high variance and therefore are prone to overfitting. To overcome this issue, we have discussed countermeasures, such as pruning techniques, stopping criteria, or possible post-optimizations, that are successfully used in traditional rule learning methods for single-label classification. Finally, the framework in Chapter 4 also allows the use of complexity reduction methods by tailoring individual rules to a subset of the available training examples, restricting their conditions to specific attributes, or deciding on a subset of labels for which they may predict.

Separate-and-Conquer Approach

Chapter 5 investigated the first instantiation of the framework previously introduced in Chapter 4. It is based on the separate-and-conquer paradigm many traditional rule learners use. Similar to existing SeCo-based approaches for single-label classification, it results in a sorted list of deterministic rules. They are applied in the order of their induction to obtain predictions. As illustrated by several examples throughout the chapter (cf. Table 5.2 and Table 5.6), such a model representation appears to be promising if human domain experts should be enabled to analyze a model globally. However, one may consider the necessity to interpret individual rules in the context of their predecessors as a disadvantage. By restricting the predictions of rules for particular labels to be either positive or negative, a multi-label SeCo algorithm can be used to obtain a model in disjunctive normal form. Such a model, where the predictions of individual rules may not conflict, can likely be understood more easily by human analysts. As argued in Section 5.1, an application of the SeCo principle to multi-label data demands for a strategy to keep track of partially covered training examples. As a solution to this problem, we have introduced the notion of label weights, which are flexible enough to deal with partially covered examples in different ways and allow for a straightforward extension to weighted covering strategies. In addition, generalizations of traditional rule learning heuristics are needed to assess the quality of multi-label rules for multiple labels. In Section 5.2, we have introduced a novel notation to define such heuristics more intuitively and less ambiguously. To investigate the effects of different heuristics on the characteristics and predictive performance of rule-based models, we have conducted an extensive experimental study in Section 5.3. Unlike in single-label classification, the choice of a suitable rule learning heuristic is less obvious in the multi-label setting, where a variety of potentially competing evaluation measures exists. Motivated by the properties of non-decomposable evaluation measures and the desire to reveal interesting patterns in the label space, we have proposed a novel method for constructing partial multi-label rules in Section 5.4. Following prior research on this particular aspect of multi-label rule learning, it relies on guarantees provided by commonly used heuristics to prune the search for multi-label heads, which is otherwise infeasible due to high computational costs. Compared to existing approaches, the proposed methodology is more likely to discover label dependencies in practice, as it introduces a user-controllable bias towards rules that predict for several labels. The induction of multi-label rules appears to be a promising alternative to label-dependent rules. The latter have previously been used to model label dependencies. However, they rely on a particular order of the rules and thus are less in line with the properties of a DNF.

Gradient Boosting Approach

As an alternative to the SeCo-based approach in Chapter 5, we have proposed an algorithm for learning ensembles of multi-label rules in Chapter 6. It relies on a generalization of the gradient boosting framework to multivariate problems and can be viewed as another instantiation of the rule learning framework in Chapter 4. Relying on the established and theoretically well-justified gradient boosting methodology appears to be particularly appealing in the multi-label classification setting. It allows for global optimization of different target loss functions and can thus be adjusted to varying multi-label evaluation measures. Unlike most problem adaptation methods, the BOOMER algorithm presented in Chapter 6 does not only allow for the optimization of decomposable loss functions but can also deal with non-decomposable ones. As prototypical representatives of these fundamentally different types of measures, we have considered the Hamming loss and the subset 0/1 loss as objective functions for our experiments. On the one hand, as discussed in Section 6.4, differentiable surrogate losses are needed to guide the construction of rules toward optimizing a particular target measure. On the other hand, we have argued that the surrogate loss function should also be considered for inference. In Section 6.5, we have proposed two strategies to obtain predictions from a previously trained model. They are tailored to the Hamming and subset 0/1 loss, respectively. For the experiments in Section 6.6, we relied on synthetic and real-world benchmark data, as well as on recently proposed evaluation measures for the analysis of a classifier's dependence-awareness. The results have shown that the proposed methodology can address both of the considered target losses. Our analysis further suggests that complete multi-label rules provides significant advantages if one is interested in optimizing non-decomposable evaluation measures, such as the subset 0/1 loss. This is in contrast to optimizing decomposable measures like the Hamming loss, where single-label rules are most often sufficient. Compared to state-of-the-art baselines, which use different kinds of problem transformation methods, the BOOMER algorithm generally achieves strong predictive performances and is frequently able to outperform its competitors in terms of the respective target measure. Compared to the SeCo-based approach in Chapter 5, gradient boosting provides several advantages. As it aims to learn complex ensembles, where the predictions of many ensemble members are combined, it can be expected to outperfom the former in terms of predictive performance. Due to the reduced variance that comes with an ensemble method and because BOOMER can easily be tailored to different target measures, it is also more likely to produce strong predictive results out-of-the-box, i.e., without using techniques for overfitting avoidance or the need for extensive parameter tuning. On the downside, due to the large number of rules in an ensemble, it becomes nearly impossible for human analysts to comprehend a model as a whole or reason about the predictions it provides without additional tools.

Optimizations and Approximations

Motivated by the increasing need for highly scalable machine learning algorithms that can process large amounts of training data in a timely manner, we have also elaborated on implementation details that allow for the efficient induction of rules in this work. As the SeCo-based approach in Chapter 5 and the BOOMER algorithm in Chapter 6 are based on the same framework, both methods can potentially benefit from the optimizations that have been discussed in Chapter 7.[1] First of all, the principles of a pre-sorted search algorithm, as existing rule learning methods commonly use it, have been discussed in Section 7.1. Furthermore, we have discussed ways to deal with nominal and missing feature values and presented an extension to the basic algorithm that allows dealing with sparsity in the feature space more efficiently. Our experiments have shown that the ability to use sparse feature representations drastically reduces training times on many multi-label datasets. To handle large datasets with many numerical features, where the exploitation of feature sparsity does not provide any benefits, in Section 7.2, we have considered a histogram-based approximation technique as an alternative. It has previously been used in decision tree learning. According to our findings, its use reduces the training time of the BOOMER algorithm while at the same time preserving its predictive performance. In addition to the experiments mentioned above, we have also investigated the potential benefits in terms of training efficiency that result from the use of multi-threading to parallelize different aspects of a multi-label rule learning algorithm. We have found that the speedup that can be achieved through parallelization depends on the dataset and the configuration of an algorithm. In particular, it is heavily influenced by the characteristics of the evaluation measure to be dealt with. Whereas optimizing a non-decomposable measure benefit from multi-threading only in certain scenarios, there is a much larger potential for improvements if a non-decomposable target measure is used due to the greater computational demand this use case entails. Said use case is also the focus of Chapter 8, where we proposed gradient-based label binning as an approximation technique that deliberately addresses the computational bottleneck of the BOOMER algorithm. When dealing with a non-decomposable loss function, where the computation of a rule's predictions requires solving a linear system, GBLB can significantly reduce training time. By restricting the predictions to bins of similar labels rather than considering each label individually, an upper bound is imposed on this costly and frequently performed operation. The experimental study in Section 8.3 has shown that GBLB generally preserves a learner's predictive capabilities and revealed that it results in drastic improvements in some cases, most probably due to a regularization effect it introduces.

1: An efficient implementation of the SeCo algorithm in Chapter 5, which is based on the BOOMER algorithm's source code and benefits from its numerous optimizations, was still in development at the time this thesis was published. It can be obtained from the author of this thesis.

