Karlsruher Institut für Technologie

Leveraging Constraints for User-Centric Feature Selection

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Jakob Bach

Tag der mündlichen Prüfung: 20. Januar 2025
1. Referent/Referentin: T.T.-Prof. Dr. Peer Nowack
2. Referent/Referentin: Prof. Dr. Ira Assent

Acknowledgments

Obtaining a PhD is a long and challenging process, with many ups, downs, and intermediate results on the way there. Nevertheless, all that formally matters are the dissertation and defense at the very end. In this sense, I want to thank Prof. Peer Nowack and Prof. Ira Assent for serving as reviewers, Prof. Benjamin Schäfer for serving as the third examiner, and Prof. Klemens Böhm for the freedom and funding in the many years before.

To all PhD students reading this text: Do not be fooled by the large number of people who are often mentioned in a dissertation's acknowledgments. The PhD is, first and foremost, a personal project and essentially depends on your own motivation and work. It can be a very lonely struggle from time to time. If you do not advance your PhD project, nobody else will do. However, other researchers and non-researchers can (and should) have a positive impact on your PhD's success, generally and in my personal case.

In the context of this dissertation, I want to thank Holger Trittenbach and Vadim Arzamasov for inspiration and advice on the topics that constitute my PhD research. Further, my gratitude goes to Florian, Miki, Stas, and Tobias for proofreading. Additional thanks go to all attendees of my defense and to those who made me a nice hat.

In the context of my work at KIT, several other persons made my PhD years more pleasant. Elaheh, Markus, Pawel, and Tobias: Thanks for the successful collaborations and for showing me that collaborative research is much more fun than working alone, subject to having the right collaborators. Edouard: Thanks for our joint undertaking to enhance the data science course. Federico: Thanks for the sound teamwork regarding, but not limited to, the data science lab course. Miki: Thanks for bringing cakes, games, and humor to the chair. Pawel: Thanks for deep conversations on the PhD, research in general, and various other topics. The rest of the chair: Thanks for also being there (participation trophy). I particularly want to thank my fellow board-game and table-soccer players. For legal reasons, I hereby declare that all playing took place during breaks and after work.

Finally, I want to thank my family and friends for their moral support, though they will not read this dissertation anyway. (Honestly – why should they?) Special thanks go to my mum, whose systematic bread-slicing activities inspired me to investigate the problem of Number Partitioning, which kicked off my journey into complexity theory.

Abstract

Feature selection in machine learning aims to identify the variables in a dataset that are most useful for predictions. Feature-selection methods are ubiquitous for a variety of reasons: They can increase prediction quality, reduce hardware requirements, and ease understanding of the data. However, existing feature-selection methods do not satisfy user needs in certain scenarios: (1) Users may want to integrate domain knowledge into feature selection. For example, established laws or hypotheses from the domain can make selecting certain feature combinations unintuitive for users. In contrast, existing feature-selection methods typically ignore domain knowledge or only support particular types of it. (2) Multiple, differently composed feature sets may yield good predictions. Such alternatives may provide users with different explanations for predictions. In contrast, existing feature-selection methods typically yield just one feature set.

In this thesis, we make feature selection more user-centric by introducing constraints on the composition of feature sets. Integrating such constraints into existing feature-selection methods is challenging since constraints may limit admissible feature sets arbitrarily, particularly when combining different constraint types. In addition, constraints may negatively affect the feature sets' predictive quality. Our contribution is fourfold:

(1) We analyze the impact of constraints on feature-selection results. First, we formalize constrained feature selection as an optimization problem. Our basic problem definition is independent of the feature-selection method. To consider constraints, we propose employing a Satisfiability Modulo Theories (SMT) solver, which allows the use and combination of a wide range of constraint types. Second, we evaluate the impact of constraints empirically. We observe a trade-off between the strength of the constraints and the predictive quality of the selected features. However, the effect is non-linear, i.e., strong constraints can still result in high feature-set quality.

(2) We use constraints to express and compare scientific hypotheses in a materials-science use case. In particular, we collaborate with domain experts to formulate corresponding constraint types. Our experiments demonstrate that some constraint types lead to differently composed feature sets with high quality, which motivates our following work.

(3) We use constraints to find alternative feature sets, i.e., feature sets that differ from others while simultaneously optimizing feature-set quality. First, we formalize alternative feature selection with 0-1 integer linear constraints. Users can control the number and dissimilarity of alternatives with one parameter each. Second, we discuss how to integrate existing notions of feature-set quality as the objective of the optimization problem. Third, we analyze the time complexity of this problem and show $\mathcal{NP}$-hardness, for a simple

notion of feature-set quality already. Fourth, we introduce heuristic approximations for the latter. Fifth, we evaluate our approaches with five feature-selection methods. We observe that our approaches can find high-quality alternative feature sets, and the two parameters allow users to exercise control over the quality.

(4) We use constraints to find sparse and alternative feature sets for subgroups. Subgroup-discovery methods search interesting regions in a dataset that admit concise descriptions, e.g., a logical conjunction of bounds on feature values. First, we formalize subgroup discovery as an SMT optimization problem. Second, we formalize two user-centric constraint types: (a) We make subgroup descriptions sparse by limiting the number of features used. (b) We propose the problem of finding alternative subgroup descriptions, which use different features to describe a given subgroup. Third, we prove $\mathcal{NP}$-hardness of optimization with either constraint type. Fourth, we describe how to integrate both constraint types into existing heuristic subgroup-discovery methods. Fifth, we evaluate solver-based and heuristic subgroup discovery empirically. We observe that heuristic search methods are not only fast but also yield high-quality subgroups, even with the two constraint types.

Zusammenfassung

Die Merkmalsauswahl (*feature selection*) im maschinellen Lernen zielt darauf ab, die nützlichsten Variablen in einem Datensatz für Vorhersagen zu finden. Methoden zur Merkmalsauswahl sind aus vielfältigen Gründen allgegenwärtig: Sie können die Vorhersagequalität verbessern, Hardwareanforderungen verringern und das Verständnis der Daten erleichtern. Allerdings reichen bestehende Methoden zur Merkmalsauswahl in bestimmten Szenarien nicht aus, um Nutzerbedürfnisse zu befriedigen: (1) Manche Nutzer möchten Domänenwissen in die Merkmalsauswahl miteinbeziehen. Beispielsweise können etablierte Gesetzmäßigkeiten oder Hypothesen aus der Domäne die Auswahl bestimmter Merkmalskombinationen unintuitiv für Nutzer machen. Bestehende Methoden zur Merkmalsauswahl ignorieren jedoch typischerweise Domänenwissen oder unterstützen nur bestimmte Arten davon. (2) Mehrere, unterschiedlich zusammengesetzte Merkmalsmengen können gute Vorhersagen liefern. Solche Alternativen können Nutzern verschiedene Erklärungen für Vorhersagen bieten. Bestehende Methoden zur Merkmalsauswahl geben jedoch typischerweise nur eine Merkmalsmenge zurück.

In dieser Dissertation machen wir die Merkmalsauswahl Nutzer-zentrierter, indem wir Nebenbedingungen (*constraints*) für die Zusammensetzung der Merkmalsmengen einführen. Die Integration solcher Nebenbedingungen in bestehende Methoden zur Merkmalsauswahl ist herausfordernd, da Nebenbedingungen die zulässigen Merkmalsmengen beliebig einschränken können, insbesondere beim Kombinieren verschiedener Arten von Nebenbedingungen. Weiterhin können Nebenbedingungen die Vorhersagequalität der Merkmalsmengen negativ beeinflussen. Unsere Arbeit leistet vier wesentliche Beiträge:

(1) Wir untersuchen den Einfluss von Nebenbedingungen auf die Ergebnisse der Merkmalsauswahl. Als Erstes formalisieren wir Merkmalsauswahl unter Nebenbedingungen als ein Optimierungsproblem. Unsere grundlegende Problemdefinition ist unabhängig von der Methode zur Merkmalsauswahl. Um Nebenbedingungen zu berücksichtigen, setzen wir einen Löser für Satisfiability-Modulo-Theories-(SMT)-Probleme ein. Dieser erlaubt die Nutzung und Kombination einer großen Bandbreite von Nebenbedingungen. Als Zweites werten wir den Einfluss von Nebenbedingungen empirisch aus. Wir beobachten einen Zielkonflikt zwischen der Stärke der Nebenbedingungen und der Vorhersagequalität der ausgewählten Merkmale. Dieser Effekt ist jedoch nichtlinear; insbesondere können starke Nebenbedingungen immer noch zu einer hohen Qualität der Merkmalsmenge führen.

(2) Wir nutzen Nebenbedingungen zum Ausdrücken und Vergleichen wissenschaftlicher Hypothesen in einem materialwissenschaftlichen Anwendungsfall. Insbesondere kooperieren wir mit Domänenexperten, um die entsprechenden Nebenbedingungen zu formulieren.

Unsere Experimente zeigen, dass manche Arten der Nebenbedingungen zu unterschiedlich zusammengesetzten Merkmalsmengen mit gleichzeitig hoher Qualität führen. Dieses Ergebnis motiviert unsere folgende Arbeit.

(3) Wir nutzen Nebenbedingungen zum Finden alternativer Merkmalsmengen. Letztere sind Merkmalsmengen, die sich von anderen Merkmalsmengen unterscheiden und gleichzeitig eine möglichst hohe Qualität erreichen sollen. Als Erstes formalisieren wir alternative Merkmalsauswahl mittels ganzzahliger linearer Nebenbedingungen und binären Entscheidungsvariablen. Nutzer können die Anzahl und die Unterschiedlichkeit der Alternativen durch jeweils einen Parameter bestimmen. Als Zweites diskutieren wir, wie sich bestehende Konzepte für die Qualität von Merkmalsmengen als Zielfunktion des Optimierungsproblems integrieren lassen. Als Drittes untersuchen wir die Zeitkomplexität des Optimierungsproblems und zeigen, dass es bereits mit einer einfachen Zielfunktion $\mathcal{NP}$-schwer ist. Als Viertes stellen wir Näherungsheuristiken für diese Zielfunktion vor. Als Fünftes werten wir unsere Ansätze mit fünf Methoden zur Merkmalsauswahl aus. Wir beobachten, dass unsere Ansätze alternative Merkmalsmengen mit hoher Qualität finden können und dass die beiden Parameter den Nutzern Kontrolle über die Qualität geben.

(4) Wir nutzen Nebenbedingungen, um kleine und alternative Merkmalsmengen für Untergruppen (*subgroups*) zu finden. Methoden zur Untergruppenerkennung (*subgroup discovery*) suchen in einem Datensatz nach interessanten Bereichen, die sich prägnant beschreiben lassen, beispielsweise mittels logischer Konjunktionen über Intervalle für Merkmalswerte. Als Erstes formalisieren wir Untergruppenerkennung als ein SMT-Optimierungsproblem. Als Zweites formalisieren wir zwei Nutzer-zentrierte Arten von Nebenbedingungen: (a) Wir verkleinern die Beschreibungen von Untergruppen, indem wir die Zahl der genutzten Merkmale begrenzen. (b) Wir definieren das Problem des Findens alternativer Untergruppenbeschreibungen. Letztere beschreiben eine gegebene Untergruppe mit anderen Merkmalen. Als Drittes beweisen wir, dass die Optimierung von Untergruppen mit beiden Arten von Nebenbedingungen $\mathcal{NP}$-schwer ist. Als Viertes beschreiben wir, wie sich die beiden Arten von Nebenbedingungen in bestehende Heuristiken zur Untergruppenerkennung integrieren lassen. Als Fünftes werten wir SMT-Löser-basierte und heuristische Methoden zur Untergruppenerkennung empirisch aus. Wir beobachten, dass heuristische Suchmethoden nicht nur schnell sind, sondern auch Untergruppen mit hoher Qualität liefern, selbst wenn beide Arten von Nebenbedingungen berücksichtigt werden.

Contents

1. Introduction

1.1. Background

With the rising importance of machine learning in general, the issue of interpretability has gained prominence in recent years [36, 69, 153]. In particular, interpretability considers the user perspective, going beyond prediction performance as the only quality criterion for machine-learning approaches. There are various ways to foster interpretability in machine-learning pipelines. Two important fields in this direction are feature selection and subgroup discovery. We will briefly and informally introduce these two fields next; see Chapter 2 for detailed fundamentals.

Feature selection Feature selection aims to identify the variables in a dataset that are most useful for predictions [84]. Some feature-selection methods act as a pre-processing step before training a prediction model, while others operate during or in combination with model training. Several reasons speak for feature selection [37, 132]: By reducing dataset dimensionality, it lowers the computational cost and memory requirements of training, storing, and using prediction models. Next, prediction models may generalize better without irrelevant and spurious predictors. While some model types can implicitly select relevant features, others cannot. Finally, prediction models with fewer inputs may become simpler, thereby improving interpretability.

Subgroup discovery The goal of subgroup discovery is to find 'interesting' subgroups, i.e., subsets of a dataset, like data objects where the prediction target takes a particular value [8]. Additionally, such subgroups should be described with a combination of simple conditions on feature values, thereby fostering interpretability. E.g., Figure 1.1 displays a rectangle-shaped subgroup description for a two-dimensional, real-valued dataset with a binary prediction target. This subgroup is defined by $(\mathit{Feature\_1} \in [3.0, 5.1]) \wedge (\mathit{Feature\_2} \in [1.0, 1.8])$ and contains a considerably higher fraction of data objects with $\mathit{Target} = 1$ than the complete dataset. One may see subgroup discovery as an extension of feature selection. In particular, the result of subgroup discovery is not only a feature set but also comprises the value conditions. Instead of training a prediction model with the selected feature set, the subgroup description implicitly forms a prediction model already. In general, the subgroup description may define conditions on all features. However, if it becomes too complex, i.e., involves too many features, its interpretability may decrease [147]. Thus, it makes sense to only use a subset of features in the subgroup description.

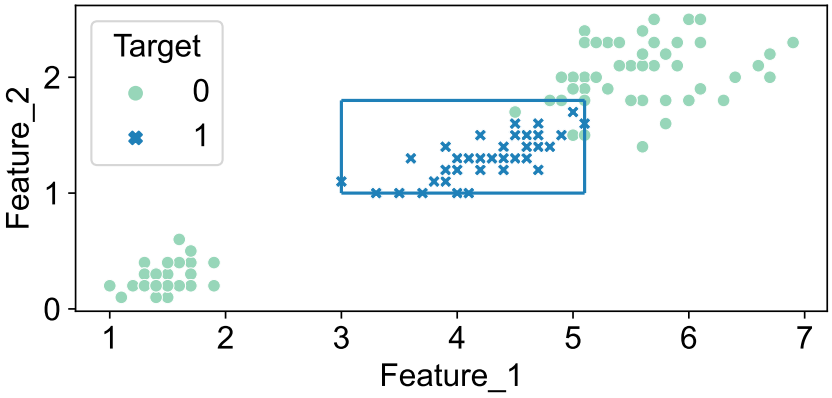


Figure 1.1.: Exemplary subgroup description in the form of a rectangle for a dataset with two real-valued features and a binary prediction target.

1.2. Research Gaps

There are many existing methods for feature selection and subgroup discovery [8, 132]. However, we see two main research gaps where these two fields can be made more user-centric: integrating domain knowledge and finding alternative solutions. We will elaborate on these two points next. Additionally, we will explain how we address both points with constraints on feature sets.

1.2.1. Integrating Domain Knowledge

Motivation Most methods for feature selection and subgroup discovery optimize a quantitative quality criterion, e.g., a metric for prediction performance. However, such approaches may yield suboptimal results from the user perspective, e.g., for interpreting the feature sets qualitatively. In particular, these methods disregard domain knowledge about the dataset and its features. Such domain knowledge is available in many scientific settings [104, 218]. Users may want to consider different types of knowledge:

1. *Firm domain knowledge:* To make prediction models consistent with the domain, users may want them to adhere to known facts from the domain. For example, users may *know* that some features are redundant from the domain perspective and thus want to rule out selecting combinations of these features.

2. *Hypotheses:* Hypotheses are ideas or expectations that users would like the selected feature sets to respect. For example, users may *hypothesize* that certain features are redundant, and they want to study whether feature sets without these hypothesized redundancies suffice for predictions. If prediction performance drops significantly for feature sets respecting this hypothesis, the hypothesis may be wrong.

3. *Preferences:* Besides domain knowledge in the narrow sense, users may have further preferences on the selected feature sets. One example, which is already common in

feature selection and subgroup discovery, is limiting the size of the feature sets. In particular, smaller feature sets may be easier to interpret.