9.2 Future Work

Extensions to the SeCo Approach

Even though we have addressed several aspects crucial to the effectiveness of a SeCo-based multi-label rule learning method, there are still open questions that have not been investigated yet. In particular, due to the lack of suitable stopping criteria that work well across a various multi-label datasets, it is necessary to manually filter the rules that should be included in a model if one is interested in strong predictive performance. This contrasts with single-label classification, where established algorithms like RIPPER come with integrated pre-pruning strategies that work well on average and allow for strong out-of-the-box performance even without excessive parameter tuning. However, a generalization of existing pre-pruning techniques to the multi-label setting, e.g., the MDL stopping criterion (Quinlan, 1990), is not straightforward. Although existing techniques can be applied to several labels in a binary relevance fashion, optimizing varying target measures most probably demands different stopping or filter criteria. Furthermore, the predictive performance of decision lists, including those learned by our multi-label SeCo approach, is heavily affected by inaccurate rules that overfit the data. This is in contrast to large rule-based ensembles, where individual rules have only a small impact on the overall performance and can be rectified by other ensemble members. To overcome this problem, techniques for overfitting avoidance, such as incremental reduced error pruning (Fürnkranz and Widmer, 1994) or RIPPER's post-optimization procedure (Cohen, 1995), are typically employed by single-label rule learners. Again, a generalization of such methods to the multi-label setting remains for future research and most probably requires considering the particularities of multi-label classification. For example, when learning multi-label rules that predict for several labels, it remains unclear if a pruning strategy should only affect the conditions in a rule's body, as is the case in single-label classification, or if the predictions that are provided by a rule's head should also be adjusted. Finally, there is ongoing research on branch-and-bound algorithms (see, e.g., Angelino et al., 2018; Boley et al., 2021; Webb, 1995) that aim to overcome the search myopia of greedy rule learning methods, such those used for most of this work. Despite higher computational costs, branch-and-bound algorithms may come with advantages if compact models should be learned, as they can be expected to result in more accurate rules. However, we are not aware of any existing work where they have been applied to multi-label classification problems.

Interpretable Additive Models

As mentioned earlier, the use of the SeCo paradigm is mainly motivated by the need to learn compact models that are simple enough to be analyzed and verified by domain experts as a whole. However, the gradient boosting methodology, which we proposed as an alternative primarily focused on predictive performance, could also serve as a foundation for learning such simple models. Compared to the SeCo approach, the latter's use is appealing because it does not rely on heuristics to guide the induction of individual rules but follows a statistically well-justified procedure to optimize a target measure globally. Constraints may be imposed on the complexity of the models that result from a boosting approach to facilitate their interpretation. For example, besides limiting the number of rules, one could restrict their predictions to small integer values rather than continuous real-valued scores or even binary outputs. The main difference between such models and those learned by a SeCo algorithm is the aggregation of individual rules at prediction time. Whereas the SeCo approach results in an ordered decision list, where the first rule that covers an example is responsible for making a prediction, boosted rules are combined in an additive way by summing up their predictions and applying a threshold afterward. Which one of these representations is preferable in terms of interpretability remains unclear. However, small additive models are often argued to be inherently interpretable (C.-H. Chang et al., 2021). By enforcing the predictions for individual labels to be either positive or negative, the boosting methodology might even be used to learn models that resemble the properties of a DNF. In the single-label setting, SLIPPER (Cohen and Singer, 1999) can be considered an early attempt to make use of boosting principles for the construction of this kind of simple model. Imposing constraints on the complexity of a boosting model most likely comes with a deterioration in predictive performance because it reduces bias at the expense of variance. This presumably results in models that are more prone to overfitting. To counteract these effects, existing rule learning techniques for overfitting avoidance, such as pruning or post-optimization methods, are most probably needed in such a scenario. However, these techniques have not yet been studied extensively in the context of probabilistic models. In applications where the ability to analyze entire models is not a strict requirement, but one is merely interested in local explanations of the predictions made for individual examples, complex ensembles should probably be preferred over simple models due to their advantages in terms of predictive performance. Compared to complex statistical methods like neural networks, which are entirely intransparent and must be treated as black-boxes, a rule-based ensemble, even if it is very complex, arguably encodes a lot of valuable information post-hoc interpretation methods may directly access.

Optimization of Additional Target Measures

A major advantage of the BOOMER algorithm is its ability to learn models that are deliberately tailored to a given target measure. In particular, it is not only capable of optimizing decomposable loss functions but can also deal with non-decomposable ones. In this work, we have restricted ourselves to the Hamming and subset 0/1 loss as they are prototypical examples of these two types of multi-label evaluation measures. However, as argued in Section 3.3, both measures are rather extreme and come with practical limitations when dealing with large datasets that consist of hundreds or even thousands of labels. The subset 0/1 loss becomes increasingly difficult to optimize as the number of labels increases and often fails to provide meaningful estimates of a classifier's performance. Moreover, as the distribution of labels tends to be rather imbalanced in such use cases (cf. Section 3.2), individual examples are often associated with only a small fraction of the available labels. As a result, the Hamming loss is dominated mainly by irrelevant labels, and models that perform well for this particular measure often suffer from low recall because negative predictions have a higher chance of being correct than positive ones. Other measures, such as the example-wise F1-measure or the Jaccard index (see, e.g., Gouk, Pfahringer, and Cree, 2016, for a definition), appear better suited in the scenario mentioned above. For example, the F1-measure strives to balance precision and recall and therefore penalizes classifiers that predict too cautiously for relevant labels. By implementing a suitable surrogate loss function and an appropriate prediction strategy, the BOOMER algorithm may be extended by the ability to optimize the example-wise F1-measure. Surrogates for the F1-measure have previously been proposed for use in logistic regression (M. Zhang, Ramaswamy, and Agarwal, 2020) or deep neural networks (Bénédict et al., 2021). The problem of inferring binary predictions from probabilistic classification models, such that they are optimal with respect to the example-wise F1-measure, has also been studied quite extensively in the literature (see, e.g., Dembczyński, Jachnik, et al., 2013; Dembczyński, Waegeman, et al., 2011).

Exploitation of Label Sparsity

A significant amount of the BOOMER algorithm's training time is spent updating and aggregating the label space statistics, i.e., the gradients and Hessians, that correspond to individual examples and labels. Compared to using a decomposable loss function, where the number of label space statistics depends linearly on the number of labels, this is especially problematic if a non-decomposable loss function should be optimized because the number of statistics grows quadratically in this case. As shown in Chapter 8, gradient-based label binning helps to diminish the computational costs of deriving loss-minimizing predictions from the quadratic number of statistics. Nevertheless, the minimization of non-decomposable losses remains costly. Our analysis has revealed that the computation and aggregation of statistics become the new computational bottleneck when GBLB is used. Therefore, reducing the number of statistics that the algorithm must deal with is inevitable if one is interested in applying it to large-scale datasets that include many labels. Fortunately, there is prior research that might help to achieve this goal. Si et al. (2017) propose to exploit the fact that large multi-label datasets typically come with high label sparsity. If this is the case, a simple default rule, which predicts each label as irrelevant, already predicts correctly in most cases. Consequently, the remaining rules can be focused on regions of the label space where positive predictions are needed. By using surrogate loss functions that fulfill specific mathematical properties, it can be ensured that their gradients and Hessians evaluate to zero if a label is already correctly dealt with by the default rule. This enables the use of sparse data structures to store the label space statistics and reduces the computational costs needed to update and aggregate them. In addition, to maintain a high degree of sparsity among the statistics even after new rules have been learned, individual rules should be restricted to small subregions of the label space. This prohibits the use of complete rule heads that predict for all available labels and thus are likely to introduce a large number of non-zero statistics. Instead, a method for inducing partial multi-label rules is required. Although the work of Si et al. (2017), which was partly adopted by Z. Zhang and C. Jung (2020), is restricted to decomposable loss functions and assumes first-order gradient boosting to be used, it may serve as a foundation for enhancing the BOOMER algorithm's computational capabilities. Besides a methodology for the induction of partial rules, this requires the investigation of non-decomposable surrogate loss functions that fulfill the properties above.

Use of Dimensionality Reduction Methods

As mentioned in Section 3.2, research on multi-label classification has lately shifted towards applications with vast numbers of labels. Rather than focusing on this emerging topic, which is referred to as extreme multi-label classification, we have restricted ourselves to smaller datasets that can be tackled without complexity reduction methods. However, in the future, it might be desirable to apply the methods developed in this thesis, in particular the BOOMER algorithm, to such large-scale problems. As BOOMER is a problem adaptation method that can deal with multiple labels directly, i.e., without the need for problem transformation approaches, it appears to be well-suited as a base learner in label space transformation methods, such as those discussed in Section 3.8. For example, this includes HOMER, which transforms a multi-label problem into a hierarchy of smaller MLC tasks, or label embedding methods that map a high-dimensional label space to a smaller subspace. Unlike the problem adaptation methods typically used by said dimensionality reduction methods (cf. Section 3.7), BOOMER is capable of optimizing non-decomposable loss functions. An experimental comparison to existing baselines may reveal whether this ability is advantageous when applied to label subspaces. Another possible direction for future research is the integration of dimensionality reduction methods into the rule learning framework presented in Chapter 4. The framework provides means to restrict individual rules to predetermined subsets of the available features, examples, or labels. Strategies for selecting random subsets of the available features or examples have partly been used for the experiments in this work, as they typically speed up training and result in more diverse rules being learned. However, one could also investigate more sophisticated methods that take information about the feature values of the training examples or the distribution of their ground truth labels into account. This does not only include existing feature selection methods that have deliberately been designed for multi-label problems but also enables rules to be tailored to specific subsets of the training data. The latter appears to be especially interesting when combined with methods for label selection. By identifying regions in the feature space where certain combinations of labels occur more frequently, one could learn rules that are deliberately tailored to the respective subregion. Other labels, which are ideally known to be uncorrelated with the considered labels within the given subregion, can be ignored, as even the optimization of non-decomposable loss functions cannot be expected to benefit from taking interactions between them into account.

Comparison of Boosted Trees vs. Rules

In this work, we have preferred rule-based model representations over the closely related decision trees, not only due to their advantages in terms of interpretability, but also as ensemble members in our gradient boosting approach, where interpretability is not a key aspect. Even though we have shown experimentally that the use of rules in a gradient boosting setting can compete with gradient boosted decision trees in terms of predictive performance, it remains unclear whether one of these approaches provides significant advantages over the other. On the one hand, both types of models rely on conditional statements that test for the feature values of given examples and thus are restricted to decision boundaries parallel to the axes of the feature space. Hence, it appears unlikely that one approach is significantly more potent than the other, given that the hyper-parameters of the respective learning algorithm are chosen appropriately. On the other hand, decision trees are global models that provide predictions for any given example, whereas a single rule focuses on a particular region of the feature space. Compared to the combination of global models, as is the case in gradient boosted decision trees, rules might allow more efficient use of computational resources by learning more rules for regions where accurate predictions are difficult and fewer rules for regions that are easier to model. Such behavior has also been found to be beneficial for computational efficiency by the authors of LightGBM (Ke et al., 2017). Based on prior work by Shi (2007), their GBDT method uses a "best-first" strategy, where only a few branches of a decision tree — which can be viewed as individual rules — are grown to the full extent, whereas less computational effort is put into others. Nevertheless, a systematic comparison between the use of decision trees and probabilistic rules in gradient boosting methods, with a focus on computational efficiency, remains for future work.

References

Agrawal, Rahul, Archit Gupta, Yashoteja Prabhu, and Manik Varma (2013). 'Multi-Label Learning with Millions of Labels: Recommending Advertiser Bid Phrases for Web Pages'. In: *Proc. International World Wide Web Conference (WWW)*, pp. 13–24.

Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami (1993). 'Mining Association Rules between Sets of Items in Large Databases'. In: *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 207–216.

Allamanis, Miltiadis, Fani A. Tzima, and Pericles A. Mitkas (2013). 'Effective Rule-Based Multi-label Classification with Learning Classifier Systems'. In: *Proc. International Conference on Adaptive and Natural Computing Algorithms*, pp. 466–476.

Alsabti, Khaled, Sanjay Ranka, and Vineet Singh (1998). 'CLOUDS: A Decision Tree Classifier for Large Datasets'. In: *Proc. International Conference on Knowledge Discovery and Data Mining*, pp. 2–8.

Amit, Yonatan, Ofer Dekel, and Yoram Singer (2007). 'A Boosting Algorithm for Label Covering in Multilabel Problems'. In: *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 27–34.

Anderson, Edward, Zhaojun Bai, Christian Bischof, L. Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, et al. (1999). *LAPACK Users' guide*. SIAM.

Angelino, Elaine, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin (2018). 'Learning Certifiably Optimal Rule Lists for Categorical Data'. In: *Journal of Machine Learning Research* 18, pp. 1–78.

Angluin, Dana (1988). 'Queries and Concept Learning'. In: *Machine Learning* 2.4, pp. 319–342.

Arunadevi, J. and V. Rajamani (2011). 'An Evolutionary Multi Label Classification using Associative Rule Mining for Spatial Preferences'. In: *IJCA Special Issue on Artificial Intelligence* 3, pp. 28–37.

Ávila-Jiménez, José Luis, Eva L. Gibaja, and Sebastián Ventura (2010). 'Evolving Multi-label Classification Rules with Gene Expression Programming: A Preliminary Study'. In: *Proc. International Conference on Hybrid Artificial Intelligence Systems (HAIS)*, pp. 9–16.

Balasubramanian, Krishnakumar and Guy Lebanon (2012). 'The Landmark Selection Method for Multiple Output Prediction'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 283–290.

Bénard, Clément, Gérard Biau, Sébastien Da Veiga, and Erwan Scornet (2021). 'SIRUS: Stable and Interpretable RUle Set for classification'. In: *Electronic Journal of Statistics* 15.1, pp. 427–505.

Bénédict, Gabriel, Vincent Koops, Daan Odijk, and Maarten de Rijke (2021). 'sigmoidF1: A Smooth F1 Score Surrogate Loss for Multilabel Classification'. In: *arXiv preprint arXiv:2108.10566.*

Bhatia, Kush, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain (2015). 'Sparse Local Embeddings for Extreme Multi-Label Classification'. In: *Proc. International Conference on Neural Information Processing Systems (NIPS)*, pp. 730–738.

Bi, Wei and James Kwok (2013). 'Efficient Multi-label Classification with Many Labels'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 405–413.

Bishop, Christopher M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press.

Blackford, L. Susan, Antoine Petitet, Roldan Pozo, Karin Remington, R. Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. (2002). 'An updated set of basic linear algebra subprograms (BLAS)'. In: *ACM Transactions on Mathematical Software* 28.2, pp. 135–151.

Blockeel, Hendrik, Luc De Raedt, and Jan Ramon (1998). 'Top-down induction of clustering trees'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 55–63.

Bohlender, Simon, Eneldo Loza Mencía, and Moritz Kulessa (2020). 'Extreme Gradient Boosted Multi-label Trees for Dynamic Classifier Chains'. In: *Proc. International Conference on Discovery Science*, pp. 471–485.

Boley, Mario, Simon Teshuva, Pierre Le Bodic, and Geoffrey I. Webb (2021). 'Better Short than Greedy: Interpretable Models through Optimal Rule Boosting'. In: *Proc. SIAM International Conference on Data Mining*, pp. 351–359.

Boström, Henrik (1995). 'Covering vs. Divide-and-Conquer for Top-Down Induction of Logic Programs'. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1194–1200.

Boutell, Matthew R., Jiebo Luo, Xipeng Shen, and Christopher M. Brown (2004). 'Learning multi-label scene classification'. In: *Pattern Recognition* 37.9, pp. 1757–1771.

Breiman, Leo (1996). *Bias, Variance, and Arcing Classifiers*. Tech. rep. Technical Report 460, Statistics Department, University of California, Berkeley.

Breiman, Leo (2001). 'Random Forests'. In: *Machine Learning* 45.1, pp. 5–32.

Breiman, Leo, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone (1984). *Classification and Regression Trees*. Wadsworth & Brooks.

Briggs, Forrest, Yonghong Huang, Raviv Raich, Konstantinos Eftaxias, Zhong Lei, William Cukierski, Sarah Frey Hadley, Adam Hadley, Matthew Betts, Xiaoli Z Fern, et al. (2013). 'The 9th annual MLSP competition: New methods for acoustic classification of multiple simultaneous bird species in a noisy environment'. In: *Proc. IEEE International Workshop on Machine Learning for Signal Processing*, pp. 1–8.

Cano, Alberto, Amelia Zafra, Eva L. Gibaja, and Sebastián Ventura (2013). 'A Grammar-Guided Genetic Programming Algorithm for Multi-Label Classification'. In: *Proc. European Conference on Genetic Programming*, pp. 217–228.

Cestnik, Bojan (1990). 'Estimating probabilities: A crucial task in machine learning'. In: *Proc. European Conference on Artificial Intelligence*, pp. 147–150.

Chang, Chih-Chung and Chih-Jen Lin (2011). 'LIBSVM: A Library for Support Vector Machines'. In: *ACM Transactions on Intelligent Systems and Technology* 2.3, pp. 1–27.

Chang, Chun-Hao, Sarah Tan, Ben Lengerich, Anna Goldenberg, and Rich Caruana (2021). 'How Interpretable and Trustworthy are GAMs?' In: *Proc. ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 95–105.