Problem and challenges In all previous scenarios, users want to limit the space of valid feature sets but not select individual feature sets manually. Thus, we propose constraints on feature sets as a suitable approach to consider domain knowledge automatically. We call the corresponding paradigms *constrained feature selection* and *constrained subgroup discovery*. While observing constraints, we still want to optimize the quality of solutions, i.e., feature sets or subgroups. The constraints may lower solution quality, but their exact impact is unclear a priori, and a certain decrease may still be acceptable for users. Another central question is how to consider these constraints. An ideal approach should support a wide range of constraints and their combination. This flexibility gives users a lot of freedom, but it makes integration into existing methods for feature selection or subgroup discovery harder. Also, the approach for integrating constraints should not be tied to one particular method. I.e., we see the paradigm of constraints as orthogonal to existing methods. However, the latter differ considerably in their objectives and internal structure, which poses a challenge for integrating constraints in a general fashion.

Related work Constraints have been used in many areas of data mining and machine learning [71], e.g., automated machine learning [162], clustering [48, 49], explainable AI [52, 70, 156, 202], and pattern mining [163, 204]. In feature selection, existing works tend to aim at just one feature-selection method and one constraint type, like group constraints [61, 97, 206, 229, 235], cost constraints [98, 154, 174, 179, 234], or cardinality constraints [108, 125, 197, 226]. Thus, there is potential for a more general approach to constrained feature selection. In subgroup discovery, considering constraints is more common [8, 147]. However, the subgroup-discovery methods that support constraints are typically limited to individual constraint types or at least require constraints to have particular properties, since the constraints need to be integrated into algorithmic search procedures.

1.2.2. Finding Alternative Solutions

Motivation Most methods for feature selection and subgroup discovery return only one solution, i.e., the optimum according to a quality criterion. Nevertheless, there may be different solutions, i.e., using other features, that achieve similar quality. Such alternative solutions are interesting for users, e.g., to obtain several diverse explanations. In particular, alternatives can provide additional insights into predictions, enable users to develop and test different hypotheses, appeal to different kinds of users, and foster trust in the predictions [110, 220]. For example, in a dataset describing physical experiments, feature selection may help discover relationships between physical quantities. If multiple alternative sets of similar quality exist, further analyses and experiments may be necessary to reveal the true underlying physical mechanism. Only knowing one high-quality feature set and using it as the only explanation may be misleading in such a situation.

Problem and challenges We call our paradigms for finding alternative solutions *alternative feature selection* and *alternative subgroup description discovery*. In the former case, we want to optimize feature-set quality; in the latter case, similarity regarding the set of data objects in the subgroup. In both cases, constraints should enforce that the solutions are alternative in the sense of using different features. For example, one can search for multiple solutions sequentially and require each new solution to optimize quality while being sufficiently dissimilar to previous solutions. The constraints should be domain-independent since obtaining alternatives is a general concern and we want to free users from manually formulating corresponding constraint types. However, users should have control over alternatives, e.g., their number and dissimilarity. Depending on these user parameters, quality may drop, as with constraints in general, and users should decide on an acceptable quality trade-off. Another recurring concern is to keep the approach for alternatives orthogonal to existing methods for feature selection and subgroup discovery.

Related work Only a few feature-selection methods return multiple, diverse feature sets [31]. Existing approaches often do not guarantee diversity or do not give users control over diversity. In fields related to feature selection, the goal of obtaining multiple, diverse solutions has been studied, e.g., for subspace clustering [92, 157], subspace search [212], or explainable-AI techniques [5, 109, 156, 188]. These approaches are not directly applicable or easily adaptable to feature selection, and most of them provide limited or no user control. In subgroup discovery, various methods yield a diverse set of subgroups [25, 32, 126, 130, 139, 181]. However, this notion of alternatives typically aims to cover different subsets of data objects from the dataset. In contrast, our notion of alternative subgroup descriptions aims to cover a similar set of data objects as in a given subgroup but with different features in the description. Finally, for feature selection as well as subgroup discovery, existing work on alternatives typically proposes particular new search algorithms rather than an approach that is orthogonal to existing methods.

1.3. Contributions

This dissertation provides four core contributions to make feature selection more user-centric with the help of constraints. We address both research gaps from the previous section, i.e., integrating domain knowledge (cf. Section 1.2.1) and finding alternative solutions (cf. Section 1.2.2). Each core contribution corresponds to one chapter in the main part of this dissertation and may comprise several sub-contributions:

Evaluating the impact of constraints on feature-selection results (cf. Chapter 4) First, we formalize constrained feature selection as an optimization problem. Our problem definition is independent of the feature-selection method. We also discuss how to solve this problem. Second, we systematically analyze the impact of constraints. To this end, we use datasets from various domains and generate random constraints. A Satisfiability Modulo Theories (SMT) optimizer finds the optimal feature sets under these constraints. Such a solver-based

approach supports a wide range of constraint types. We analyze the relationships between various evaluation metrics, describing the constraints and the feature-selection results. Our experiments show that the impact of constraints strongly depends on the constraint type. Additionally, we observe a nonlinear effect between the strength of constraints and feature-set quality: Even if constraints prune a significant fraction of feature sets, the quality may not decrease to the same extent.

Formulating scientific hypotheses as constraints (cf. Chapter 5) We conduct a case study in materials science to evaluate the impact of constraints for a concrete use case. Here, we collaborate with domain experts to formulate constraints. In particular, the constraints represent preferences and hypotheses from the domain. We evaluate the hypotheses by analyzing how the corresponding constraints affect feature selection. Our experiments show little variance in feature-set quality over the constraints, indicating that the data does not contradict the hypotheses. However, the resulting feature sets differ in their composition, demonstrating that alternative solutions of similar quality exist.

Finding alternative feature sets (cf. Chapter 6) First, we formalize alternative feature selection as an optimization problem. In particular, we define alternatives via 0-1 integer linear constraints on feature sets. Our approach is orthogonal to the feature-selection method and equips users with two parameters, i.e., the number of alternatives and a dissimilarity threshold. For multiple alternatives, we consider sequential as well as simultaneous search. Second, we discuss how to solve this optimization problem. To that end, we describe how to integrate different categories of existing feature-selection methods into a solver-based search for alternatives. Third, we analyze the time complexity of the optimization problem. We show $\mathcal{NP}$-hardness, even for a simple notion of feature-set quality. Fourth, we propose heuristic search methods that achieve a constant-factor approximation for a particular notion of feature-set quality. Fifth, we conduct comprehensive experiments with five feature-selection methods, five search methods for alternatives, and varying the two user parameters. Our experiments demonstrate that users can influence the quality of alternatives: This quality tends to decrease for more alternatives and a higher dissimilarity threshold. Runtime-wise, a solver-based sequential search for multiple alternatives is significantly faster than a simultaneous one while yielding a similar quality. Additionally, our heuristic search methods yield a high quality in negligible runtime.

Discovering spare and alternative subgroup descriptions (cf. Chapter 7) First, we formalize subgroup discovery as a Satisfiability Modulo Theories (SMT) optimization problem. This formulation admits a solver-based search for subgroups and allows integrating constraints. Second, we formalize two constraint types for this optimization problem. *Feature-cardinality constraints* limit the number of features used in subgroup descriptions. *Alternative subgroup descriptions* should use different features as a given subgroup description but cover a similar set of data objects. Users control the number of alternatives and a dissimilarity threshold. Third, we describe how to integrate these two constraint types into three existing heuristic search methods and two novel baselines for subgroup discovery.

Fourth, we analyze the time complexity of the subgroup-discovery problem with each of these two constraint types and prove several $\mathcal{NP}$-completeness results. Fifth, we conduct comprehensive experiments with different experimental scenarios: no constraints, feature-cardinality constraints, alternative subgroup descriptions, and different solver timeouts. We observe that subgroups using only a few features already show relatively high quality compared to unconstrained subgroups. With and without constraints, heuristic search methods yield similar subgroup quality as optimal solutions found by solver-based search, while being considerably faster.

1.4. Materials

We publish all code and experimental data for this dissertation under permissive licenses.

We provide the code, implemented in Python, via multiple *GitHub* repositories, which we additionally backed up in the *Software Heritage archive*:

- Chapters 4 and 5: `https://github.com/Jakob-Bach/Constrained-Filter-Feature-Selection`
  Archived at swh:1:dir:d94796f72c48b6f539cf528d0a8969b4ff61d42f
- Chapter 6: `https://github.com/Jakob-Bach/Alternative-Feature-Selection`
  Archived at swh:1:dir:1e2b6605040a2151d31a90cde947787e92c98519
- Chapter 7: `https://github.com/Jakob-Bach/Constrained-Subgroup-Discovery`
  Archived at swh:1:dir:5527bcc4a86a1583e8bedf241f4738fc9b0e535c

We released all repositories under the MIT license. Each repository contains detailed instructions on how to reproduce the corresponding experiments. Also, requirements files specify the versions of all dependencies. Additionally, we organized all generally applicable functionality of each repository, e.g., methods for alternative feature selection, as a Python package to ease reuse. The corresponding packages *alfese*, *cffs*, and *csd* are publicly available on *PyPI*.

We provide the experimental data via *RADAR4KIT* and put it under the CC-BY 4.0 license: `https://doi.org/10.35097/4kjyeg0z2bxmr6eh`

1.5. Prior Works

This dissertation bases on the following preprints and publications:

- Jakob Bach et al. “An Empirical Evaluation of Constrained Feature Selection”. In: *SN Comput. Sci.* 3.6 (2022). DOI: `10.1007/s42979-022-01338-z`
- Jakob Bach. *Leveraging Constraints for User-Centric Selection of Predictive Features*. AI Hub @ Karlsruhe. 2022. DOI: `10.5445/IR/1000151285`

- Jakob Bach. *Finding Optimal Diverse Feature Sets with Alternative Feature Selection*. arXiv:2307.11607v2 [cs.LG]. 2024. DOI: 10.48550/arXiv.2307.11607
- Jakob Bach and Klemens Böhm. "Alternative feature selection with user control". In: *Int. J. Data Sci. Anal.* (2024). DOI: 10.1007/s41060-024-00527-8
- Jakob Bach. *Using Constraints to Discover Sparse and Alternative Subgroup Descriptions*. arXiv:2406.01411v1 [cs.LG]. 2024. DOI: 10.48550/arXiv.2406.01411

We have reused these prior works' content but revised it to form a coherent monograph. In particular, these prior works contain all major contributions of this dissertation. However, we have extended, rephrased, restructured, or shortened the text in various places. Algorithms, definitions, equations, figures, notation, propositions, and tables in this dissertation may slightly differ from prior works, e.g., due to harmonization. The four chapters in the main part of this dissertation explicitly state which prior works they base on. All other chapters may reuse content from each of these prior works.

1.6. Dissertation Outline

The remainder of this dissertation is structured into three parts.

First, we present preliminaries. Chapter 2 describes relevant fundamentals of feature selection and subgroup discovery. Chapter 3 discusses related work for all our contributions.

The second part of this dissertation is the main part. It consists of four chapters featuring our contributions. Chapter 4 evaluates the impact of constraints on feature-selection results. Chapter 5 presents a case study where constraints express scientific hypotheses. Chapter 6 uses constraints to find alternative feature sets. Chapter 7 employs constraints to discover sparse and alternative subgroup descriptions.

The third part of this dissertation wraps up our work. Chapter 8 summarizes the conclusions, and Chapter 9 discusses future work. After the bibliography, Appendix A contains proofs and technical details not included in the main part of this dissertation.

2. Fundamentals

In this section, we introduce relevant fundamentals of feature selection (cf. Section 2.1) and subgroup discovery (cf. Section 2.2).

Notation We focus on tabular, real-valued data. In particular, $X \in \mathbb{R}^{m \times n}$ stands for a dataset in the form of a matrix. Each row is a data object, and each column is a feature. $\tilde{F} = \{f_1, \ldots, f_n\}$ is the corresponding set of feature names. We assume that categorical features have been made numeric, e.g., via a one-hot or an ordinal encoding [144]. $X_{i\cdot} \in \mathbb{R}^n$ denotes the values of all features for the i-th data object, while $X_{\cdot j} \in \mathbb{R}^m$ denotes the values of the j-th feature for all data objects. We consider supervised-learning scenarios. In particular, the vector $y \in Y^m$ represents the prediction target with domain Y, e.g., $Y = \{0, 1\}$ for binary classification or $Y = \mathbb{R}$ for regression. The function $Q(\cdot)$ returns the quality of a feature set or subgroup; we will define its corresponding arguments later.

2.1. Feature Selection

In this section, we introduce relevant fundamentals of feature selection: the optimization problem (cf. Section 2.1.1) and feature-selection methods (cf. Section 2.1.2).

2.1.1. Problem

Feature selection makes a binary decision $s_j \in \{0, 1\}$ for each feature, i.e., either selects it or not. The vector $s \in \{0, 1\}^n$ combines all these selection decisions and yields the selected feature set $F_s = \{f_j \mid s_j = 1\} \subseteq \tilde{F}$. To simplify notation, we drop the subscript s in definitions where we do not explicitly refer to the value of s but only the set F. The function $Q(s, X, y)$ returns the quality of such a feature set and depends on the feature-selection method. Without loss of generality, we assume that this function should be maximized:

Definition 1 (Feature selection)**.** *Given a dataset $X \in \mathbb{R}^{m \times n}$ with prediction target $y \in Y^m$,* feature selection *is the problem of making feature-selection decisions $s \in \{0, 1\}^n$ that maximize a given notion of feature-set quality $Q(s, X, y)$.*

Additionally, the user often defines a parameter $k \in \mathbb{N}$ denoting the exact or maximal number of selected features, i.e., the *cardinality* of the feature set.

2.1.2. Methods

There are different ways to define feature-set quality $Q(s, X, y)$. We only give a short overview and provide details for the methods we use in this dissertation. See [2, 37, 53, 132, 169] for comprehensive studies and surveys of feature selection. Feature-selection methods are typically categorized into filter, wrapper, and embedded methods [84].

2.1.2.1. Filter Methods

Filter feature selection evaluates the quality $Q(s, X, y)$ without training a prediction model. Univariate filters assess each feature's relevance independently. Multivariate filters also consider relationships between features. They often combine a measure of univariate feature relevance with a measure of bivariate feature redundancy. Some filter methods also consider feature intercooperation, i.e., the joint relevance of two or more features [208].

Univariate filter methods Univariate filter methods break $Q(s, X, y)$ down to the qualities $q_j = q(X_{\cdot j}, y)$ of individual features. $q(\cdot)$ is a bivariate dependency measure to quantify the relationship between each feature and the prediction target, e.g., mutual information [115] or the absolute value of Pearson correlation. The overall optimization objective is an aggregate of the selected features' qualities, typically their sum:

$$\max_s \quad Q_{\text{uni}}(s, X, y) = \sum_{j=1}^{n} q(X_{\cdot j}, y) \cdot s_j = \sum_{j=1}^{n} q_j \cdot s_j \tag{2.1}$$

This objective is cheap to compute but ignores interactions between features. The optimal feature set of size $k \in \mathbb{N}$ consists of the features with the k highest individual qualities.

FCBF The Fast Correlation-Based Filter (FCBF) [228] is a multivariate filter method that bases on the notion of predominance: Each selected feature's correlation with the prediction target must exceed (1) a user-defined threshold and (2) the correlation of each other selected feature with the given one. The first criterion ensures feature relevance, and the second criterion reduces feature redundancy. [228] proposes a search algorithm to find a set of such predominant features.

mRMR Minimal Redundancy Maximal Relevance (mRMR) [177] is a multivariate filter method that combines feature relevance and feature redundancy in one objective. Relevance corresponds to the dependency between features and the prediction target, which should be maximized, as for univariate filters. Redundancy, in turn, corresponds to the dependency between features, which should be minimized. Both terms are averaged over the selected features. Using a bivariate dependency measure $q(\cdot)$, the objective is maximizing the following difference between relevance and redundancy:

$$\max_s \quad Q_{\text{mRMR}}(s, X, y) = \frac{\sum_{j=1}^{n} q(X_{\cdot j}, y) \cdot s_j}{\sum_{j=1}^{n} s_j} - \frac{\sum_{j_1=1}^{n} \sum_{j_2=1}^{n} q(X_{\cdot j_1}, X_{\cdot j_2}) \cdot s_{j_1} \cdot s_{j_2}}{(\sum_{j=1}^{n} s_j)^2} \tag{2.2}$$

2.1.2.2. Wrapper Methods

Wrapper methods [112] evaluate feature-set quality $Q(s, X, y)$ by training prediction models with feature sets and measuring prediction quality. They employ a generic search strategy to iterate over candidate feature sets, e.g., genetic algorithms [2]. Feature-set quality is a black-box function in this search, i.e., the optimization algorithm only observes the objective's outputs rather than its internal structure. On the one hand, the repeated model training makes wrapper methods costly. On the other hand, the notion of feature-set quality stems from an actual prediction task rather than a simplified proxy objective.