Charte, Francisco, Antonio J. Rivera, María José del Jesus, and Francisco Herrera (2012). 'Improving Multi-label Classifiers via Label Reduction with Association Rules'. In: *Proc. International Conference on Hybrid Artificial Intelligence Systems (HAIS)*, pp. 188–199.

Charte, Francisco, Antonio J. Rivera, María José del Jesus, and Francisco Herrera (2013). 'A First Approach to Deal with Imbalance in Multi-label Datasets'. In: *Proc. International Conference on Hybrid Artificial Intelligence Systems (HAIS)*, pp. 150–160.

Charte, Francisco, Antonio J. Rivera, María José del Jesus, and Francisco Herrera (2019). 'REMEDIAL-HwR: Tackling multilabel imbalance through label decoupling and data resampling hybridization'. In: *Neurocomputing* 326-327, pp. 110–122.

Chen, Tianqi and Carlos Guestrin (2016). 'XGBoost: A Scalable Tree Boosting System'. In: *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794.

Chen, Yao-Nan and Hsuan-Tien Lin (2012). 'Feature-aware Label Space Dimension Reduction for Multi-Label Classification'. In: *Proc. International Conference on Neural Information Processing Systems (NIPS)*, pp. 1529–1537.

Cheng, Weiwei, Krzysztof Dembczyński, Eyke Hüllermeier, Adrian Jaroszewicz, and Willem Waegeman (2012). 'F-Measure Maximization in Topical Classification'. In: *Proc. International Conference on Rough Sets and Current Trends in Computing*, pp. 439–446.

Choquet, Gustave (1954). 'Theory of capacities'. In: *Annales de l'institut Fourier*. Vol. 5, pp. 131–295.

Chua, Tat-Seng, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng (2009). 'NUS-WIDE: A Real-World Web Image Database from National University of Singapore'. In: *Proc. ACM International Conference on Image and Video Retrieval*, pp. 1–9.

Cissé, Moustapha, Nicolas Usunier, Thierry Artieres, and Patrick Gallinari (2013). 'Robust Bloom Filters for Large Multilabel Classification Tasks'. In: *Proc. International Conference on Neural Information Processing Systems (NIPS)*, pp. 1851–1859.

Clare, Amanda and Ross D. King (2001). 'Knowledge Discovery in Multi-label Phenotype Data'. In: *Proc. European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, pp. 42–53.

Clark, Peter and Robin Boswell (1991). 'Rule Induction with CN2: Some Recent Improvements'. In: *Proc. European Working Session on Learning*, pp. 151–163.

Cohen, William W. (1995). 'Fast Effective Rule Induction'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 115–123.

Cohen, William W. and Yoram Singer (1999). 'A Simple, Fast, and Effective Rule Learner'. In: *Proc. AAAI Conference on Artificial Intelligence*, pp. 335–342.

Deerwester, Scott, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman (1990). 'Indexing by Latent Semantic Analysis'. In: *Journal of the American Society for Information Science* 41.6, pp. 391–407.

Dembczyński, Krzysztof, Weiwei Cheng, and Eyke Hüllermeier (2010). 'Bayes Optimal Multilabel Classification via Probabilistic Classifier Chains'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 279–286.

Dembczyński, Krzysztof, Arkadiusz Jachnik, Wojciech Kotłowski, Willem Waegeman, and Eyke Hüllermeier (2013). 'Optimizing the F-measure in multi-label classification: Plug-in rule approach versus structured loss minimization'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 1130–1138.

Dembczyński, Krzysztof, Wojciech Kotłowski, and Eyke Hüllermeier (2012). 'Consistent Multilabel Ranking through Univariate Losses'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 1319–1326.

Dembczyński, Krzysztof, Wojciech Kotłowski, and Roman Słowiński (2010). 'ENDER: A statistical framework for boosting decision rules'. In: *Data Mining and Knowledge Discovery* 21.1, pp. 52–90.

Dembczyński, Krzysztof, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier (2011). 'An Exact Algorithm for F-Measure Maximization'. In: *Proc. International Conference on Neural Information Processing Systems (NIPS)*, pp. 1404–1412.

Dembczyński, Krzysztof, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier (2012). 'On label dependence and loss minimization in multi-label classification'. In: *Machine Learning* 88.1-2, pp. 5–45.

Demšar, Janez (2006). 'Statistical Comparisons of Classifiers over Multiple Data Sets'. In: *Journal of Machine Learning Research* 7, pp. 1–30.

Deng, Jia, Alexander C. Berg, Kai Li, and Li Fei-Fei (2010). 'What Does Classifying More Than 10,000 Image Categories Tell Us?' In: *Proc. European Conference on Computer Vision (ECCV)*, pp. 71–84.

Denœux, Thierry (2006). 'The cautious rule of combination for belief functions and some extensions'. In: *Proc. International Conference on Information Fusion*, pp. 1–8.

Denton, Emily, Jason Weston, Manohar Paluri, Lubomir Bourdev, and Rob Fergus (2015). 'User Conditional Hashtag Prediction for Images'. In: *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1731–1740.

Diao, Lili, Keyun Hu, Yuchang Lu, and Chunyi Shi (2002). 'Boosting Simple Decision Trees with Bayesian Learning for Text Categorization'. In: *Proc. World Congress on Intelligent Control and Automation*, pp. 321–325.

Diplaris, Sotiris, Grigorios Tsoumakas, Pericles A. Mitkas, and Ioannis Vlahavas (2005). 'Protein Classification with Multiple Algorithms'. In: *Proc. Panhellenic Conference on Informatics*, pp. 448–456.

Doquire, Gauthier and Michel Verleysen (2011). 'Feature Selection for Multi-label Classification Problems'. In: *Proc. International Work-conference on Artificial Neural Networks (IWAN)*, pp. 9–16.

Du, Mengnan, Ninghao Liu, and Xia Hu (2019). 'Techniques for interpretable machine learning'. In: *Communications of the ACM* 63.1, pp. 68–77.

Duda, Richard O., Peter E. Hart, and David G. Stork (2000). *Pattern Classification*. John Wiley and Sons.

Elisseeff, André and Jason Weston (2001). 'A kernel method for multi-labelled classification'. In: *Proc. International Conference on Neural Information Processing Systems (NIPS)*. Vol. 14, pp. 681–687.

Esuli, Andrea, Tiziano Fagni, and Fabrizio Sebastiani (2006). 'MP-Boost: A Multiple-Pivot Boosting Algorithm and its Application to Text Categorization'. In: *Proc. International Symposium on String Processing and Information Retrieval*, pp. 1–12.

Freund, Yoav and Robert E. Schapire (1997). 'A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting'. In: *Journal of Computer and System Sciences* 55.1, pp. 119–139.

Friedman, Jerome H. (2002). 'Stochastic gradient boosting'. In: *Computational statistics & data analysis* 38.4, pp. 367–378.

Friedman, Jerome H., Trevor Hastie, and Robert Tibshirani (2000). 'Additive Logistic Regression: A Statistical View of Boosting'. In: *The Annals of Statistics* 28.2, pp. 337–407.

Friedman, Jerome H. and Bogdan E. Popescu (2008). 'Predictive learning via rule ensembles'. In: *The Annals of Applied Statistics* 2.3, pp. 916–954.

Friedman, Milton (1937). 'The use of ranks to avoid the assumption of normality implicit in the analysis of variance'. In: *Journal of the American Statistical Association* 32.200, pp. 675–701.

Friedman, Milton (1940). 'A comparison of alternative tests of significance for the problem of m rankings'. In: *The Annals of Mathematical Statistics* 11.1, pp. 86–92.

Fürnkranz, Johannes (1994). 'FOSSIL: A Robust Relational Learner'. In: *Proc. European Conference on Machine Learning (ECML)*, pp. 122–137.

Fürnkranz, Johannes (1997). 'Pruning Algorithms for Rule Learning'. In: *Machine Learning* 27.2, pp. 139–172.

Fürnkranz, Johannes (1999). 'Separate-and-Conquer Rule Learning'. In: *Artificial Intelligence Review* 13.1, pp. 3–54.

Fürnkranz, Johannes (2002). 'A Pathology of Bottom-Up Hill-Climbing in Inductive Rule Learning'. In: *Proc. International Conference on Algorithmic Learning Theory*, pp. 263–277.

Fürnkranz, Johannes (2005). 'From Local to Global Patterns: Evaluation Issues in Rule Learning Algorithms'. In: *Local Pattern Detection*, pp. 20–38.