2.1.2.3. Embedded Methods

Embedded methods train prediction models with built-in feature selection, e.g., decision trees [34] or random forests [33]. Thus, the criterion for feature-set quality is model-specific. For example, tree-based models often use information gain or the Gini index to select features for creating the tree's decision nodes during training.

2.1.2.4. Post-Hoc Feature-Importance Methods

Apart from conventional feature selection, there are various methods that assess feature importance after training a model. These methods range from local explanation methods like LIME [184] or SHAP [140], focusing on individual data objects, to global importance methods like permutation importance [33] or SAGE [45]. In particular, assessing feature importance plays a crucial role in the field of machine-learning interpretability [36, 153].

2.2. Subgroup Discovery

In this section, we introduce relevant fundamentals of subgroup discovery: the optimization problem (cf. Section 2.2.1) and common heuristic search methods (cf. Section 2.2.2).

2.2.1. Problem

In general, subgroup discovery involves finding descriptions of interesting subsets of a dataset [8]. There are multiple options for the type of dataset, kind of subgroup description, and criterion of interestingness. In the following, we formalize the notion of subgroup discovery that we tackle in this dissertation. For broader surveys, see [8, 88, 90, 215].

Subgroup (description) A subgroup description typically comprises a conjunction of conditions on individual features [147]. For real-valued data, the conditions constitute intervals. Thus, the subgroup description defines a hyperrectangle (cf. Figure 1.1) with a lower and upper bound for each feature. The bounds for a feature may also be infinite to leave it unrestricted. A data object resides in the subgroup if all its feature values are in the intervals formed by lower and upper bounds:

Definition 2 (Subgroup (description))**.** *Given a dataset* $X \in \mathbb{R}^{m \times n}$*, a* subgroup *is described by its lower bounds* $lb \in \{\mathbb{R} \cup \{-\infty, +\infty\}\}^n$ *and upper bounds* $ub \in \{\mathbb{R} \cup \{-\infty, +\infty\}\}^n$*. Data object* X_i*. is a* member *of this subgroup if* $\forall j \in \{1, \ldots, n\} : \left(X_{ij} \geq lb_j\right) \wedge \left(X_{ij} \leq ub_j\right)$*.*

While we focus on real-valued data, some subgroup-discovery methods only support categorical data and require continuous features to be discretized [90, 147]. In this case, the inequality comparisons become equality comparisons against feature values [8].

Throughout this dissertation, we often use the terms *subgroup* and *subgroup description* interchangeably. In a more strict sense, the former term denotes the subgroup's members, and the latter denotes the subgroup's bounds [8].

Subgroup discovery Subgroup discovery sets the bounds lb and ub to optimize a notion of subgroup quality $Q(lb, ub, X, y)$, i.e., the interestingness of the subgroup. To harmonize formalization and evaluation, we focus on binary-classification targets $y \in \{0, 1\}^m$. In general, one may also conduct subgroup discovery in multi-class, multi-target, or regression scenarios [8]. Without loss of generality, we assume a maximization problem:

Definition 3 (Subgroup discovery)**.** *Given a dataset* $X \in \mathbb{R}^{m \times n}$ *with prediction target* $y \in \{0, 1\}^m$*,* subgroup discovery *is the problem of finding a subgroup (cf. Definition 2) with bounds* $lb, ub \in \{\mathbb{R} \cup \{-\infty, +\infty\}\}^n$ *that maximizes a given notion of subgroup quality* $Q(lb, ub, X, y)$*.*

While this definition refers to one subgroup, some methods return a subgroup set [8].

Subgroup quality For binary targets, interesting subgroups should typically contain many data objects from one class but few from the other. Without loss of generality, we assume that the class with label '1' is the class of interest, also called *positive* class. Weighted Relative Accuracy (WRAcc) [120] is a popular metric for subgroup quality [147]:

$$\text{WRAcc} = \frac{m_b}{m} \cdot \left(\frac{m_b^+}{m_b} - \frac{m^+}{m} \right) \tag{2.3}$$

Besides the total number of data objects m, this metric considers the number of positive data objects m^+, the number of data objects in the subgroup m_b, and the number of positive data objects in the subgroup m_b^+. In particular, WRAcc is the product of two factors: m_b/m expresses the generality of the subgroup as the relative frequency of subgroup membership. The second factor measures the relative accuracy of the subgroup, i.e., the difference in the

relative frequency of the positive class between the subgroup and the whole dataset. If the subgroup contains the same fraction of positive data objects as the whole dataset, WRAcc is zero. The theoretical maximum and minimum of WRAcc depend on the class frequencies in the dataset. In particular, the maximum WRAcc for a dataset equals the product of the relative frequencies of positive and negative data objects in the dataset [143]:

$$\mathrm{WRAcc}_{\max} = \frac{m^+}{m} \cdot \left(1 - \frac{m^+}{m}\right) \tag{2.4}$$

This maximum is reached if all positive data objects are in the subgroup and all negative ones are outside, i.e., $m_b^+ = m_b = m^+$. The corresponding value is 0.25 if both classes occur with equal frequency but becomes smaller the more imbalanced the classes are. Thus, it makes sense to normalize WRAcc when working with datasets with different class frequencies. One option is a max-normalization to the range $[-1, 1]$ [143]:

$$\mathrm{nWRAcc} = \frac{\mathrm{WRAcc}}{\mathrm{WRAcc}_{\max}} = \frac{m_b^+ \cdot m - m^+ \cdot m_b}{m^+ \cdot (m - m^+)} \tag{2.5}$$

Alternatively, one can also min-max-normalize the range to $[0, 1]$ [35, 215].

2.2.2. Methods

There are heuristic search methods to discover subgroups, like PRIM [62] and Best Interval [141], as well as exhaustive search methods, like SD-Map [9, 11], MergeSD [73], and BSD [128, 130]. In this section, we discuss three heuristic search methods that are relevant for Chapter 7; see [8, 88, 90, 215] for comprehensive surveys.

PRIM *Patient Rule Induction Method (PRIM)* [62] is an iterative search algorithm. Its basic form consists of a peeling phase and a pasting phase. Peeling restricts the bounds of the subgroup iteratively, while pasting expands them. Algorithm 1 outlines the peeling phase for finding one subgroup, which is the flavor of PRIM we consider in this dissertation and denote as *PRIM*. Pasting may have little effect on the subgroup quality and is often left out [6]. Other extensions of PRIM are bumping [62, 117], which uses bagging of multiple PRIM runs to improve subgroup quality, and covering [62], which returns a sequence of subgroups covering different data objects.

The algorithm *PRIM* starts with a subgroup containing all data objects, which is the initial solution candidate (Lines 1–4). It continues peeling until the current solution candidate contains at most a fraction β_0 of data objects (Line 5). The support threshold $\beta_0 \in [0, 1]$ is a user parameter. The returned subgroup is the optimal solution candidate over all peeling iterations (Line 20). In our *PRIM* implementation, we additionally set *non-excluding bounds* to infinity (Lines 21–23). These are bounds that do not exclude any data objects from the subgroup, i.e., equal the minimum/maximum feature value over all data objects. Thereby, we ensure that these bounds remain non-excluding for any new data and make it easier to see which features are selected in the subgroup description (cf. Definition 14).

Algorithm 1: *PRIM* for subgroup discovery.

Input: Dataset $X \in \mathbb{R}^{m \times n}$,
Prediction target $y \in \{0,1\}^m$,
Subgroup-quality function $Q(lb, ub, X, y)$,
Peeling fraction $\alpha \in (0,1)$,
Support threshold $\beta_0 \in [0,1]$
Output: Subgroup bounds $lb, ub \in \{\mathbb{R} \cup \{-\infty, +\infty\}\}^n$

1 **for** $j \leftarrow 1$ **to** n **do** `// Start with unrestricted subgroup`
2 $\quad (lb_j^{\text{opt}},\ ub_j^{\text{opt}}) \leftarrow (-\infty, +\infty)$
3 $Q^{\text{opt}} \leftarrow Q(lb^{\text{opt}}, ub^{\text{opt}}, X, y)$
4 $(lb^{\text{peel}}, ub^{\text{peel}}) \leftarrow (lb^{\text{opt}},\ ub^{\text{opt}})$
5 **while** $\frac{m_b}{m} > \beta_0$ **do** `// Support threshold satisfied`
6 $\quad Q^{\text{cand}} \leftarrow -\infty$
7 $\quad$ **for** $j \in$ *get_permissible_feature_idxs(…)* **do**
8 $\quad\quad (lb,\ ub) \leftarrow (lb^{\text{peel}},\ ub^{\text{peel}})$ `// Try peeling lower bound`
9 $\quad\quad lb_j \leftarrow \text{quantile}(X_{\cdot j}, lb, ub, \alpha)$
10 $\quad\quad$ **if** $Q(lb, ub, X, y) > Q^{cand}$ **then**
11 $\quad\quad\quad (lb^{\text{cand}}, ub^{\text{cand}}) \leftarrow (lb, ub)$
12 $\quad\quad (lb,\ ub) \leftarrow (lb^{\text{peel}},\ ub^{\text{peel}})$ `// Try peeling upper bound`
13 $\quad\quad ub_j \leftarrow \text{quantile}(X_{\cdot j}, lb, ub, 1-\alpha)$
14 $\quad\quad$ **if** $Q(lb, ub, X, y) > Q^{cand}$ **then**
15 $\quad\quad\quad (lb^{\text{cand}}, ub^{\text{cand}}) \leftarrow (lb, ub)$
16 $\quad (lb^{\text{peel}}, ub^{\text{peel}}) \leftarrow (lb^{\text{cand}}, ub^{\text{cand}})$ `// Retain best candidate`
17 $\quad$ **if** $Q(lb^{peel}, ub^{peel}, X, y) > Q^{opt}$ **then** `// Update optimum`
18 $\quad\quad Q^{\text{opt}} \leftarrow Q(lb^{\text{peel}}, ub^{\text{peel}}, X, y)$
19 $\quad\quad (lb^{\text{opt}},\ ub^{\text{opt}}) \leftarrow (lb^{\text{peel}},\ ub^{\text{peel}})$
20 $(lb, ub) \leftarrow (lb^{\text{opt}}, ub^{\text{opt}})$
21 **for** $j \leftarrow 1$ **to** n **do** `// Reset non-excluding bounds`
22 $\quad$ **if** $lb_j = \min_{i \in \{1,\ldots,m\}} X_{ij}$ **then** $lb_j \leftarrow -\infty$
23 $\quad$ **if** $ub_j = \max_{i \in \{1,\ldots,m\}} X_{ij}$ **then** $ub_j \leftarrow +\infty$
24 **return** lb, ub

In the iterative peeling procedure (Lines 5–19), the algorithm generates new solution candidates by trying to restrict each *permissible feature* (Lines 7–15). By default, each feature is permissible, but the function *get_permissible_feature_idxs(…)* will become helpful once we introduce constraints. For each Feature j, the algorithm tests a new lower bound at the α-quantile of feature values in the subgroup and a new upper bound at the $1-\alpha$-quantile of feature values in the subgroup. The peeling fraction $\alpha \in (0,1)$ is a user parameter. It describes which fraction of data objects gets excluded from the subgroup in each peeling iteration. Having tested two new bounds for each feature, the algorithm

Algorithm 2: Generic beam search for subgroup discovery.

Input: Dataset $X \in \mathbb{R}^{m \times n}$,
Prediction target $y \in \{0,1\}^m$,
Subgroup-quality function $Q(lb, ub, X, y)$,
Beam width $w \in \mathbb{N}$
Output: Subgroup bounds $lb, ub \in \{\mathbb{R} \cup \{-\infty, +\infty\}\}^n$

1 **for** $l \leftarrow 1$ **to** w **do** // Initialize beam
2 **for** $j \leftarrow 1$ **to** n **do**
3 $(lb_j^{(\text{beam}, l)},\ ub_j^{(\text{beam}, l)}) \leftarrow (-\infty, +\infty)$ // Unrestricted
4 $cand\_has\_changed^{(l)} \leftarrow$ **true** // Subgroup should be updated
5 $Q^{(l)} \leftarrow Q(lb^{(\text{beam}, l)}, ub^{(\text{beam}, l)}, X, y)$
6 **while** $\left(\sum_{l=1}^{w} cand\_has\_changed^{(l)}\right) > 0$ **do** // Beam has changed
7 $prev\_cand\_changed\_idxs \leftarrow \{l \mid cand\_has\_changed^{(l)}\}$
8 **for** $l \leftarrow 1$ **to** w **do** // Create temporary solution candidates
9 $(lb^{(\text{cand}, l)},\ ub^{(\text{cand}, l)}) \leftarrow (lb^{(\text{beam}, l)},\ ub^{(\text{beam}, l)})$
10 $cand\_has\_changed^{(l)} \leftarrow$ **false**
11 **for** $l \in prev\_cand\_changed\_idxs$ **do** // Prepare beam updates
12 **for** $j \in get\_permissible\_feature\_idxs(\dots)$ **do**
13 evaluate_subgroup_updates(…) // Algorithm 3 or 4
14 **for** $l \leftarrow 1$ **to** w **do** // Update beam
15 $(lb^{(\text{beam}, l)},\ ub^{(\text{beam}, l)}) \leftarrow (lb^{(\text{cand}, l)},\ ub^{(\text{cand}, l)})$
16 $l \leftarrow \arg\max_{l \in \{1,\dots,w\}} Q^{(l)}$ // Select best subgroup from beam
17 $(lb,\ ub) \leftarrow (lb^{(\text{beam}, l)},\ ub^{(\text{beam}, l)})$
18 **for** $j \leftarrow 1$ **to** n **do** // Reset non-excluding bounds
19 **if** $lb_j = \min_{i \in \{1,\dots,m\}} X_{ij}$ **then** $lb_j \leftarrow -\infty$
20 **if** $ub_j = \max_{i \in \{1,\dots,m\}} X_{ij}$ **then** $ub_j \leftarrow +\infty$
21 **return** lb, ub

takes the subgroup with the highest associated quality (Line 16) and continues peeling it in the next iteration. Further, if this solution candidate improves upon the optimal solution candidate from all prior iterations, it is stored as the new optimum (Lines 17–19).

Beam Search Beam search is a generic search strategy that is also common in subgroup discovery [12]. It maintains a set of currently best solution candidates, i.e., the beam, which it iteratively updates. The number of solution candidates in the beam is a user parameter, i.e., the beam width $w \in \mathbb{N}$. We outline our specific implementation, which we denote as *Beam Search*, in Algorithm 2 combined with Algorithm 3. It is an adapted version of the beam-search implementation in the Python package *pysubgroup* [129].

Algorithm 3: evaluate_subgroup_updates(…) for *Beam Search*.

Input: Parameters and variables from Algorithm 2
Output: None; modifies variables from Algorithm 2 in-place

1 $(lb,\ ub) \leftarrow (lb^{(\text{beam},\, l)},\ lb^{(\text{beam},\, l)})$ `// Next, update lower bound`
2 **for** $b \in sort(unique(get\_feature\_values(X,\ j,\ lb^{(beam,\, l)},\ ub^{(beam,\, l)})))$ **do**
3 $\quad lb_j \leftarrow b$
4 $\quad$ **if** $\left(Q(lb, ub, X, y) > \min_{l \in \{1,\ldots,w\}} Q^{(l)}\right)$ ***and*** $(lb,\ ub) \notin \{(lb^{(cand,\, l)},\ ub^{(cand,\, l)}) \mid l \in \{1, \ldots, w\}\}$ **then**
5 $\quad\quad l \leftarrow \arg\min_{l \in \{1,\ldots,w\}} Q^{(l)}$ `// Replace worst candidate`
6 $\quad\quad (lb^{(\text{cand},\, l)},\ ub^{(\text{cand},\, l)}) \leftarrow (lb, ub)$
7 $\quad\quad cand\_has\_changed^{(l)} \leftarrow$ **true**
8 $\quad\quad Q^{(l)} \leftarrow Q(lb, ub, X, y)$
9 $(lb,\ ub) \leftarrow (lb^{(\text{beam},\, l)},\ lb^{(\text{beam},\, l)})$ `// Next, update upper bound`
10 **for** $b \in sort(unique(get\_feature\_values(X,\ j,\ lb^{(beam,\, l)},\ ub^{(beam,\, l)})))$ **do**
11 $\quad ub_j \leftarrow b$
12 $\quad$ **if** $\left(Q(lb, ub, X, y) > \min_{l \in \{1,\ldots,w\}} Q^{(l)}\right)$ ***and*** $(lb,\ ub) \notin \{(lb^{(cand,\, l)},\ ub^{(cand,\, l)}) \mid l \in \{1, \ldots, w\}\}$ **then**
13 $\quad\quad l \leftarrow \arg\min_{l \in \{1,\ldots,w\}} Q^{(l)}$ `// Replace worst candidate`
14 $\quad\quad (lb^{(\text{cand},\, l)},\ ub^{(\text{cand},\, l)}) \leftarrow (lb, ub)$
15 $\quad\quad cand\_has\_changed^{(l)} \leftarrow$ **true**
16 $\quad\quad Q^{(l)} \leftarrow Q(lb, ub, X, y)$

First, the algorithm *Beam Search* initializes the beam by creating w unrestricted subgroups (Lines 1–5). Further, it stores the quality of each of these subgroups. Additionally, it records which subgroups changed in the previous iteration (Lines 6–15) of the search. In particular, it stops once all subgroups in the beam remain unchanged (Line 6). Subsequently, it returns the best subgroup from the beam (Lines 16–21). As for *PRIM* (cf. Algorithm 1), we replace all non-excluding bounds with infinity as a post-processing step.

The main loop (Lines 6–15) updates the beam. In particular, for each subgroup that changed in the previous iteration, the algorithm creates new solution candidates by attempting to update the bounds of each feature separately (Lines 11–13). There are different options for this update step. Algorithm 3 outlines the specific update procedure for *Beam Search*, while *Best Interval* uses a slightly different one (cf. Algorithm 4). For *Beam Search*, the procedure tries to refine either the lower bound (Lines 1–8) or the upper bound (Lines 9–16) for a given Feature j. In particular, all unique feature values from data objects in the subgroup may act as new bounds. Each solution candidate that improves upon the minimum subgroup quality from the beam replaces the corresponding subgroup, unless it already is part of the beam due to another update action (Lines 4–8 and 12–16).

Algorithm 4: evaluate_subgroup_updates(…) for *Best Interval*.

Input: Parameters and variables from Algorithm 2
Output: None; modifies variables from Algorithm 2 in-place

1 $(lb,\ ub) \leftarrow (lb^{(\text{beam},\, l)},\ lb^{(\text{beam},\, l)})$ `// Value at index j will change`
2 $(lb^{\text{opt}},\ ub^{\text{opt}}) \leftarrow (lb^{(\text{beam},\, l)},\ lb^{(\text{beam},\, l)})$ `// (l, r) in [141]`
3 $Q^{\text{opt}} \leftarrow Q(lb^{\text{opt}}, ub^{\text{opt}}, X, y)$ `// WRAcc_max in [141]`
4 $Q^{\text{temp}} \leftarrow -\infty$ `// h_max in [141]`
5 $lb_j^{\text{temp}} \leftarrow -\infty$ `// t_max in [141]`
6 **for** $b \in sort(unique(get\_feature\_values(X,\ j,\ lb^{(beam,\, l)},\ ub^{(beam,\, l)})))$ **do**
7 $lb_j \leftarrow b$
8 $ub_j \leftarrow ub_j^{(\text{beam},\, l)}$
9 **if** $Q(lb, ub, X, y) > Q^{temp}$ **then**
10 $lb_j^{\text{temp}} \leftarrow b$
11 $Q^{\text{temp}} \leftarrow Q(lb, ub, X, y)$
12 $lb_j \leftarrow lb_j^{\text{temp}}$
13 $ub_j \leftarrow b$
14 **if** $Q(lb, ub, X, y) > Q^{opt}$ **then**
15 $(lb^{\text{opt}},\ ub^{\text{opt}}) \leftarrow (lb,\ ub)$
16 $Q^{\text{opt}} \leftarrow Q(lb, ub, X, y)$
17 **if** $\left(Q^{opt} > \min_{l \in \{1,\ldots,w\}} Q^{(l)}\right)$ ***and*** $(lb^{opt},\ ub^{opt}) \notin \{(lb^{(cand,\, l)},\ ub^{(cand,\, l)}) \mid l \in \{1, \ldots, w\}\}$ **then**
18 $l \leftarrow \arg\min_{l \in \{1,\ldots,w\}} Q^{(l)}$ `// Replace worst candidate`
19 $(lb^{(\text{cand},\, l)},\ ub^{(\text{cand},\, l)}) \leftarrow (lb^{\text{opt}}, ub^{\text{opt}})$
20 $cand\_has\_changed^{(l)} \leftarrow$ **true**
21 $Q^{(l)} \leftarrow Q^{\text{opt}}$

Best Interval *Best Interval* [141] offers an update procedure for subgroups (cf. Algorithm 4) that is tailored towards WRAcc (cf. Equation 2.3) as the subgroup-quality function. This update procedure can be used within a generic beam-search strategy (cf. Algorithm 2). As before, a solution candidate from an update step becomes part of the beam if it improves upon the worst subgroup quality there and is not a duplicate (Lines 17–21 in Algorithm 4).

However, solution candidates are generated differently than in the update procedure of *Beam Search* (cf. Algorithm 3). In particular, *Best Interval* updates lower and upper bounds for a given Feature j simultaneously rather than separately (Lines 1–16 in Algorithm 4). Thus, this procedure optimizes over all potential combinations of lower and upper bounds. However, it still only requires one pass over the unique values of Feature j rather than quadratic cost, as it leverages theoretical properties of the WRAcc function [141].

3. Related Work

In this chapter, we discuss related work. We organize this discussion according to the two main research gaps we address (cf. Section 1.2): Section 3.1 covers approaches for integrating constraints and domain knowledge, while Section 3.2 examines approaches for finding alternative solutions. In both sections, we elaborate on the fields of feature selection and subgroup discovery in detail before reviewing other related fields.

3.1. Integrating Constraints and Domain Knowledge

In this section, we discuss related work on considering constraints and domain knowledge in feature selection (cf. Section 3.1.1), subgroup discovery (cf. Section 3.1.2), and other related fields (cf. Section 3.1.3). Constraints for finding alternatives follow in Section 3.2.

3.1.1. Feature Selection

Constraint types and approaches Despite the number and variety of existing feature-selection methods (cf. Section 2.1.2), considering constraints and domain knowledge is less well-researched. Existing work typically has a narrower scope than ours since it tends to focus only on one feature-selection method and one constraint type rather than pursuing a more general approach. There is work on cost constraints [98, 154, 174, 179, 234] and cardinality constraints [108, 125, 197, 226]. Feature selection with predefined groups of features has been studied as well, though mainly specific to linear prediction models, e.g., group lasso and its variants [61, 97, 206, 229, 235]. As a more general work, [74] presents four wrapper-feature-selection methods that can theoretically support arbitrary constraint types. However, their evaluation only employs a narrow set of constraint types, related to lagged temporal features or a hierarchy of feature groups. Also, their wrapper approaches treat constraints as a black box, i.e., they only check whether constraints are satisfied and correspondingly exclude invalid candidate feature sets from the search. In contrast, we primarily use white-box solvers to consider constraints in the optimization. [161] introduces a wrapper approach supporting arbitrary constraint types and feature-selection methods. However, they also implement constraints as black-box functions. Apart from feature-set size, their constraints do not refer to feature sets per se but metrics for the whole machine-learning system, e.g., accuracy, fairness, or training time. Further, the constraints are not integrated into feature selection but are checked after training and evaluating a prediction model with the selected features.

Other notions of constrained feature selection Some subfields of data mining refer to 'constraints' in combination with 'feature selection' but mean different problems than we do. First, feature selection with constraints is studied in semi-supervised learning [91, 187, 200, 230]. However, the constraints there express relationships between data objects, not between features. For example, constraints may state whether two data objects belong to the same class, without assigning a class label. Second, constraints play a role in unsupervised feature selection [138, 232, 233]. These constraints do not directly express user needs on feature sets but help find a low-dimensional representation of the data. Third, there is constraint-based feature selection that builds on Bayesian network learning [119]. It does not involve user-defined constraints but conditional independence constraints between features, which are automatically learned and propagated.

3.1.2. Subgroup Discovery

White-box formulations To consider constraints in subgroup discovery, we provide an SMT formulation of the problem (cf. Section 7.3.1) and tackle it with a corresponding solver. To our knowledge, this formulation is novel. There are a few other white-box formulations of particular variants of subgroup discovery, e.g., constraint-programming formulations of Discriminative Itemset Mining [77, 111] and Relevant Subgroup Discovery [111], and integer-programming formulations of the Maximum Box problem [55], Maximum α-Pattern problem [30], and the Box Search problem [137]. Additionally, there are also white-box formulations for other prediction models [95] that share some similarities with subgroup descriptions, e.g., formulations in propositional logic (SAT) for decision trees, decision sets, and decision lists [160, 198, 227]. All previously mentioned works address different problem definitions than we do, e.g., use additional constraints but not the specific constraint types we analyze. Also, their evaluations differ from ours. For example, they do not compare against existing heuristic subgroup-discovery methods.

Constraint types There is work on considering different constraint types in subgroup discovery. Typically, constraints are integrated into algorithmic search methods rather than formulated declaratively for solver-based optimization. [147] mentions three common constraint types: lower bounds on subgroup quality, lower bounds on the number of subgroup members, and upper bounds on the search depth. The latter influences the number of features used. Several subgroup-discovery methods employ quality-based pruning to reduce the search space, e.g., using optimistic estimates in exhaustive search [8, 9, 73]. However, such automatically determined bounds on subgroup quality are not user-defined constraints. [10, 12, 13] provide a taxonomy and examples for knowledge-based constraint types in subgroup discovery. [13] expresses domain knowledge declaratively with the logic programming language Prolog but does not use a solver for optimization.

Feature-cardinality constraints Feature-cardinality constraints (cf. Definition 15) are one of the two constraint types we analyze in Chapter 7. While feature cardinality is a

well-known metric for subgroup complexity [88, 90, 215], our SMT formulation of this constraint type is novel. [133] formulates a quadratic program to select non-redundant features for subgroups, but with real-valued feature weights as decision variables and only as a subroutine within an algorithmic search. Two alternatives to feature-cardinality constraints are constraints [121] or post-processing [62] to eliminate irrelevant features.

Several works on subgroup discovery use a feature-cardinality constraint in their experiments [7, 121, 126, 127, 141], but there is a lack of studies that analyze the impact of different feature-cardinality thresholds on different subgroup-discovery methods broadly and systematically. [62, 130, 181] compare multiple feature-cardinality thresholds for one subgroup-discovery method each, while [88] compares multiple subgroup-discovery methods for one feature-cardinality threshold. [147] evaluates three subgroup-discovery methods and four feature-cardinality thresholds but mainly focuses their evaluation on comparing strategies for handling numeric data. Also, they use fewer and lower-dimensional datasets than we do. Further, they do not compare to an unconstrained setting.

3.1.3. Other Fields

Data mining and machine learning Besides feature selection and subgroup discovery, working with constraints is a research area for various subfields of data mining and machine learning [71], e.g., automated machine learning [162], clustering [48, 49], explainable AI [52, 70, 156, 202], and pattern mining [163, 204]. There is also work in the other direction, i.e., using machine-learning techniques in constraint solving [180].

Materials science Since we conduct a case study on constrained feature selection in materials science (cf. Chapter 5), we also review related work in this field. In materials science, various feature-selection methods have already been applied [1, 96, 101], but without considering constraints. At the same time, researchers argue for the integration of domain knowledge in machine learning [142, 218]. To this end, [42, 183] survey machine-learning approaches in materials science, some of which use domain knowledge when creating new features rather than when selecting them. For example, physical laws can guide which interaction terms should be created from the original features. As another example, 'fingerprint' feature vectors for describing materials have implicit physical constraints on feature values [93]. Apart from constraints, one can directly involve domain experts in feature selection. [135] lets domain experts rate the importance of features and combines these manual ratings with several automated feature-selection steps. Such an approach requires domain knowledge about the usefulness of individual features, while our constraints express knowledge about the relationships between features.

Software engineering There are approaches for constrained feature selection in software engineering [26, 65, 86]. Despite technical similarities, the semantics of *feature* and *feature-set quality* differ from our work, which limits the comparability of empirical studies.

A *feature* in software engineering is a characteristic of a software product, not a column in a tabular dataset (cf. Section 2.1.1). *Feature selection* then aims at configuring a software system, which may even be a machine-learning system [46]. *Feature models* express logical relationships, i.e., constraints, between features, often forming a hierarchical structure. Further, *feature attributes* express properties like component costs, memory requirements, etc. These properties form the base for further constraints or one or several target functions [80, 89, 191]. Typical approaches to find feature sets include sampling [172], constraint solving [222], adapting general-purpose optimization techniques [80], and combining satisfiability solving with general-purpose optimization [81, 82, 89].

Empirical studies on feature selection in software engineering often use feature models with predefined constraints from repositories like LVAT [199] and SPLOT [148]. Only some works also generate constrained feature models for a more systematic evaluation. [80, 171, 211] iteratively generate constraints, randomly picking constraint types and the features to be used, similar to our study with random constraints (cf. Chapter 4). However, these works use generated constraints to evaluate new approaches for dealing with constraints, while we focus on evaluating the impact of constraints themselves.

3.2. Finding Alternative Solutions

In this section, we discuss related work on finding alternative solutions in feature selection (cf. Section 3.2.1), subgroup discovery (cf. Section 3.2.2), and other related fields (cf. Section 3.2.3).

3.2.1. Feature Selection

Conventional feature selection Most feature-selection methods only yield one solution [31], though some exceptions exist [57, 158, 203]. However, none of the cited approaches searches for alternatives in our sense. For example, they pursue different objectives and give users less control, e.g., do not guarantee the dissimilarity of alternatives.

Ensemble feature selection Ensemble feature selection [189, 196] combines feature-selection results, e.g., obtained by different feature-selection methods or on different samples of the data. Fostering diverse feature sets may be a sub-goal to improve prediction performance [83, 134, 201, 225], but it is usually only an intermediate step. This focus differs from our notion of alternative feature sets, and users have less control over alternatives.

Statistically equivalent feature sets Methods for statistically equivalent feature sets [31, 119] use statistical tests to determine features or feature sets that are equivalent for predictions. Our notion of alternatives differs in several aspects. In particular, building optimal alternatives from equivalent feature sets is not straightforward. There can be arbitrarily

many equivalent feature sets, without an explicit quality-based ordering, while we provide a user-defined number of alternatives. Also, our alternatives need not have equivalent quality but should be optimal under constraints. Further, our dissimilarity threshold allows controlling the overlap between sets instead of eliminating all redundancies.

3.2.2. Subgroup Discovery

To our knowledge, alternative subgroup descriptions in the sense of this dissertation (cf. Definition 16) are a novel concept. In particular, we *maximize* the set similarity of contained data objects relative to a given subgroup while using a different subgroup description. In contrast, various existing approaches strive for alternatives in the sense of diverse or non-redundant sets of subgroups, which *minimize* the overlap of contained data objects without constraints on the description. After presenting such techniques for *subgroup-set selection*, we discuss approaches focusing on subgroup descriptions.

Subgroup-set selection Since subgroups only cover specific regions of the data, it is natural to search for multiple subgroups to cover multiple regions; see [8] for an overview of subgroup-set selection. The *covering* approach supports any subgroup-discovery method by removing all data objects contained in previous subgroups and repeating subgroup discovery [62]. Alternatively, one may reweigh [67, 122] or resample [194] data objects based on their subgroup membership. Other subgroup-discovery methods specifically search for multiple solutions simultaneously [126, 127, 130, 139, 181]. The notion of subgroup quality then typically goes beyond predictive quality like WRAcc and involves metrics for the diversity [25, 126, 127, 139] and number [88, 90, 215] of subgroups. One may also filter redundant subgroups as a post-processing step [32, 72, 94, 127].

Description-based diverse subgroup-set selection [126] introduces six strategies to foster diversity between subgroups. Two strategies refer to subgroup descriptions: The first excludes subgroup descriptions that have the same quality and differ in only one condition from an existing subgroup description. The second uses a global upper bound on how often a feature may be selected in a set of subgroup descriptions. Both these strategies give users less control over the overlap of subgroup descriptions than we do. Further, [126] optimizes subgroup quality in a simultaneous beam search. In contrast, we search for alternative descriptions sequentially, optimize similarity to the original subgroup, and consider a solved-based search method in addition to heuristic search.