Fürnkranz, Johannes and Peter A. Flach (2004). 'An Analysis of Stopping and Filtering Criteria for Rule Learning'. In: *Proc. European Conference on Machine Learning (ECML)*, pp. 123–133.

Fürnkranz, Johannes and Peter A. Flach (2005). 'ROC 'n' Rule Learning — Towards a Better Understanding of Covering Algorithms'. In: *Machine learning* 58.1, pp. 39–77.

Fürnkranz, Johannes, Dragan Gamberger, and Nada Lavrač (2012). *Foundations of Rule Learning*. Springer Science & Business Media.

Fürnkranz, Johannes, Tomáš Kliegr, and Heiko Paulheim (2020). 'On cognitive preferences and the plausibility of rule-based models'. In: *Machine Learning* 109.4, pp. 853–898.

Fürnkranz, Johannes and Gerhard Widmer (1994). 'Incremental Reduced Error Pruning'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 70–77.

Gamberger, Dragan and Nada Lavrač (2000). 'Confirmation Rule Sets'. In: *Proc. European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, pp. 34–43.

Gao, Wei and Zhi-Hua Zhou (2013). 'On the Consistency of Multi-Label Learning'. In: vol. 199-200, pp. 22–44.

Gibaja, Eva L. and Sebastián Ventura (2014). 'Multi-Label Learning: A Review of the State of the Art and Ongoing Research'. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4.6, pp. 411–444.

Gibney, Elizabeth (2016). 'Google AI algorithm masters ancient game of Go'. In: *Nature* 529.7587, pp. 445–446.

Gjorgjioski, Valentin, Dragi Kocev, and Sašo Džeroski (2011). 'Comparison of distances for multi-label classification with PCTs'. In: *Proc. Slovenian Conference on Data Mining and Data Warehouses*.

Gonçalves, Eduardo Corrêa, Alexandre Plastino, and Alex A. Freitas (2013). 'A Genetic Algorithm for Optimizing the Label Ordering in Multi-label Classifier Chains'. In: *Proc. IEEE International Conference on Tools with Artificial Intelligence*, pp. 469–476.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.

Gouk, Henry, Bernhard Pfahringer, and Michael Cree (2016). 'Learning Distance Metrics for Multi-Label Classification'. In: *Asian Conference on Machine Learning*, pp. 318–333.

Guzella, Thiago S. and Walmir M. Caminhas (2009). 'A review of machine learning approaches to Spam filtering'. In: *Expert Systems with Applications* 36.7, pp. 10206–10222.

Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten (2009). 'The WEKA Data Mining Software: An Update'. In: *SIGKDD Explorations* 11.1, pp. 10–18.

Han, Jiawei, Jian Pei, and Yiwen Yin (2000). 'Mining Frequent Patterns Without Candidate Ceneration'. In: *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 1–12.

Hastie, Trevor, Robert Tibshirani, and Jerome H. Friedman (2009). *The Elements of Statistical Learning*. Springer Science & Business Media.

Hoy, Matthew B. (2018). 'Alexa, Siri, Cortana, and more: An introduction to voice assistants'. In: *Medical Reference Services Quarterly* 37.1, pp. 81–88.

Hsu, Daniel, Sham M. Kakade, John Langford, and Tong Zhang (2009). 'Multi-Label Prediction via Compressed Sensing'. In: *Proc. International Conference on Neural Information Processing Systems (NIPS)*, pp. 772–780.

Hsu, Feng-Hsiung (2002). *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press.

Huang, Kuan-Hao and Hsuan-Tien Lin (2017). 'Cost-sensitive label embedding for multi-label classification'. In: *Machine Learning* 106.9, pp. 1725–1746.

Hüllermeier, Eyke, Johannes Fürnkranz, Eneldo Loza Mencía, Vu-Linh Nguyen, and Michael Rapp (2020). 'Rule-based Multi-label Classification: Challenges and Opportunities'. In: *International Joint Conference on Rules and Reasoning*, pp. 3–19.

Hüllermeier, Eyke and Willem Waegeman (2021). 'Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods'. In: *Machine Learning* 110.3, pp. 457–506.

Hüllermeier, Eyke, Marcel Wever, Eneldo Loza Mencía, Johannes Fürnkranz, and Michael Rapp (2022). 'A flexible class of dependence-aware multi-label loss functions'. In: *Machine Learning* 111.2, pp. 713–737.

Janssen, Frederik and Johannes Fürnkranz (2008). 'An Empirical Investigation of the Trade-Off Between Consistency and Coverage in Rule Learning Heuristics'. In: *Proc. International Conference on Discovery Science*, pp. 40–51.

Janssen, Frederik and Johannes Fürnkranz (2010). 'On the quest for optimal rule learning heuristics'. In: *Machine Learning* 78.3, pp. 343–379.

Jasinska, Kalina, Krzysztof Dembczyński, Róbert Busa-Fekete, Karlson Pfannschmidt, Timo Klerx, and Eyke Hüllermeier (2016). 'Extreme F-Measure Maximization using Sparse Probability Estimates'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 1435–1444.

Jian, Ling, Jundong Li, Kai Shu, and Huan Liu (2016). 'Multi-Label Informed Feature Selection'. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1627–33.

Jin, Ruoming and Gagan Agrawal (2003). 'Communication and Memory Efficient Parallel Decision Tree Construction'. In: *Proc. SIAM International Conference on Data Mining*, pp. 119–129.

Joachims, Thorsten (1998). 'Text Categorization with Support Vector Machines: Learning with Many Relevant Features'. In: *Proc. European Conference on Machine Learning (ECML)*, pp. 137–142.

Johnson, Matthew and Roberto Cipolla (2005). 'Improved Image Annotation and Labelling through Multi-label Boosting'. In: *Proc. British Machine Vision Conference (BMVC)*.

Jung, Young Hun and Ambuj Tewari (2018). 'Online Boosting Algorithms for Multi-label Ranking'. In: *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 279–287.

Kamath, Chandrika, Erick Cantú-Paz, and David Littau (2002). 'Approximate Splitting for Ensembles of Trees using Histograms'. In: *Proc. SIAM International Conference on Data Mining*, pp. 370–383.

Katakis, Ioannis, Grigorios Tsoumakas, and Ioannis Vlahavas (2008). 'Multilabel Text Classification for Automated Tag Suggestion'. In: *Proc. ECML-PKDD 2008 Discovery Challenge*. Vol. 18, p. 5.

Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu (2017). 'LightGBM: A Highly Efficient Gradient Boosting Decision Tree'. In: *Advances in Neural Information Processing Systems* 30, pp. 3146–3154.

Kimura, Keigo, Mineichi Kudo, and Lu Sun (2016). 'Simultaneous Nonlinear Label-Instance Embedding for Multi-label Classification'. In: *Proc. Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition and Structural and Syntactic Pattern Recognition*, pp. 15–25.

Klein, Yannik, Michael Rapp, and Eneldo Loza Mencía (2019). 'Efficient Discovery of Expressive Multi-label Rules using Relaxed Pruning'. In: *Proc. International Conference on Discovery Science*, pp. 367–382.

Kocev, Dragi, Celine Vens, Jan Struyf, and Sašo Džeroski (2007). 'Ensembles of Multi-Objective Decision Trees'. In: *Proc. European Conference on Machine Learning (ECML)*, pp. 624–631.

Kotsiantis, Sotiris B. (2007). 'Supervised Machine Learning: A Review of Classification Techniques'. In: *Emerging Artificial Intelligence Applications in Computer Engineering* 160.1, pp. 3–24.

Kotsiantis, Sotiris B. and Dimitris Kanellopoulos (2006). 'Discretization Techniques: A recent survey'. In: *GESTS International Transactions on Computer Science and Engineering* 32.1, pp. 47–58.

Kulessa, Moritz and Eneldo Loza Mencía (2018). 'Dynamic Classifier Chain with Random Decision Trees'. In: *Proc. International Conference on Discovery Science*, pp. 33–50.

Kumar, Vikas, Arun K. Pujari, Vineet Padmanabhan, and Venkateswara Rao Kagita (2019). 'Group preserving label embedding for multi-label classification'. In: *Pattern Recognition* 90, pp. 23–34.

Lakkaraju, Himabindu, Stephen H. Bach, and Jure Leskovec (2016). 'Interpretable Decision Sets: A Joint Framework for Description and Prediction'. In: *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1675–1684.

Lang, Ken (2008). *The 20 newsgroups data set*. URL: http://qwone.com/~jason/20Newsgroups. (Accessed on May 18, 2021).

Langley, Pat (1996). *Elements of Machine Learning*. Morgan Kaufmann.

Lewis, David D., Yiming Yang, Tony Russell-Rose, and Fan Li (2004). 'RCV1: A New Benchmark Collection for Text Categorization Research'. In: *Journal of Machine Learning Research* 5, pp. 361–397.

Li, Bo, Hong Li, Min Wu, and Ping Li (2008). 'Multi-label Classification based on Association Rules with Application to Scene Classification'. In: *Proc. International Conference for Young Computer Scientists*, pp. 36–41.

Li, Ping, Qiang Wu, and Christopher Burges (2007). 'McRank: Learning to Rank Using Multiple Classification and Gradient Boosting'. In: *Advances in Neural Information Processing Systems* 20.

Li, Yachong and Youlong Yang (2020). 'Label Embedding for Multi-label Classification Via Dependence Maximization'. In: *Neural Processing Letters* 52.2, pp. 1651–1674.

Lin, Yaojin, Qinghua Hu, Jinghua Liu, Jinkun Chen, and Jie Duan (2016). 'Multi-label feature selection based on neighborhood mutual information'. In: *Applied Soft Computing* 38, pp. 244–256.

Lin, Yaojin, Qinghua Hu, Jinghua Liu, and Jie Duan (2015). 'Multi-label feature selection based on max-dependency and min-redundancy'. In: *Neurocomputing* 168, pp. 92–103.

Lin, Zijia, Guiguang Ding, Mingqing Hu, and Jianmin Wang (2014). 'Multi-label Classification via Feature-aware Implicit Label Space Encoding'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 325–333.

Liu, Bin, Konstantinos Blekas, and Grigorios Tsoumakas (2022). 'Multi-label sampling based on local label imbalance'. In: *Pattern Recognition* 122, p. 108294.

Liu, Bin and Grigorios Tsoumakas (2020). 'Dealing with class imbalance in classifier chains via random undersampling'. In: *Knowledge-Based Systems* 192, p. 105292.

Loza Mencía, Eneldo and Johannes Fürnkranz (2008). 'Efficient Pairwise Multilabel Classification for Large-Scale Problems in the Legal Domain'. In: *Proc. European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pp. 50–65.

Loza Mencía, Eneldo, Johannes Fürnkranz, Eyke Hüllermeier, and Michael Rapp (2018). 'Learning Interpretable Rules for Multi-Label Classification'. In: *Explainable and Interpretable Models in Computer Vision and Machine Learning*, pp. 81–113.

Loza Mencía, Eneldo and Frederik Janssen (2016). 'Learning rules for multi-label classification: A stacking and a separate-and-conquer approach'. In: *Machine Learning* 105.1, pp. 77–126.

Loza Mencía, Eneldo, Moritz Kulessa, Simon Bohlender, and Johannes Fürnkranz (2022). 'Tree-based dynamic classifier chains'. In: *Machine Learning*, pp. 1–37.

Lu, Yi (1996). 'Knowledge Integration in a Multiple Classifier System'. In: *Applied Intelligence* 6.2, pp. 75–86.

Luo, Xiao and A. Nur Zincir-Heywood (2005). 'Evaluation of Two Systems on Multi-class Multi-label Document Classification'. In: *Proc. International Symposium on Methodologies for Intelligent Systems*, pp. 161–169.

Madjarov, Gjorgji, Dragi Kocev, Dejan Gjorgjevikj, and Sašo Džeroski (2012). 'An extensive experimental comparison of methods for multi-label learning'. In: *Pattern Recognition* 45.9, pp. 3084–3104.

Martin, Louise, Raymond Leblanc, and Nguyen Ky Toan (1993). 'Tables for the Friedman rank test'. In: *Canadian Journal of Statistics* 21.1, pp. 39–43.

McCarthy, John (1968). 'Programs with Common Sense'. In: *Semantic Information Processing*, pp. 403–418.

Mehra, Neha and Surendra Gupta (2013). 'Survey on Multiclass Classification Methods'. In: *International Journal of Computer Science and Information Technologies* 4.4, pp. 572–576.

Mehta, Manish, Rakesh Agrawal, and Jorma Rissanen (1996). 'SLIQ: A Fast Scalable Classifier for Data Mining'. In: *Proc. International Conference on Extending Database Technology*, pp. 18–32.

Merrillees, Maximillian and Lan Du (2021). 'Stratified Sampling for Extreme Multi-label Data'. In: *Proc. Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pp. 334–345.

Michalski, Ryszard S. (1980). 'Pattern Recognition as Rule-Guided Inductive Inference'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2.4, pp. 349–361.

Mitchell, Thomas M. (1997). *Machine Learning*. McGraw Hill.

Molnar, Christoph, Giuseppe Casalicchio, and Bernd Bischl (2020). 'Interpretable Machine Learning – A Brief History, State-of-the-Art and Challenges'. In: *Proc. European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pp. 417–431.

Montañes, Elena, Robin Senge, Jose Barranquero, José Ramón Quevedo, Juan José del Coz, and Eyke Hüllermeier (2014). 'Dependent binary relevance models for multi-label classification'. In: *Pattern Recognition* 47.3, pp. 1494–1508.

Mori, Shunji, Hirobumi Nishida, and Hiromitsu Yamada (1999). *Optical Character Recognition*. John Wiley and Sons.

Moyano, Jose M., Eva L. Gibaja, and Sebastián Ventura (2017). 'MLDA: A tool for analyzing multi-label datasets'. In: *Knowledge-Based Systems* 121, pp. 1–3.

Murdoch, W. James, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu (2019). 'Definitions, methods, and applications in interpretable machine learning'. In: *Proceedings of the National Academy of Sciences* 116.44, pp. 22071–22080.

Nam, Jinseok, Eneldo Loza Mencía, Hyunwoo J. Kim, and Johannes Fürnkranz (2017). 'Maximizing Subset Accuracy with Recurrent Neural Networks in Multi-label Classification'. In: *Proc. International Conference on Neural Information Processing Systems (NIPS)*, pp. 5419–5429.

Nemenyi, Peter Björn (1963). 'Distribution-Free Multiple Comparisons'. PhD thesis. Princeton University.

Nguyen, Vu-Linh and Eyke Hüllermeier (2020). 'Reliable Multilabel Classification: Prediction with Partial Abstention'. In: *Proc. AAAI Conference on Artificial Intelligence*, pp. 5264–5271.

Nguyen, Vu-Linh, Eyke Hüllermeier, Michael Rapp, Eneldo Loza Mencía, and Johannes Fürnkranz (2020). 'On Aggregation in Ensembles of Multilabel Classifiers'. In: *Proc. International Conference on Discovery Science*, pp. 533–547.

Pagallo, Giulia and David Haussler (1990). 'Boolean Feature Discovery in Empirical Learning'. In: *Machine Learning* 5.1, pp. 71–99.

Papagiannopoulou, Christina, Grigorios Tsoumakas, and Ioannis Tsamardinos (2015). 'Discovering and Exploiting Deterministic Label Relationships in Multi-label Learning'. In: *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 915–924.

Park, Cheong Hee and Moonhwi Lee (2008). 'On applying linear discriminant analysis for multi-labeled problems'. In: *Pattern Recognition Letters* 29.7, pp. 878–887.

Park, Sang-Hyeun and Johannes Fürnkranz (2008). 'Multi-Label Classification with Label Constraints'. In: *Proc. ECML-PKDD 2008 Workshop on Preference Learning*, pp. 157–171.

Pereira, Rafael B., Alexandre Plastino, Bianca Zadrozny, and Luiz H. C. Merschmann (2018). 'Categorizing feature selection methods for multi-label classification'. In: *Artificial Intelligence Review* 49.1, pp. 57–78.

Pestian, John P., Christopher Brew, Pawel Matykiewicz, DJ Hovermale, Neil Johnson, K Bretonnel Cohen, and Wlodzislaw Duch (2007). 'A Shared Task Involving Multi-label Classification of Clinical Free Text'. In: *Proc. Biological, Translational, and Clinical Language Processing*, pp. 97–104.

Prabhu, Yashoteja, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma (2018). 'Parabel: Partitioned Label Trees for Extreme Classification with Application to Dynamic Search Advertising'. In: *Proc. International World Wide Web Conference (WWW)*, pp. 993–1002.