Diverse top-k characteristics lists [136] introduces the notion of *diverse top-k characteristic lists*, which is a set of lists, each containing multiple patterns, e.g., subgroups. Within each list, the subgroups should be diverse in terms of data objects contained. Further, exactly the same subgroup description must not appear in two lists. However, any other partial overlap of descriptions is allowed, which is less specific than our definition.

Equivalent subgroup descriptions of minimal length [29] introduces the notion of *equivalent subgroup descriptions of minimal length*, which is stricter than our notion of alternative subgroup descriptions. In particular, the former descriptions need to cover exactly the same set of data objects instead of maximizing similarity. Further, a subset of the original feature set should be found, instead of using a different feature set subject to a dissimilarity constraint. The authors prove $\mathcal{NP}$-hardness, which influenced our proof of Proposition 26, and propose two algorithms for their problem but do not pursue a solver-based search.

Redescription mining Redescription mining aims to find pairs or sets of descriptions that cover exactly or approximately the same data objects [64, 182]. We pursue a similar goal but search for alternative descriptions sequentially instead of simultaneously. Also, we have a target variable while redescription mining is unsupervised [182]. Further, redescription mining works with different feature-set dissimilarity criteria than we do [64, 66, 150, 175], and the language for redescriptions may be more complex than for subgroups [64, 66]. Finally, most existing approaches for redescription mining are algorithmic rather than using white-box optimization [64, 150], though [78] provides a constraint-programming formulation. [150] shows that several variants of redescription mining are $\mathcal{NP}$-hard.

3.2.3. Other Fields

Clustering Finding alternative solutions has been addressed extensively in the field of clustering. [23] gives a taxonomy and describes corresponding algorithms. On a high level, this work inspired our notions of alternatives in feature selection and subgroup discovery. Nevertheless, the concrete problem definitions for alternatives are fundamentally different. First, the notion of dissimilarity differs, as we focus on feature sets rather than limiting the assignments of data objects to clusters [21, 22]. Second, using constraints is only one of several approaches to obtain alternative clusterings. Third, we are in a supervised prediction scenario (cf. Section 2) while clustering is unsupervised.

Subspace clustering and subspace search Finding multiple feature sets is part of subspace clustering [76, 92, 157] and subspace search [60, 166, 212]. These approaches strive to improve the results of data-mining algorithms by using subspaces, i.e., feature sets, rather than all features. Some approaches explicitly try to remove redundancy between subspaces [76, 92, 166] or foster subspace diversity [60, 212]. However, these subspace approaches differ from alternative feature selection and alternative subgroup descriptions in at least one of the following aspects: First, they pursue different optimization objectives, e.g., a different notion of quality. Second, definitions of subspace redundancy may consider data objects and feature values instead of only binary feature-selection decisions. Third, users may have less control over the dissimilarity of alternatives, e.g., only a regularization parameter rather than a hard threshold.

Explainable artificial intelligence (XAI) Alternative explanations in XAI can provide additional insights into predictions, enable users to develop and test different hypotheses, appeal to different kinds of users, and foster trust in the predictions [110, 220]. In contrast, significantly different explanations for the same prediction may raise doubts about how meaningful the explanations are [99]. Finding diverse explanations has been studied for various explainers, e.g., for counterfactuals [47, 102, 152, 156, 188, 217], criticisms [109], and semifactual explanations [5]. There are several approaches to foster diversity, e.g., ensembling different kinds of explanations [205], considering multiple local minima [217], using a search algorithm that maintains diversity [47], extending the optimization objective [5, 109, 156], or introducing constraints [102, 152, 188]. Of the various mentioned approaches, only [5, 152, 156] introduce a parameter to control the diversity of solutions. Of these three works, only [152] offers a user-friendly dissimilarity threshold in $[0, 1]$, while the other two approaches employ a regularization parameter in the objective.

Despite similarities, the previously mentioned XAI techniques tackle different problems than we do. In particular, they provide local explanations, i.e., aim at predictions for individual data objects, while we focus on the global quality of feature sets. For example, counterfactual explanations [75, 209, 216] alter feature *values as little as possible* to produce an alternative prediction *outcome*. In contrast, alternative feature sets might alter the feature *selection significantly* while trying to maintain the original prediction *quality*.

Rashomon sets A Rashomon set is a set of prediction models that reach a certain, e.g., close-to-optimal, prediction performance [59]. Despite similar performance, these models may still assign different feature importance scores, leading to different explanations [118]. Thus, Rashomon sets may yield partial information about alternative feature sets. However, approaches for Rashomon sets do not explicitly search for alternative feature sets as a whole but focus on the range of each feature's importance over prediction models. Further, our notion of alternatives is broader, as it is not tied to model-based feature importance.

Number partitioning Special cases of alternative feature selection resemble existing number-partitioning problems from theoretical computer science. Several restrictions of our problem are necessary to draw this connection, e.g., univariate feature-set quality and each feature being selected exactly once (cf. Section 6.2.4.2). Without restricting set sizes, a corresponding problem is called Multi-Way Number Partitioning or Multiprocessor Scheduling. It has multiple problem formulations [68, 114, 123], $\mathcal{NP}$-hardness results [68, 113], parameterized-complexity results [151], exact algorithms [85, 195, 219], and approximation algorithms [4, 51, 190, 224]. When restricting the set sizes, a related problem is called Balanced Number Partitioning or K-Partitioning. Again, there are $\mathcal{NP}$-hardness results [14, 87] as well as approximation algorithms [39, 105, 124, 149, 231]. Our $\mathcal{NP}$-hardness results and approximation algorithms for univariate alternative feature selection (cf. Section 6.2) build on these existing works. However, alternative feature selection generally has a much broader scope than number partitioning, allowing for more sophisticated objectives, additional parametrization options, and evaluation with prediction scenarios rather than only summing numbers.

4. Evaluating the Impact of Constraints on Feature-Selection Results

4.1. Overview

Scope Building on the motivation provided in Section 1.2, this chapter lays the foundations for constrained feature selection in general. In contrast, the subsequent three chapters will focus on specific constraint types. Besides formally introducing constrained feature selection, this chapter empirically evaluates the impact of constraints. Generally, adding constraints may lower feature-set quality. An important question is if there are sweet spots, i.e., high-quality feature sets that satisfy the constraints.

Contributions Our contribution in this chapter is twofold:

(1) We formalize constrained feature selection as an optimization problem and discuss how to solve it. Our problem definition is independent of the feature-selection method.

(2) We conduct a systematic, domain-independent study to analyze the impact of constraints on feature-selection results. To this end, we propose an experimental design that enables such an analysis. In particular, we use 35 regression datasets from various domains and generate random constraints systematically. We vary the constraint types, number of constraints, and features used. Further, we define evaluation metrics that quantify the constraints and the feature-selection results. We then empirically evaluate the relationships between these metrics. We find optimal feature sets with a Satisfiability Modulo Theories (SMT) solver, which supports a broad range of constraint types. In particular, we use univariate filter feature selection as the objective and formulate constraints in propositional logic and linear arithmetic. Section 4.4.4 summarizes key results.

Materials We publish all our code and experimental data online (cf. Section 1.4).

Prior works The content of this chapter bases on the following prior work:

- Jakob Bach et al. “An Empirical Evaluation of Constrained Feature Selection”. In: *SN Comput. Sci.* 3.6 (2022). DOI: 10.1007/s42979-022-01338-z

Chapter outline The remainder of this chapter is structured as follows: Section 4.2 describes and analyzes constrained feature selection. Section 4.3 outlines our experimental design. Section 4.4 presents the corresponding experimental results.

4.2. Constrained Feature Selection

In this section, we introduce the problem of constrained feature selection. After briefly defining the overall optimization problem (cf. Section 4.2.1), we describe potential constraints (cf. Section 4.2.2) and how to consider them (cf. Section 4.2.3). Finally, we discuss the time complexity of this problem (cf. Section 4.2.4).

4.2.1. Optimization Problem

For constrained feature selection, we keep the objective and decision variables of conventional feature selection (cf. Definition 1). I.e., we still optimize the feature-set quality $Q(s, X, y)$ for a dataset $X \in \mathbb{R}^{m \times n}$ and a prediction target $y \in Y^m$ by making feature-selection decisions $s \in \{0,1\}^n$. Thus, the general concept of constrained feature selection is independent of the feature-selection method, which determines the objective $Q(s, X, y)$. The novelty is admitting a set C of user-defined constraints for this optimization problem. Each constraint $c \in C$ maps the feature-selection decisions s to true (1) or false (0). We assume hard constraints, i.e., a valid feature set needs to satisfy all constraints:

$$\begin{aligned} \max_{s} \quad & Q(s, X, y) \\ \text{subject to:} \quad & \forall c \in C: \ c(s) = 1 \\ \text{with:} \quad & C \subseteq \{c \mid c : \{0,1\}^n \rightarrow \{0,1\}\} \end{aligned} \tag{4.1}$$

The full textual problem definition corresponding to Equation 4.1 is the following:

Definition 4 (Constrained feature selection)**.** *Given a dataset $X \in \mathbb{R}^{m \times n}$ with prediction target $y \in Y^m$, and a set of constraints $C \subseteq \{c \mid c : \{0,1\}^n \rightarrow \{0,1\}\}$,* constrained feature selection *is the problem of making feature-selection decisions $s \in \{0,1\}^n$ that maximize a given notion of feature-set quality $Q(s, X, y)$ while satisfying all constraints, i.e., $\forall c \in C : c(s) = 1$.*

4.2.2. Constraints

There is a broad range of possible constraint types. Limiting the feature-set size to a user-defined cardinality $k \in \mathbb{N}$, as common in many existing feature-selection methods, is only one possible constraint type. In general, users should ultimately decide which constraint types to employ. However, depending on the optimization method (cf. Section 4.2.3), it makes sense to limit the categories of admissible constraint types, e.g., by prescribing particular logics to formulate constraints. Such a limitation still offers more flexibility than

only allowing one particular constraint type, as common in related work (cf. Section 3.1.1). In our work, we admit arbitrary constraints in propositional logic and linear arithmetic. These two categories are domain-independent and allow the formulation of various expressive constraint types. In particular, all constraint types in our domain-specific case study (cf. Chapter 5) can be formulated that way. Additionally, propositional logic and linear arithmetic subsume typical constraint types from related work (cf. Chapter 3.1). Finally, these logics allow using a white-box solver for optimization (cf. Section 4.2.3).

Propositional constraints Propositional constraints describe relationships between features with propositional logic. The binary selection decisions $s_j \in \{0, 1\}$ correspond to the logical values $\{\text{false}, \text{true}\}$. One can now use logical operators like AND ($\wedge$), inclusive OR ($\vee$), exclusive OR ($\oplus$), NOT ($\neg$), IMPLIES ($\rightarrow$), and IFF ($\leftrightarrow$). For example, $s_1 \rightarrow s_2$ means that if Feature 1 is selected, then Feature 2 has to be selected as well. By combining the operators, one can describe more complex relationships. For example, $(s_4 \wedge s_7 \wedge s_{10}) \oplus (s_2 \wedge s_{11})$ means that either the three features from the first group or the two features from the second group have to be selected.

Arithmetic constraints The theory of linear arithmetic supports four operators [24]: addition (+), subtraction (−), multiplication ($\cdot$), and less-or-equal ($\leq$). In the terminology of first-order logic, the first three operators are functions, returning another arithmetic value, while the inequality is a predicate, yielding a logical value. Linear arithmetic does not allow multiplying two variables, as at least one operand has to be a constant [24]. In general, one can use linear-arithmetic constraints by assigning each feature a numeric value and formulating an inequality. For example, suppose that each feature has a weight $w_j \in \mathbb{R}_{\geq 0}$, e.g., measurement cost, and one wants to select a feature set with a total weight below a threshold $w_{\max} \in \mathbb{R}_{>0}$. This yields the following linear inequality:

$$\sum_{j=1}^{n} s_j \cdot w_j \leq w_{\max} \tag{4.2}$$

For instance, such a situation occurs if feature values are obtained with sensors with different energy consumption, and the total amount of energy per measurement is limited.

A cardinality constraint on the feature-set size is a special case of a weighted-sum constraint. It has uniform weights and a desired minimum, exact, or maximum cardinality $k \in \mathbb{N}$:

$$\sum_{j=1}^{n} s_j = k \tag{4.3}$$

One may use propositional and arithmetic constraints interchangeably in some situations. Think of a group of five features, and one wants to select at least one of them. Arithmetically, the sum of selection decisions for these five features should be at least one. With propositional logic, one may employ the OR ($\vee$) operator.

4.2.3. Optimization Methods

Depending on the formulation of the constrained optimization problem, different optimization methods are suitable. Conventional feature-selection methods are typically not designed to operate in a constrained search space; they support a cardinality constraint at most. Adapting individual feature-selection methods to individual constraint types is feasible in some cases but lacks generality. Instead, we propose to use a generic white-box or black-box optimizer to tackle constrained feature selection.

White-box optimizers consider the internal structure of the optimization problem. In particular, they must support formulating the chosen objective and constraint types. Nevertheless, they provide more flexibility than adapting existing feature-selection methods since they typically support a broad range of expressions. In particular, they allow users to combine arbitrary constraints within the supported logic declaratively.

In contrast, black-box optimization methods only process the objective value without knowing how its function is defined. This characteristic increases generality but also means that some information about the internals of the optimization problem remains unused. In some cases, constraint types may be black-box functions as well.

Our chapter on alternative feature selection (cf. Section 6.2.3) provides a detailed discussion on white-box and black-box optimization for different feature-selection methods. In the current chapter's experiments, we use univariate filter feature selection (cf. Equation 2.1) as the optimization objective. This simple linear objective is independent of the prediction model and allows us to use an off-the-shelf white-box solver to find optimal feature sets. In particular, we employ a Satisfiability Modulo Theories (SMT) optimizer [24]. Such solvers support constraints in propositional logic and first-order theories like arithmetic, bit vectors, or arrays. We already decided to formulate constraints in propositional logic and linear arithmetic (cf. Section 4.2.2), which fits the scope of SMT. The objective function in Equation 2.1 is also linear once the feature qualities $q(X_{\cdot j}, y)$ are computed.

If all constraints are in propositional logic, then the optimization problem with this objective becomes a weighted Max One problem [106]. This is a special case of a weighted partial maximum satisfiability (MaxSAT) problem [15, 131], where the individual binary decision variables form the soft clauses for optimization and the user-defined constraints constitute the hard clauses. Such a problem formulation gives way to specialized MaxSAT solvers. If we include arithmetic constraints, the formulation is a MaxSMT problem [168].

4.2.4. Time Complexity

We analyze three aspects of time complexity for constrained feature selection: the size of the search space for exhaustive search, parameterized complexity, and $\mathcal{NP}$-hardness. While the results here are relatively straightforward, they also lay the foundation for further complexity analyses regarding alternative feature selection (cf. Section 6.2.4) and constrained subgroup discovery (cf. Sections 7.3.2.5 and 7.3.3.5).

Exhaustive search Without constraints, the search space of feature selection grows exponentially with the number of features n. In particular, there are $2^n - 1$ possibilities to form a non-empty feature set of arbitrary size. In the worst case, finding the optimal feature set requires iterating over this entire space. In practice, particular notions of feature-set quality may allow finding the optimum faster, or a feature-selection method may only strive for an approximate solution.

Constraints can reduce the search space, but the upper bound remains the same:

Proposition 1 (Complexity of exhaustive search for (constrained) feature selection)**.** *An exhaustive search for feature selection with or without constraints (cf. Definitions 1 and 4) needs to evaluate $O(2^n)$ feature sets.*

Even if constraints reduce the search space, an essential question is whether one can efficiently enumerate all valid feature sets. The latter is possible for cardinality constraints, which have $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!} \leq n^k$ solution candidates for a fixed feature-set size k:

Proposition 2 (Complexity of exhaustive search for feature selection with cardinality constraint)**.** *An exhaustive search for feature selection (cf. Definition 1) with an exact cardinality constraint (cf. Equation 4.3) needs to evaluate $O(n^k)$ feature sets.*

Parameterized complexity Depending on the constraint types employed, the size of the search space may no longer be exponential in n but may depend on other parameters. This situation occurs for cardinality constraints: If we assume $k \ll n$, $k \in O(1)$, i.e., k being a small constant, independent from n, then the complexity in Proposition 2 is polynomial in n. This assumption makes sense if one wants to obtain a small feature set from a high-dimensional dataset, which is a typical goal of feature selection. However, the exponent k may still render an exhaustive search practically infeasible. In terms of parameterized complexity, the problem resides in class $\mathcal{XP}$ since the complexity term has the form $O(f(k) \cdot n^{g(k)})$ [54], here with parameter k and functions $f(k) = 1, g(k) = k$.