Prabhu, Yashoteja and Manik Varma (2014). 'FastXML: A Fast, Accurate and Stable Tree-classifier for extreme Multi-label Learning'. In: *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 263–272.

Prajapati, Purvi and Amit Thakkar (2019). 'Extreme multi-label learning: A large scale classification approach in machine learning'. In: *Journal of Information and Optimization Sciences* 40.4, pp. 983–1001.

Provost, Foster and Pedro Domingos (2003). 'Tree Induction for Probability-Based Ranking'. In: *Machine Learning* 52.3, pp. 199–215.

Quinlan, J. Ross (1986). 'Induction of Decision Trees'. In: *Machine Learning* 1.1, pp. 81–106.

Quinlan, J. Ross (1990). 'Learning Logical Definitions from Relations'. In: *Machine Learning* 5.3, pp. 239–266.

Quinlan, J. Ross (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Rapp, Michael (2021). 'BOOMER – An Algorithm for Learning Gradient Boosted Multi-label Classification Rules'. In: *Software Impacts* 10, p. 100137.

Rapp, Michael, Moritz Kulessa, Eneldo Loza Mencía, and Johannes Fürnkranz (2021). 'Correlation-based Discovery of Disease Patterns for Syndromic Surveillance'. In: *Frontiers in Big Data* 4.

Rapp, Michael, Eneldo Loza Mencía, and Johannes Fürnkranz (2018). 'Exploiting Anti-monotonicity of Multi-label Evaluation Measures for Inducing Multi-label Rules'. In: *Proc. Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pp. 29–42.

Rapp, Michael, Eneldo Loza Mencía, and Johannes Fürnkranz (2019). 'On the Trade-off Between Consistency and Coverage in Multi-label Rule Learning Heuristics'. In: *Proc. International Conference on Discovery Science*, pp. 96–111.

Rapp, Michael, Eneldo Loza Mencía, Johannes Fürnkranz, and Eyke Hüllermeier (2021). 'Gradient-based Label Binning in Multi-label Classification'. In: *Proc. European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pp. 462–477.

Rapp, Michael, Eneldo Loza Mencía, Johannes Fürnkranz, Vu-Linh Nguyen, and Eyke Hüllermeier (2020). 'Learning Gradient Boosted Multi-label Classification Rules'. In: *Proc. European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pp. 124–140.

Read, Jesse (2008). 'A Pruned Problem Transformation Method for Multi-label Classification'. In: *Proc. New Zealand Computer Science Research Student Conference (NZCSRS)*, pp. 143–150.

Read, Jesse (2010). 'Scalable Multi-label Classification'. PhD thesis. University of Waikato.

Read, Jesse, Bernhard Pfahringer, and Geoff Holmes (2008). 'Multi-label Classification Using Ensembles of Pruned Sets'. In: *Proc. IEEE International Conference on Data Mining*, pp. 995–1000.

Read, Jesse, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank (2011). 'Classifier chains for multi-label classification'. In: *Machine Learning* 85.3, p. 333.

Read, Jesse, Peter Reutemann, Bernhard Pfahringer, and Geoffrey Holmes (2016). 'MEKA: A Multi-label/Multi-target Extension to WEKA'. In: *Journal of Machine Learning Research* 17, pp. 1–5.

Reyes, Oscar, Carlos Morell, and Sebastián Ventura (2015). 'Scalable extensions of the ReliefF algorithm for weighting and selecting features on the multi-label learning context'. In: *Neurocomputing* 161, pp. 168–182.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2016). '"Why Should I Trust You?" Explaining the Predictions of Any Classifier'. In: *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144.

Rijsbergen, Cornelis van (1979). *Information Retrival*. Butterworths.

Ripley, Brian D. (2007). *Pattern Recognition and Neural Networks*. Cambridge University Press.

Rokach, Lior, Alon Schclar, and Ehud Itach (2014). 'Ensemble Methods for multi-label classification'. In: *Expert Systems with Applications* 41.16, pp. 7507–7523.

Schapire, Robert E. and Yoram Singer (2000). 'BoosTexter: A Boosting-based System for Text Categorization'. In: *Machine Learning* 39.2, pp. 135–168.

Sebastiani, Fabrizio (2002). 'Machine Learning in Automated Text Categorization'. In: *ACM Computing Surveys (CSUR)* 34.1, pp. 1–47.

Sechidis, Konstantinos, Grigorios Tsoumakas, and Ioannis Vlahavas (2011). 'On the stratification of multi-label data'. In: *Proc. European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, pp. 145–158.

Senge, Robin, Juan José del Coz, and Eyke Hüllermeier (2013). 'Rectifying Classifier Chains for Multi-Label Classification'. In: *Proc. Lernen, Wissen & Adaptivität*, pp. 151–158.

Shafer, John C., Rakesh Agrawal, and Manish Mehta (1996). 'SPRINT: A Scalable Parallel Classifier for Data Mining'. In: *Proc. International Conference on Very Large Data Bases*. Vol. 96, pp. 544–555.

Shao, Huan, GuoZheng Li, GuoPing Liu, and YiQin Wang (2013). 'Symptom selection for multi-label data of inquiry diagnosis in traditional Chinese medicine'. In: *Information Sciences* 56.5, pp. 1–13.

Shi, Haijian (2007). 'Best-first Decision Tree Learning'. PhD thesis. University of Waikato.

Si, Si, Huan Zhang, S. Sathiya Keerthi, Dhruv Mahajan, Inderjit S. Dhillon, and Cho-Jui Hsieh (2017). 'Gradient Boosted Decision Trees for High Dimensional Sparse Output'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 3182–3190.

Snoek, Cees GM, Marcel Worring, Jan C Van Gemert, Jan-Mark Geusebroek, and Arnold WM Smeulders (2006). 'The Challenge Problem for Automated Detection of 101 Semantic Concepts in Multimedia'. In: *Proc. ACM International Conference on Multimedia*, pp. 421–430.

Sorower, Mohammad S. (2010). *A Literature Survey on Algorithms for Multi-label Learning*. Tech. rep. Department of Computer Science, Oregon State University.

Spolaôr, Newton, Everton Alvares Cherman, Maria Carolina Monard, and Huei Diana Lee (2013). 'A Comparison of Multi-label Feature Selection Methods using the Problem Transformation Approach'. In: *Electronic Notes in Theoretical Computer Science* 292, pp. 135–151.

Srivastava, Ashok N. and Brett Zane-Ulman (2005). 'Discovering Recurring Anomalies in Text Reports Regarding Complex Space Systems'. In: *Proc. IEEE Aerospace Conference*, pp. 3853–3862.

Sugeno, Michio (1974). 'Theory of fuzzy integrals and its applications'. PhD thesis. Tokyo Institute of Technology.

Sun, Liang, Shuiwang Ji, and Jieping Ye (2010). 'Canonical correlation analysis for multilabel classification: A least-squares formulation, extensions, and analysis'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1, pp. 194–200.

Sun, Liang, Shuiwang Ji, and Jieping Ye (2019). *Multi-Label Dimensionality Reduction*. Chapman and Hall/CRC.

Szymański, Piotr and Tomasz Kajdanowicz (2017). 'A Network Perspective on Stratification of Multi-Label Data'. In: *Proc. International Workshop on Learning with Imbalanced Domains: Theory and Applications*, pp. 22–35.

Tai, Farbound and Hsuan-Tien Lin (2012). 'Multilabel classification with principal label space transformation'. In: *Neural Computation* 24.9, pp. 2508–2542.

Thabtah, Fadi A. and Peter I. Cowling (2007). 'A greedy classification algorithm based on association rule'. In: *Applied Soft Computing* 7.3, pp. 1102–1111.

Thabtah, Fadi A., Peter I. Cowling, and Yonghong Peng (2004). 'MMAC: A New Multi-class, Multi-label Associative Classification Approach'. In: *Proc. IEEE International Conference on Data Mining (ICDM)*, pp. 217–224.

Thabtah, Fadi A., Peter I. Cowling, and Yonghong Peng (2006). 'Multiple labels associative classification'. In: *Knowledge and Information Systems* 9.1, pp. 109–129.

Theron, Hendrik and Ian Cloete (1996). 'BEXA: A Covering Algorithm for Learning Propositional Concept Descriptions'. In: *Machine Learning* 24.1, pp. 5–40.

Thrun, Sebastian (2010). 'Toward Robotic Cars'. In: *Communications of the ACM* 53.4, pp. 99–106.

Trajdos, Pawel and Marek Kurzynski (2019). 'Dynamic classifier chains for multi-label learning'. In: *Proc. German Conference on Pattern Recognition*, pp. 567–580.

Trohidis, Konstantinos, Grigorios Tsoumakas, George Kalliris, and Ioannis Vlahavas (2008). 'Multi-label classification of music into emotions'. In: *Proc. International Society for Music Information Retrieval (ISMIR)*, pp. 325–330.

Tsamardinos, Ioannis, Elissavet Greasidou, and Giorgos Borboudakis (2018). 'Bootstrapping the out-of-sample predictions for efficient and accurate cross-validation'. In: *Machine Learning* 107.12, pp. 1895–1922.

Tsoumakas, Grigorios, Anastasios Dimou, Eleftherios Spyromitros, Vasileios Mezaris, Ioannis Kompatsiaris, and Ioannis Vlahavas (2009). 'Correlation-Based Pruning of Stacked Binary Relevance Models for Multi-Label Learning'. In: *Proc. International Workshop on Learning from Multi-Label Data*, pp. 101–116.

Tsoumakas, Grigorios, Ioannis Katakis, and Ioannis Vlahavas (2008). 'Effective and efficient multilabel classification in domains with large number of labels'. In: *Proc. ECML-PKDD 2008 Workshop on Mining Multidimensional Data*. Vol. 21, pp. 53–59.

Tsoumakas, Grigorios, Ioannis Katakis, and Ioannis Vlahavas (2009). 'Mining Multi-label Data'. In: *Data Mining and Knowledge Discovery Handbook*, pp. 667–685.

Tsoumakas, Grigorios, Ioannis Katakis, and Ioannis Vlahavas (2010). 'Random k-Labelsets for Multilabel Classification'. In: *IEEE Transactions on Knowledge and Data Engineering* 23.7, pp. 1079–1089.

Tsoumakas, Grigorios, Eleftherios Spyromitros-Xioufis, Jozef Vilcek, and Ioannis Vlahavas (2011). 'MULAN: A Java library for Multi-Label Learning'. In: *Journal of Machine Learning Research* 12, pp. 2411–2414.

Turnbull, Douglas, Luke Barrington, David Torres, and Gert Lanckriet (2008). 'Semantic Annotation and Retrieval of Music and Sound Effects'. In: *IEEE Transactions on Audio, Speech, and Language Processing* 16.2, pp. 467–476.

Ueda, Naonori and Kazumi Saito (2003). 'Parametric Mixture Models for Multi-Labeled Text'. In: *Proc. Advances in Neural Information Processing Systems*, pp. 737–744.

Waegeman, Willem, Krzysztof Dembczyński, Arkadiusz Jachnik, Weiwei Cheng, and Eyke Hüllermeier (2014). 'On the Bayes-Optimality of F-Measure Maximizers'. In: *Journal of Machine Learning Research* 15, pp. 3333–3388.

Wang, Hua, Chris Ding, and Heng Huang (2010). 'Multi-label Linear Discriminant Analysis'. In: *Proc. European Conference on Computer Vision (ECCV)*, pp. 126–139.

Webb, Geoffrey I. (1995). 'OPUS: An efficient admissible algorithm for unordered search'. In: *Journal of Artificial Intelligence Research* 3, pp. 431–465.

Weiss, Sholom M. and Nitin Indurkhya (2000). 'Lightweight Rule Induction'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 1135–1142.

Wohlrab, Lars and Johannes Fürnkranz (2011). 'A review and comparison of strategies for handling missing values in separate-and-conquer rule learning'. In: *Journal of Intelligent Information Systems* 36.1, pp. 73–98.

Wold, Svante, Kim Esbensen, and Paul Geladi (1987). 'Principal Component Analysis'. In: *Chemometrics and Intelligent Laboratory Systems* 2.1-3, pp. 37–52.

Wu, Qingyao, Mingkui Tan, Hengjie Song, Jian Chen, and Michael K. Ng (2016). 'ML-FOREST: A Multi-Label Tree Ensemble Method for Multi-Label Classification'. In: *IEEE Transactions on Knowledge and Data Engineering* 28.10, pp. 2665–2680.

Xie, Ming-Kun and Sheng-Jun Huang (2018). 'Partial Multi-Label Learning'. In: *Proc. AAAI Conference on Artificial Intelligence*. Vol. 32. 1.

Xu, Jianhua (2018). 'A weighted linear discriminant analysis framework for multi-label feature extraction'. In: *Neurocomputing* 275, pp. 107–120.

Xu, Jianhua, Jiali Liu, Jing Yin, and Chengyu Sun (2016). 'A multi-label feature extraction algorithm via maximizing feature variance and feature-label dependence simultaneously'. In: *Knowledge-Based Systems* 98, pp. 172–184.

Yager, Ronald R. and Dimitar P. Filev (1999). 'Induced ordered weighted averaging operators'. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 29.2, pp. 141–150.

Yager, Ronald R. and Janusz Kacprzyk (2012). *The Ordered Weighted Averaging Operators: Theory and Applications*. Springer Science & Business Media.

Yan, Rong, Jelena Tešić, and John R. Smith (2007). 'Model-Shared Subspace Boosting for Multi-label Classification'. In: *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 834–843.

Yang, Yiming and Jan O. Pedersen (1997). 'A Comparative Study on Feature Selection in Text Categorization'. In: *Proc. International Conference on Machine Learning (ICML)*. Vol. 97. 412-420, p. 35.

Yu, Kai, Shipeng Yu, and Volker Tresp (2005). 'Multi-Label Informed Latent Semantic Indexing'. In: *Proc. International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 258–265.

Zhang, Min-Ling, Yu-Kun Li, Xu-Ying Liu, and Xin Geng (2018). 'Binary relevance for multi-label learning: An overview'. In: *Frontiers of Computer Science* 12.2, pp. 191–202.

Zhang, Min-Ling, José M Peña, and Victor Robles (2009). 'Feature selection for multi-label naive Bayes classification'. In: *Information Sciences* 179.19, pp. 3218–3229.

Zhang, Min-Ling and Lei Wu (2014). 'LIFT: Multi-Label Learning with Label-Specific Features'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.1, pp. 107–120.

Zhang, Min-Ling and Zhi-Hua Zhou (2007). 'ML-KNN: A lazy learning approach to multi-label learning'. In: *Pattern Recognition* 40.7, pp. 2038–2048.

Zhang, Min-Ling and Zhi-Hua Zhou (2014). 'A review on multi-label learning algorithms'. In: *IEEE Transactions on Knowledge and Data Engineering* 26.8, pp. 1819–1837.

Zhang, Mingyuan, Harish Guruprasad Ramaswamy, and Shivani Agarwal (2020). 'Convex Calibrated Surrogates for the Multi-Label F-Measure'. In: *Proc. International Conference on Machine Learning (ICML)*, pp. 11246–11255.

Zhang, Xiatian, Quan Yuan, Shiwan Zhao, Wei Fan, Wentao Zheng, and Zhong Wang (2010). 'Multi-label Classification without the Multi-label Cost'. In: *Proc. SIAM International Conference on Data Mining*, pp. 778–789.

Zhang, Yi and Jeff Schneider (2011). ‘Multi-Label Output Codes using Canonical Correlation Analysis’. In: *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 873–882.

Zhang, Yin and Zhi-Hua Zhou (2010). ‘Multilabel Dimensionality Reduction via Dependence Maximization’. In: *ACM Transactions on Knowledge Discovery from Data)* 4.3, pp. 1–21.

Zhang, Zhendong and Cheolkon Jung (2020). ‘GBDT-MO: Gradient-Boosted Decision Trees for Multiple Outputs’. In: *IEEE Transactions on Neural Networks and Learning Systems*.

Zhou, Tianyi, Dacheng Tao, and Xindong Wu (2012). ‘Compressed labeling on distilled labelsets for multi-label learning’. In: *Machine Learning* 88.1-2, pp. 69–126.

Zhu, Pengfei, Qi Hu, Qinghua Hu, Changqing Zhang, and Zhizhao Feng (2018). ‘Multi-view label embedding’. In: *Pattern Recognition* 84, pp. 126–135.

Zilke, Jan Ruben, Eneldo Loza Mencía, and Frederik Janssen (2016). ‘DeepRED – Rule Extraction from Deep Neural Networks’. In: *Proc. International Conference on Discovery Science*, pp. 457–473.