Proposition 3 (Parameterized complexity of feature selection with cardinality constraint)**.** *The problem of feature selection (cf. Definition 1) with an exact cardinality constraint (cf. Equation 4.3) resides in the parameterized complexity class $\mathcal{XP}$ for the parameter k.*

NP-Hardness The constraint types also influence whether constrained feature selection resides in complexity class $\mathcal{P}$ or $\mathcal{NP}$. In general, even determining whether there is at least one valid solution candidate is $\mathcal{NP}$-hard for propositional logic [44] and SMT [24]. In particular, only a few categories of propositional formulas guarantee a polynomial-time check for satisfiability [192]. Further, optimization problems are generally at least as hard as the corresponding decision problems [68]. Thus, we obtain the following proposition:

Proposition 4 (Complexity of constrained feature selection with propositional constraints)**.** *Assuming an arbitrary feature-set quality function $Q(s, X, y)$ whose value can be computed in polynomial time, the problem of constrained feature selection (cf. Definition 4) with arbitrary constraints in propositional logic is $\mathcal{NP}$-complete.*

This result is not tied to a particular feature-selection method. Even with a univariate objective (cf. Equation 2.1) and propositional constraints, optimization is hard in general:

Proposition 5 (Complexity of constrained feature selection with univariate objective and propositional constraints)**.** *Assuming univariate feature qualities (cf. Equation 2.1), the problem of constrained feature selection (cf. Definition 4) with arbitrary constraints in propositional logic is $\mathcal{NP}$-complete.*

In particular, this weighted Max One problem only resides in $\mathcal{P}$ for a few categories of constraint types, but in $\mathcal{NP}$ for all others, and even finding an approximate solution with guaranteed quality may be hard [106]. One polynomial-time case concerns the cardinality constraint. If the optimization problem only involves this constraint type, one can sort the univariate feature qualities and pick the k highest ones to obtain the exact optimum:

Proposition 6 (Complexity of feature selection with univariate objective and cardinality constraint)**.** *Assuming univariate feature qualities (cf. Equation 2.1), the problem of feature selection (cf. Definition 1) with an exact cardinality constraint (cf. Equation 4.3) has a time complexity of $O(n \cdot \log n)$.*

4.3. Experimental Design

In this section, we introduce our experimental design for studying the impact of constraints. We start with a brief overview (cf. Section 4.3.1). Next, we describe the components of the experimental design: constraints (cf. Section 4.3.2), objective function (cf. Section 4.3.3), prediction models (cf. Section 4.3.4), evaluation metrics (cf. Section 4.3.5), and datasets (cf. Section 4.3.6). Finally, we briefly outline our implementation (cf. Section 4.3.7).

4.3.1. Overview

We evaluate the impact of constraints on feature-selection results with 35 regression datasets. For our analyses, we repeatedly generate random constraints of ten different types and optimize the objective of univariate filter feature selection. We use four evaluation metrics to quantify constraints and three evaluation metrics to quantify feature-selection results. We analyze how the evaluation metrics relate to each other and how the impact of constraints differs between constraint types.

4.3.2. Constraints

The space of potential constraints is vast. To obtain general insights, we systematically generate constraints with many repetitions, varying three aspects. First, the *generation method* varies the features involved in constraints, i.e., the operands of the constraints. Second, the *constraint type* defines the operator for constraints, e.g., AND, OR, etc. Third, we conduct 10-fold cross-validation on multiple datasets (cf. Section 4.3.6).

Algorithm 5: Constraint-generation method for evaluating the constraints' impact.

Input: Constraint type t,
Optimization problem o,
Number of iterations $n\_iters \in \mathbb{N}$
Output: Evaluation metrics from Section 4.3.5

```
1  for iters ← 1 to n_iters do
2      o′ ← o                                                  // make copy
3      C ← ∅                                                   // set of constraints
4      num_co ← Choose num_co ∈ {1, ..., 10} uniformly random
5      for con_iters ← 1 to num_co do
6          if t is 'single' constraint then
7              n′ ← 2                                          // number of features in constraint
8          else if t is 'group' constraint then
9              n′ ← Choose n′ ∈ {2, ..., n} uniformly random
10         F ← Choose n′ distinct features uniformly random
11         c ← Apply t to F                                    // one constraint
12         C ← C ∪ {c}
13     o′ ← Add C to o′
14     Solve o′
15     Evaluate o′
```

Generation method We employ the same generation method for all constraint types except for two, which we discuss later. The generation method varies several characteristics of a constraint set: the number of constraints, the number of features in each constraint, and the actual features in each constraint. This flexibility gives way to a broad coverage of the evaluation space but also causes considerable variance. Consequently, we repeat the constraint generation $n\_iters = 1000$ times per combination of constraint type, dataset, and cross-validation split. We call each iteration a *constraint-generation run*.

Algorithm 5 outlines the process of generating and evaluating constraints. The initial optimization problem o only consists of the objective function (cf. Section 4.3.3) initialized for the current cross-validation split and dataset. In each constraint-generation run, we vary the number of constraints uniformly at random between one and ten (Line 4). Next, we decide on the number of features per constraint (Lines 6–9). Our *single* constraint types always involve two features, while *group* constraint types support larger sets, with the group size $n' \in \{2, \ldots, n\}$ chosen uniformly at random. We choose the features involved in a constraint uniformly random as well, sampling without replacement (Line 10). After creating the constraints, we add them to the optimization problem, run the optimizer, and compute the evaluation metrics (cf. Section 4.3.5).

Constraint types To specify constraint types, we can select from a broad range of possible operators. Also, combining operators can yields further operators. For example, NAND can

be expressed with an AND and a NOT. Thus, exhaustively evaluating all possible constraint types is infeasible. Additionally, the datasets in this study (cf. Section 4.3.6) come from different domains. Thus, it is impossible to formulate a common set of domain-specific constraints. Instead, we use ten generic constraint types. When compiling the subsequent list, we aimed at simple yet diverse constraint types.

When reading the following formulas, consider these hints: Each constraint type either refers to two features with indices $\{j_1, j_2\}$, a group of features with indices $\{j_1, \ldots, j_{n'}\}$, or all n features with indices $\{1, \ldots, n\}$. Some constraint types have an additional parameter $k \in \mathbb{N}$ for the feature-set size. For illustration purposes, we sometimes give two equivalent definitions. Generally, our implementation may differ slightly but is logically equivalent.

(T1) *Global-AT-MOST*: From the set of all n features, select at most k features. As this constraint type always refers to all features instead of a random subset, we do not use Algorithm 5 for generation. Instead, we evaluate all possible values of k exhaustively.

$$\text{Global-AT-MOST}(\{s_1, \ldots, s_n\}, k) = \sum_{j=1}^{n} s_j \leq k, \text{ with } k \in \{1, \ldots, n-1\} \tag{4.4}$$

(T2) *Group-AT-MOST*: From a subset of features of size n', select at most k. We choose $k \in \{1, \ldots, n'-1\}$ uniformly at random.

$$\text{Group-AT-MOST}(\{s_{j_1}, \ldots, s_{j_{n'}}\}, k) = \sum_{l=1}^{n'} s_{j_l} \leq k, \text{ with } k \in \{1, \ldots, n'-1\} \tag{4.5}$$

(T3) *Group-AT-LEAST*: From a subset of features of size n', select at least k. Again, we choose $k \in \{1, \ldots, n'-1\}$ uniformly at random. This constraint type alone does not exclude the trivial solution of selecting all features. Thus, we combine it with (T1), requiring that at most half of all features are selected globally.

$$\text{Group-AT-LEAST}(\{s_{j_1}, \ldots, s_{j_{n'}}\}, k) = \sum_{l=1}^{n'} s_{j_l} \geq k, \text{ with } k \in \{1, \ldots, n'-1\} \tag{4.6}$$

(T4) *Single-IFF*: From a pair of features, select either both or none. To exclude the trivial solution of selecting all features, we combine this constraint type with (T1), requiring that at most half of all features are selected globally.

$$\text{Single-IFF}(s_{j_1}, s_{j_2}) = s_{j_1} \leftrightarrow s_{j_2} = (s_{j_1} \wedge s_{j_2}) \vee (\neg s_{j_1} \wedge \neg s_{j_2}) \tag{4.7}$$

(T5) *Group-IFF*: An extension of (T4): From a subset of features of size n', either select all or none. To exclude the trivial solution, we combine this constraint type with (T1), requiring that at most half of all features are selected globally.

$$\begin{aligned} \text{Group-IFF}(\{s_{j_1}, \ldots, s_{j_{n'}}\}) &= s_{j_1} \leftrightarrow s_{j_2} \leftrightarrow \cdots \leftrightarrow s_{j_{n'}} \\ &= (s_{j_1} \wedge s_{j_2} \wedge \cdots \wedge s_{j_{n'}}) \vee (\neg s_{j_1} \wedge \neg s_{j_2} \wedge \cdots \wedge \neg s_{j_{n'}}) \end{aligned} \tag{4.8}$$

(T6) *Single-NAND*: From a pair of features, do not select both simultaneously. This constraint type is a special case of (T2) with $n' = 2$ and $k = 1$.

$$\text{Single-NAND}(s_{j_1}, s_{j_2}) = \neg(s_{j_1} \wedge s_{j_2}) \tag{4.9}$$

(T7) *Group-NAND*: From a subset of features of size n', select at most $n' - 1$.

$$\text{Group-NAND}(\{s_{j_1}, \ldots, s_{j_{n'}}\}) = \neg(s_{j_1} \wedge s_{j_2} \wedge \cdots \wedge s_{j_{n'}}) = \sum_{l=1}^{n'} s_{j_l} \leq n' - 1 \tag{4.10}$$

(T8) *Single-XOR*: From a pair of features, select exactly one.

$$\text{Single-XOR}(s_{j_1}, s_{j_2}) = s_{j_1} \oplus s_{j_2} = (s_{j_1} \wedge \neg s_{j_2}) \vee (\neg s_{j_1} \wedge s_{j_2}) \tag{4.11}$$

(T9) *Group-MIXED*: For each constraint to be generated, pick with equal probability between (T2), (T3), (T5), (T7) and (T8). There is no global cardinality constraint (T1).
(T10) *UNCONSTRAINED*: As a reference point, we also consider the unconstrained solution, i.e., the upper bound for the objective value. We only compute this once for each dataset and cross-validation split instead of using Algorithm 5 since there are no random effects or parameters that would warrant repetitions. This constraint type is always satisfied.

$$\text{UNCONSTRAINED}(\{s_1, \ldots, s_n\}) = 1 \tag{4.12}$$

4.3.3. Objective Function and Optimization

We optimize the univariate objective from Equation 2.1. To this end, we compute the feature qualities on the training set of the current dataset and cross-validation split. For the dependency measure $q(\cdot)$, we take mutual information [115], which can capture non-linear relationships in the data. We obtained similar insights in experiments with the absolute value of Pearson correlation. For optimization, we use the SMT solver *Z3* [28, 50], which is popular in related work in software engineering (cf. Section 3.1.3).

4.3.4. Prediction

After selecting features under constraints, we use the resulting feature sets in regression models. We choose four models with different learning paradigms, i.e., linear and tree-based, and different levels of complexity: linear regression (*Lin*), regression tree (*Tree*), boosted linear regression (*B-Lin*), and boosted trees (*B-Tree*). We leave the hyperparameters of the models mostly at their defaults but set random seeds to enable reproducibility. We set the ensemble size for the two boosting models to 20 estimators.

4.3.5. Evaluation Metrics

In this section, we outline our evaluation metrics for this study. After a brief overview (cf. Section 4.3.5.1), we present metrics describing the constraints (cf. Section 4.3.5.2) and feature-selection results (cf. Section 4.3.5.3).

4.3.5.1. Overview

Since we want to evaluate the impact of constraints on feature-selection results, we employ two kinds of metrics: metrics describing the constraints and metrics describing the feature-selection results. To describe the constraints, we consider four metrics, i.e., *number of constraints* num_{co}, *fraction of constrained features* $frac_{\text{cf}}$, *fraction of unique constrained features* $frac_{\text{ucf}}$, and *fraction of solutions* $frac_{\text{so}}$. To describe the feature-selection results, we use three metrics, i.e., *fraction of selected features* $frac_{\text{se}}$, *objective value* Q, and *prediction performance* R^2. We mostly normalize metrics to $[0, 1]$ to account for differences between the datasets, e.g., in the numbers of features and the distribution of feature qualities.

4.3.5.2. Metrics Describing Constraints

By nesting logical and arithmetic formulas, constraints can become arbitrarily complex. Thus, one can come up with many metrics describing the constraints. In the field of SAT solving, there are dozens of metrics to characterize logical formulas [3, 170]. However, many of these metrics require formulas in a specific form, i.e., conjunctive normal form (CNF). Further, many metrics capture rather complex concepts, e.g., referring to a graph representation of the logical formula. In this study, we use a small set of metrics that have simple interpretations and describe constraints at different granularity levels.

Number of constraints num_{co} This metric is very coarse-grained, as it neither describes the strength of individual constraints nor the interactions between constraints. For a set C of constraints, the metric returns its size:

$$num_{\text{co}} = |C| \tag{4.13}$$

Fraction of constrained features $frac_{\text{cf}}$ For this metric, we count the decision variables in the formulas representing the constraints, i.e., how many features appear in the latter. To that end, let *feature_list(c)* be a function that returns a list with the features that appear in the formula of the constraint c. If a feature occurs several times in a constraint or a set of constraints, we count it that many times. For example, the constraint set $C = \{s_1 \leftrightarrow s_2, s_1 \vee s_3, s_4 \rightarrow s_5\}$ refers to Feature 1 twice. For normalization, we divide by the number of features in the dataset, though the metric can still be greater than 1:

$$frac_{\text{cf}} = \frac{\sum_{c \in C} |\text{feature\_list}(c)|}{n} \tag{4.14}$$

Fraction of unique constrained features $frac_{\text{ucf}}$ Different from the previous metric, each feature is only counted once, even if it appears in a constraint or a set of constraints more than once. Thus, the range of the metric becomes $[0, 1]$:

$$frac_{\text{ucf}} = \frac{\left| \bigcup_{c \in C} \text{feature\_set}(c) \right|}{n} \tag{4.15}$$

Fraction of solutions $frac_{so}$ This metric quantifies the search space under the constraints, i.e., the fraction of valid feature sets. It is the most fine-grained metric describing the constraints. However, it is costly to compute, as it requires checking for each of the 2^n possible feature sets whether it satisfies the constraints. This effect limits the dimensionality of the datasets in our evaluation (cf. Section 4.3.6). We define the metric as follows:

$$frac_{\text{so}} = \frac{\sum_{s \in \{0,1\}^n} \min_{c \in C} c(s)}{2^n} \tag{4.16}$$

In particular, we count how many values of the feature-selection decisions s satisfy all constraints. Each constraint $c(s) \in C$ evaluates to zero (false) or one (true). Mathematically, all constraints are satisfied if the minimum of all constraint evaluations equals one. Finally, dividing by the total number of possible feature sets normalizes the metric to $[0, 1]$.

4.3.5.3. Metrics Describing Feature-Selection Results

We analyze the selected features and the feature-set quality.

Fraction of selected features $frac_{se}$ This metric quantifies the size of the solution, i.e., feature set. We sum over all selection decisions s_j and normalize the metric to $[0, 1]$:

$$frac_{\text{se}} = \frac{\sum_{j=1}^{n} s_j}{n} \tag{4.17}$$

In a case study, one can also evaluate the selected features qualitatively, i.e., which domain-specific conclusions one can draw if particular features are selected or not.

Objective value Q_{norm} The objective guides the search for the optimal feature set (cf. Equation 2.1 and Section 4.3.3) and is computed on the training set of the current dataset and cross-validation fold. To normalize this metric to $[0, 1]$, we divide by the value achievable in the unconstrained problem, i.e., summing all feature qualities:

$$Q_{\text{norm}} = \frac{Q(s, X, y)}{\sum_{j=1}^{n} q(X_{\cdot j}, y)} = \frac{\sum_{j=1}^{n} s_j \cdot q(X_{\cdot j}, y)}{\sum_{j=1}^{n} q(X_{\cdot j}, y)} \tag{4.18}$$

If no valid feature set exists, i.e., an infeasible optimization problem, we set Q_{norm} to 0.

Prediction performance R^2 As an alternative metric for feature-set quality, we train regression models with the features selected under constraints (cf. Section 4.3.4). We report prediction performance in terms of R^2, i.e., the coefficient of determination, which is a standard evaluation metric for regression problems. If not stated otherwise, prediction performance refers to the test set. With $\hat{y} \in \mathbb{R}^m$ denoting the prediction, $y \in \mathbb{R}^m$ the true target variable, and $\bar{y}$ the mean of y, R^2 is defined as follows [100]:

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2} \tag{4.19}$$

Though R^2 already has a natural upper bound of 1, it may vary considerably between datasets and prediction models, as we analyze in Section 4.4.1. Thus, for some evaluations, we use $R^2_{\text{norm}} \in [0, 1]$, defined as the min-max-normalization of R^2 by prediction model, dataset, and cross-validation fold. Finally, we set R^2_{norm} to 0 if no valid feature set exists.

4.3.6. Datasets

This study is domain-independent. We take 35 regression datasets from the OpenML repository [214], listed in Table 4.1. We only select and filter the datasets by technical criteria and do not conduct further pre-processing steps. First, we exclude datasets with missing values since dealing with the latter is orthogonal to our work. Second, we use medium-sized datasets with between 100 and 10,000 data objects. This number does not affect the optimization time since feature-quality values can be pre-computed once per dataset and cross-validation split. However, dataset size affects the training time for prediction models, which incurs separately for each constraint-generation run. Third, we choose datasets with 10 to 14 numeric features, which might seem relatively low. As we show in our case study in materials science (cf. Chapter 5), the solver we use can handle larger optimization problems. However, counting the fraction of valid solutions under constraints $frac_{\text{so}}$, a key evaluation metric, becomes infeasible. Also, we obtained similar results in preliminary experiments with high-dimensional data.

4.3.7. Implementation and Execution

We implemented our experimental pipeline in Python 3.7, using *scikit-learn* [176] and *xgboost* [41] for machine learning and the SMT solver *Z3* [28, 50] for optimization. All code is available online (cf. Section 1.4). We organized the constrained-feature-selection functionality as a Python package to ease reuse.

Our experimental pipeline parallelizes over datasets and constraint types, while each of these experimental tasks runs single-threaded. We ran the pipeline on a server with 128 GB RAM and an *AMD EPYC 7551* CPU, having 32 physical cores and a base clock of 2.0 GHz. With this hardware, the parallelized pipeline run took approximately 7 hours.

4.4. Evaluation

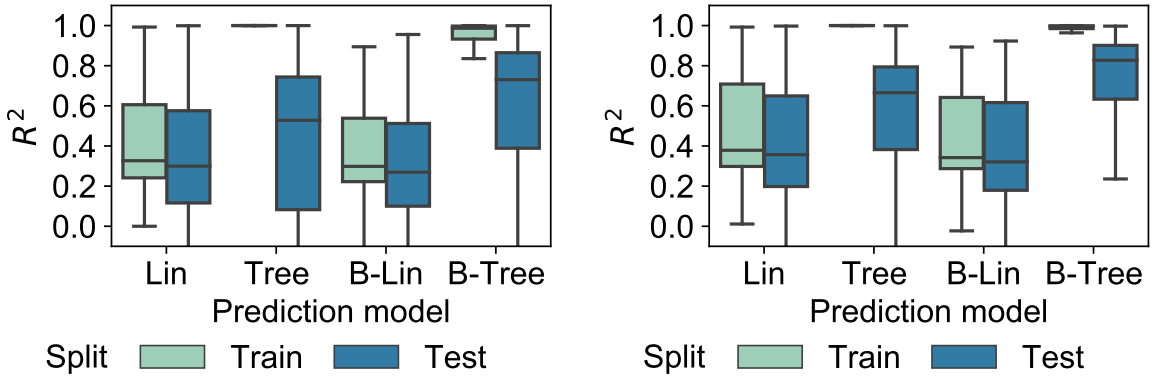
In this section, we evaluate our experiments on the impact of constraints. After comparing prediction models (cf. Section 4.4.1), we tackle the relationship between evaluation metrics (cf. Section 4.4.2) and the impact of constraint types (cf. Section 4.4.3). Finally, we summarize key findings (cf. Section 4.4.4).

Table 4.1.: Datasets from OpenML used in our experiments. m denotes the number of data objects and n the number of features.

Dataset	m	n
auml_eml_1_d	4585	10
auto_price	159	14
bodyfat	252	14
boston	506	11
chatfield_4	235	12
cpu_small	8192	12
Diabetes(scikit-learn)	442	10
forest_fires	517	10
fri_c0_1000_10	1000	10
fri_c0_100_10	100	10
fri_c0_250_10	250	10
fri_c0_500_10	500	10
fri_c1_1000_10	1000	10
fri_c1_100_10	100	10
fri_c1_250_10	250	10
fri_c1_500_10	500	10
fri_c2_1000_10	1000	10
fri_c2_100_10	100	10
fri_c2_250_10	250	10
fri_c2_500_10	500	10
fri_c3_1000_10	1000	10
fri_c3_100_10	100	10
fri_c3_250_10	250	10
fri_c3_500_10	500	10
fri_c4_1000_10	1000	10
fri_c4_100_10	100	10
fri_c4_250_10	250	10
fri_c4_500_10	500	10
ilpd-numeric	583	10
plasma_retinol	315	10
pwLinear	200	10
rmftsa_ladata	508	10
SWD	1000	10
wind	6574	14
wine_quality	6497	11

4.4.1. Comparison of Prediction Models

Before studying the impact of constraints, we analyze prediction performance regarding the impact of the prediction model and dataset. Figure 4.1a shows a considerable variation



(a) All experimental runs.

(b) Unconstrained experimental runs.

Figure 4.1.: Distribution of prediction performance over datasets, cross-validation folds, and constraint-generation runs, by prediction model. Y-axes are truncated and outliers are removed to improve readability.

in prediction performance for each model. This plot aggregates all experimental runs, i.e., different datasets, cross-validation folds, constraint types, and constraint-generation repetitions. However, Figure 4.1b displays that there is still a considerable variation in prediction performance in the unconstrained (T10) runs. I.e., the datasets have a substantial impact on prediction performance. We also see overfitting, i.e., the test-set performance is lower than the training-set performance, particularly for tree-based models. In the following sections, we focus on test-set performance. Further, we min-max normalize R^2 per dataset, cross-validation fold, and prediction model to focus on the relative impact of constraints, ignoring differences in dataset difficulty and model complexity. Finally, we limit our analyses to two prediction models: linear regression (*Lin*), the simplest model we use, and gradient-boosted trees (*B-Tree*), which yield the best average performance.

4.4.2. Relationship Between Evaluation Metrics

We study the relationships between evaluation metrics in two ways: We analyze the correlation between metrics and also plot the metrics against each other. For the correlation analysis, we take the values of each evaluation metric for all experimental runs and compute the Spearman rank correlation between these vectors. The resulting correlation matrix (cf. Figure 4.2) shows a strong positive correlation between the fraction of selected features $frac_{\text{se}}$, the fraction of solutions $frac_{\text{so}}$, the objective value Q_{norm}, and the prediction performance R^2_{norm}. All pairwise correlations between these metrics are at least 0.7.

For a more detailed view, Figure 4.3a shows a roughly linear relationship between the fraction of selected features $frac_{\text{se}}$ and the objective value Q_{norm}. The linear objective function in our experiments (cf. Section 4.3.3) makes this trend plausible. However, the quality of the selected features and thus the objective value for a fixed feature-set size can still vary since different constraints may exclude different feature combinations.

Figure 4.3b shows that the objective value Q_{norm} and the prediction performance R^2_{norm} may lead to different assessments of feature-set quality. The chosen objective function

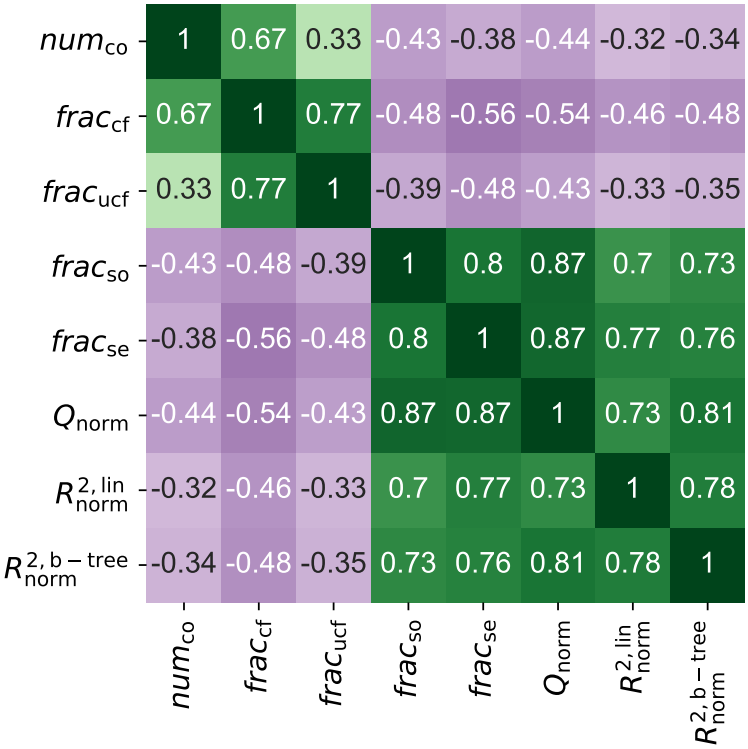
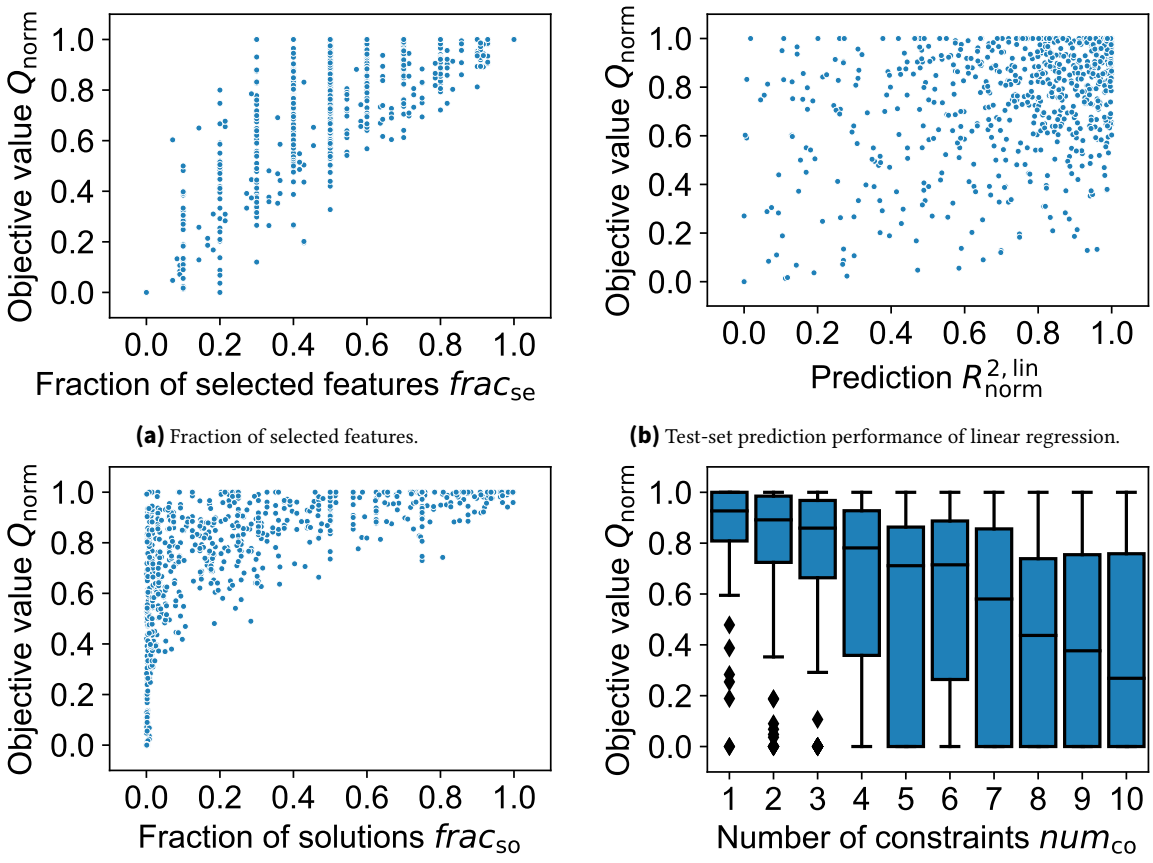


Figure 4.2.: Spearman correlation between evaluation metrics, over datasets, cross-validation folds, and constraint-generation runs.

cannot describe interactions between features, as it only sums up the individual feature qualities. This caveat is a general characteristic of univariate filter feature selection. In contrast, prediction models evaluate a feature set as a whole.

Figure 4.3c shows that the relationship between the fraction of solutions $frac_{so}$ and objective value Q_{norm} is non-linear. Decreasing the fraction of valid feature sets tends to decrease the objective value. However, it also matters which feature sets become invalid since the optimizer might find feature sets with a high objective value even in small search spaces, depending on the concrete constraints and the distribution of feature qualities. In contrast, we have not observed any scenario with a large search space but a low objective value.

Coming back to the correlation matrix in Figure 4.2, there is a moderately negative correlation between the fraction of solutions $frac_{so}$ on the one side and the number of constraints num_{co}, fraction of constrained features $frac_{cf}$, and fraction of unique constrained features $frac_{ucf}$ on the other side. This effect is expected since stronger constraint sets prune more solutions. The correlation is only moderate, though, since the latter three metrics do not consider the constraint type. E.g., for *Group-NAND* (T7), involving more features even makes the constraint weaker. Further, the latter three metrics only show a moderately negative correlation to feature-set quality since they do not consider the individual feature qualities at all. For example, Figure 4.3d shows the high variation in the relationship between the number of constraints num_{co} and the objective value Q_{norm}.



(a) Fraction of selected features.

(b) Test-set prediction performance of linear regression.

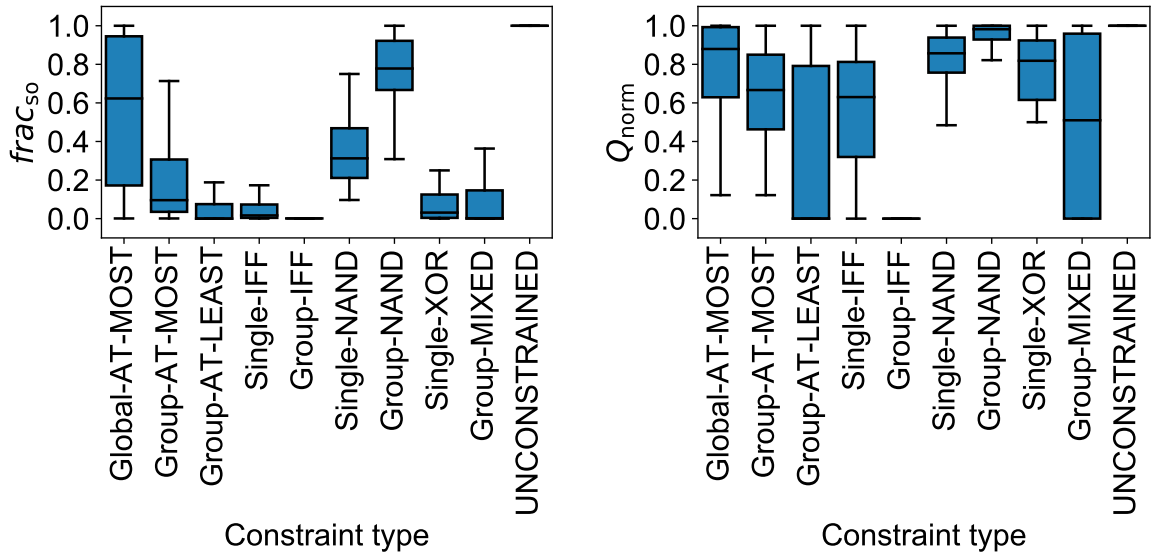
(c) Fraction of solutions.

(d) Number of constraints.

Figure 4.3.: Relationship of objective value Q_{norm} to other evaluation metrics. We have randomly sampled 1000 constraint-generation runs to keep the plot sizes and the time for rendering these plots reasonable.

4.4.3. Impact of Constraint Types

Figure 4.4 shows how two evaluation metrics, i.e., fraction of solutions $frac_{\text{so}}$ and objective value Q_{norm}, vary between constraint types. Normalized prediction performance exhibits similar trends as the objective value. Overall, the impact of constraints strongly depends on the constraint type. Without constraints, i.e., for the type *UNCONSTRAINED* (T10), all feature sets are valid, and the objective value is maximal. *Global-AT-MOST* (T1) has the largest inter-quartile range of the fraction of solutions (cf. Figure 4.4a). This effect is expected since we systematically generate only one constraint of this type for each possible feature-set size, while the remaining constraint types employ up to 10 constraints (cf. Algorithm 5) and thereby typically limit the search space more. Thus, *GROUP-AT-MOST* (T2) exhibits a smaller range and lower median of $frac_{\text{so}}$ than *Global-AT-MOST* (T1). The type *Group-MIXED* (T9) shows the widest distribution of the objective value (cf. Figure 4.4b) since it combines different other constraint types. *Single-NAND* (T6) and *Group-NAND* (T7) exhibit only a small decrease in objective value since *NAND* only excludes selecting all involved features simultaneously. The more features are involved, the less restrictive the constraint becomes. In contrast, *IFF* (T4) (T5) constraints become stronger when applied to



(a) Fraction of solutions.

(b) Objective value.

Figure 4.4.: Distribution of evaluation metrics over datasets, cross-validation folds, and constraint-generation runs, by constraint type. Outliers are removed to improve readability.

more features since they require either all of these features to be selected or none. Further, remember that we combine *IFF* with a *Global-AT-MOST* (T1) constraint to prevent the trivial outcome of selecting all features. The more features are involved in *IFF*s, the more difficult it becomes to satisfy the global cardinality constraint. In the worst case, the set of selected features becomes empty, which is valid but has an objective value of zero.

4.4.4. Summary

We observed a relationship between the size of the search space $frac_{so}$ and the feature-set quality, i.e., objective value Q_{norm} and prediction performance R^2_{norm}. Both the fraction of selected features $frac_{se}$ and the feature-set quality tended to decrease when constraints became stronger, i.e., pruned solutions. However, the effect was non-linear, i.e., stronger constraints could still yield high feature-set quality. In particular, there may be sweet spots, i.e., high-quality feature sets that adhere to the constraints. While feature-set quality was strongly related to the fraction of solutions, the latter is costly to compute. To a lesser extent, more coarse-grained metrics like the number of constraints num_{co} and the fraction of constrained features $frac_{cf}$ were also related to feature-set quality. Further, the distribution of evaluation metrics strongly depended on the constraint type. Thus, it is difficult to make statements about the impact of constraints apart from some general trends. Finally, the optimization's objective value Q_{norm} and the prediction performance R^2_{norm} may yield different values for feature-set quality, particularly if the objective employs a simplified quality criterion like the linear function from Equation 2.1.

5. Formulating Scientific Hypotheses as Constraints – A Case Study

5.1. Overview

Scope In Chapter 4, we formally introduced constrained feature selection and evaluated the impact of randomly generated constraints in a domain-independent manner. We observed several general trends but also noted that the results depend on the constraint types. In this chapter, we conduct a domain-specific case study in materials science. The solver-based optimization approach for constrained feature selection remains the same as in Chapter 4. However, the goal and several other components of the current study's experimental design differ significantly from the previous study. In particular, we involve users, and the constraints represent hypotheses and user preferences (cf. Section 1.2.1).

Contributions Our contribution in this chapter is a case study to evaluate the impact of constraints for a concrete scenario. We use a dataset from materials science with 135 features. The dataset represents the microstructural evolution in a material specimen subjected to tensile loading. The physical processes behind the data are complex and currently not fully understood. Thus, we involve domain experts as users to formulate constraints. We evaluate domain-specific hypotheses about the data by analyzing how the corresponding constraints affect feature selection, particularly feature-set quality and the composition of feature sets. As in the study with generated constraints (cf. Chapter 4), we use univariate filter feature selection as the objective, formulate constraints with propositional logic and linear arithmetic, and find optimal feature sets with an SMT solver. Section 5.3.3 summarizes key results.

Materials We publish all our code and experimental data online (cf. Section 1.4).

Prior works The content of this chapter bases on the following prior work:

- Jakob Bach et al. "An Empirical Evaluation of Constrained Feature Selection". In: *SN Comput. Sci.* 3.6 (2022). DOI: 10.1007/s42979-022-01338-z

Chapter outline The remainder of this chapter is structured as follows: Section 5.2 outlines our experimental design, and Section 5.3 presents the experimental results.

5.2. Experimental Design

In this section, we introduce our experimental design for the case study in materials science. First, we give a brief overview (cf. Section 5.2.1). Next, we present our scenario and dataset (cf. Section 5.2.2), objective function (cf. Section 5.2.3), constraint types (cf. Section 5.2.4), evaluation metrics (cf. Section 5.2.5), and prediction models (cf. Section 5.2.6). Finally, we briefly outline our implementation (cf. Section 5.2.7).

5.2.1. Overview

In this case study, we evaluate constraints on one dataset from materials science. In particular, we analyze twelve constraint types representing hypotheses from the domain and also add three domain-independent constraint types representing user preferences. As in the study with generated constraints, we employ the objective of univariate filter feature selection. We evaluate feature-set quality and the composition of the feature sets.

5.2.2. Scenario and Dataset

We collaborate with materials scientists who provide us with a dataset and act as domain experts for formulating constraints and interpreting results. The general scenario of our study involves crystalline materials, such as most metals, on the micro-scale. These materials incorporate defect structures, i.e., dislocations, within the regular crystal structure. The line-like dislocations evolve over time and build complex 3D networks, which are mainly responsible for the permanent deformation of the material under load. Understanding the network evolution is vital in characterizing the deformation behavior.

The particular dataset in our study represents an aluminum specimen under tensile load [210]. Our domain experts created the dataset with numerical simulations using the method of discrete dislocation dynamics [221]. From a materials-science perspective, the research aims to study the evolution of properties characterizing the dislocation network. An example of such a property is the slip system, i.e., the slip plane and slip direction in which a dislocation can move. In face-centered cubic crystals, like our specimen, there are twelve slip systems. Another example is the so-called Schmid factor. It characterizes the slip plane and the slip direction that resolves the highest mechanical stress.

Dislocation lines can react with each other in several ways. Our prediction target is the density of dislocation line segments attached to so-called glissile reactions. Table 5.1 gives an overview of the features in the dataset, which are other physical quantities of the system, like dislocation density, shear stress, etc. Overall, the dataset consists of 14903 data objects and 135 features. Each data object represents the state of the material specimen at a particular location and time step. For predictions, we exclude the features for location, time, and simulation-step size, which leaves 130 features. As Table 5.1 displays, some physical quantities correspond to one feature, while others yield 17 features. In particular,

Table 5.1.: Physical quantities and features in the materials-science dataset.

Base quantity	Symbol	Slip systems	Aggregates	Features
edge_resolved_shear	$\sigma_{\text{rss}}^{\text{edge}}$	12	5	17
eps_equiv	$\varepsilon_{\text{eq}}^{\text{pl}}$	0	0	1
eps_xx	$\varepsilon_{\text{xx}}^{\text{pl}}$	0	0	1
eps_xy	$\varepsilon_{\text{xy}}^{\text{pl}}$	0	0	1
eps_xz	$\varepsilon_{\text{xz}}^{\text{pl}}$	0	0	1
eps_yy	$\varepsilon_{\text{yy}}^{\text{pl}}$	0	0	1
eps_yz	$\varepsilon_{\text{yz}}^{\text{pl}}$	0	0	1
eps_zz	$\varepsilon_{\text{zz}}^{\text{pl}}$	0	0	1
free_path_per_voxel	$\bar{l}_{\text{norm}}$	0	0	1
gamma	γ	12	5	17
gamma_abs	γ_{abs}	12	5	17
kappa1	κ_{screw}	12	5	17
kappa2	κ_{edge}	12	5	17
mean_free_path	$\bar{l}$	0	0	1
n_loops	n_{loops}	0	0	1
pos_x	x	0	0	1
pos_y	y	0	0	1
pos_z	z	0	0	1
q	q	12	5	17
rho	ρ	12	5	17
step	–	0	0	1
time	t	0	0	1
vonMises	σ_{vM}	0	0	1

some quantities in our dataset are measured separately for each of the twelve slip systems. In this case, we also compute five aggregates, i.e., the minimum, maximum, median, sum, and standard deviation over the twelve slip systems.

5.2.3. Objective Function and Optimization

As in the study with generated constraints (cf. Section 4.3.3), we use the SMT solver *Z3* [28, 50] to optimize the univariate objective from Equation 2.1. To instantiate the dependency measure $q(\cdot)$, we take the absolute values of the Pearson correlation of each feature with the target variable, rather than the mutual information used in Chapter 4. In particular, preliminary experiments showed strong linear relationships in the data. Also, the constraint types (I2) and (I3), which we introduce in the next section, assume that the dependency measure $q(\cdot)$ is bounded to $[0, 1]$, which mutual information violates.

5.2.4. Constraints

Overview This case study aims to evaluate the impact of constraints on feature selection in a concrete use case. We did not identify firm domain knowledge suited for constraints. However, we can still formulate constraints about hypothesized relationships between features. We then evaluate how the resulting feature sets and feature-set quality change compared to an unconstrained selection. Several outcomes are possible. For example, the selected features may change, but the feature-set quality remains similar, indicating an alternative explanation. Instead, feature-set quality may drop, indicating that the hypothesis behind the constraints is wrong. By formulating different sets of constraints and evaluating them independently, one can also compare different hypotheses.

Besides domain-specific constraints representing the hypotheses, we also employ domain-independent constraints expressing preferences on the resulting feature sets. As in our study with generated constraints (cf. Chapter 4), each constraint type may be formulated in multiple, logically equivalent ways. In the following, we present one formulation each.

Domain-independent constraint types In preliminary experiments, we observed three phenomena that made it difficult for domain experts to interpret feature sets. These phenomena were neither specific to our case study nor the domain in general. We use three constraint types to alleviate these phenomena:

(I1) *Global-cardinality*: Larger feature sets may be harder to interpret for domain experts. Thus, we apply a global cardinality constraint, which is the same as (T1) from the previous chapter. Here, we apply thresholds of $k \in \{5, 10\}$ features, so we can also compare how the feature-set quality changes from the smaller to the larger cardinality.

$$\text{Global-cardinality}(\{s_1, \ldots, s_n\}, k) = \sum_{j=1}^{n} s_j \leq k, \text{ with } k \in \{5, 10\} \tag{5.1}$$

(I2) *Inter-correlation*: Datasets may contain features strongly correlated with each other. Having such highly correlated features in the result may be undesirable, e.g., if they describe the same physical phenomenon, only measured or encoded differently. However, the objective function in Equation 2.1 considers features independently, thus ignoring inter-feature correlation. Thus, we employ a correlation threshold: If a pair of features has an absolute Pearson correlation of at least $\tau = 0.8$, we select at most one of these features. The correlation values $q(X_{\cdot j_1}, X_{\cdot j_2})$ for each pair of features j_1, j_2 can be precomputed and are therefore constants in the following formula, as is the user parameter τ.

$$\text{Inter-correlation}(\{s_1, \ldots, s_n\}, \tau) = \bigwedge_{\substack{(j_1, j_2) \in \{1, \ldots, n\}^2 \\ j_1 \neq j_2}} \left(\left(q(X_{\cdot j_1}, X_{\cdot j_2}) \geq \tau \right) \rightarrow \neg (s_{j_1} \wedge s_{j_2}) \right) \quad \text{with } \tau = 0.8 \tag{5.2}$$

(I3) *Quality-filter*: Datasets may contain features with a quality close to zero. These features provide little value when being selected but may increase optimization time. Thus, it makes sense to remove such features manually before optimization or to add a respective constraint. We set a quality threshold of 0.2 and exclude features with a lower quality. The feature qualities $q(X_{\cdot j}, y)$ and the threshold τ are constants.

$$\text{Quality-filter}(\{s_1, \ldots, s_n\}, \tau) = \bigwedge_{j=1}^{n} \left(\left(q(X_{\cdot j}, y) < \tau \right) \rightarrow \neg s_j \right), \text{ with } \tau = 0.2 \tag{5.3}$$

Domain-specific constraint types The following constraint types express hypotheses:

(D1) *Schmid-group*: For the crystal orientation of the considered specimen, the twelve slip systems can be divided into two non-overlapping groups based on the Schmid factor. Let $\mathbb{G}$ be this partitioning, i.e., a set of sets with $|\mathbb{G}| = 2$. We hypothesize that having features from at most one of these groups should suffice. Within the chosen group, an arbitrary number of features can be selected. To formalize this notion, let P be the set of physical quantities measured for the twelve slip systems (cf. Table 5.1). Further, let the subscript (p, g) identify a feature derived for a physical quantity p and a slip system g.

$$\text{Schmid-group}(\{s_1, \ldots, s_n\}) = \sum_{G \in \mathbb{G}} \left(\bigvee_{p \in P,\ g \in G} s_{(p,g)} \right) \leq 1 \tag{5.4}$$

(D2) *Quantity-Schmid-group*: The constraint type (D1) goes over all physical quantities. Alternatively, one can choose between the two slip-system groups for each quantity independently. With such a constraint, we hypothesize that one slip-system group may be relevant for some quantities and the other group for other quantities. This constraint type makes feature selection more flexible, i.e., the hypothesis is less strict.

$$\text{Quantity-Schmid-group}(\{s_1, \ldots, s_n\}) = \bigwedge_{p \in P} \left(\sum_{G \in \mathbb{G}} \left(\bigvee_{g \in G} s_{(p,g)} \right) \leq 1 \right) \tag{5.5}$$

(D3) *Schmid-group-representative*: Using the grouping of (D1), one can also select at most one feature from each group instead of selecting features from at most one group. This constraint type corresponds to the situation where both groups are important, but it is sufficient to pick a representative feature in each group.

$$\text{Schmid-group-representative}(\{s_1, \ldots, s_n\}) = \bigwedge_{G \in \mathbb{G}} \left(\sum_{p \in P,\ g \in G} s_{(p,g)} \leq 1 \right) \tag{5.6}$$

(D4) *Quantity-Schmid-group-representative*: We merge the ideas of (D2) and (D3): For each quantity independently, select at most one feature per slip-system group.

$$\text{Quantity-Schmid-group-representative}(\{s_1, \ldots, s_n\}) = \bigwedge_{p \in P,\ G \in \mathbb{G}} \left(\sum_{g \in G} s_{(p,g)} \leq 1 \right) \tag{5.7}$$

(D5) *Plastic-strain-tensor*: Six features in our dataset describe the plastic strain tensor, corresponding to six different directions in space. These are the quantities starting with *eps_* in Table 5.1, excluding *eps_quiv*. We hypothesize that selecting three of the six directions should be sufficient. To formalize this notion, let T be a set of indices, identifying features that describe the plastic strain tensor.

$$\text{Plastic-strain-tensor}(\{s_1, \ldots, s_n\}) = \sum_{t \in T} s_t \leq 3 \tag{5.8}$$

(D6) *Dislocation-density*: In our dataset, several features describe dislocation density aggregated over all slip systems: the five standard aggregates of *rho* as well as the quantities *free_path_per_voxel* and *mean_free_path* (cf. Table 5.1). We hypothesize that selecting at most one such feature is sufficient. To formalize this notion, let D be a set of indices, identifying features that describe dislocation density aggregated over all slip systems.

$$\text{Dislocation-density}(\{s_1, \ldots, s_n\}) = \sum_{d \in D} s_d \leq 1 \tag{5.9}$$

(D7) *Plastic-strain-rate*: For the plastic strain rate, our dataset contains features originating from two different computation methods. These quantities are named *gamma* and *gamma_abs* in Table 5.1. We hypothesize that selecting features from at most one method should suffice. To formalize this notion, let R_1 and R_2 be sets of indices, identifying features originating from the two computation methods.

$$\text{Plastic-strain-rate}(\{s_1, \ldots, s_n\}) = \left(\bigvee_{r \in R_1} s_r \right) + \left(\bigvee_{r \in R_2} s_r \right) \leq 1 \tag{5.10}$$

(D8) *Aggregate*: We have aggregated several physical quantities over the twelve slip systems (cf. Table 5.1). However, the five aggregate functions may be redundant to each other. Thus, we hypothesize that at most one kind of aggregate should suffice. To formalize this notion, let P be the set of physical quantities measured for the twelve slip systems and let A be the set of aggregate functions. Further, let subscript (p, a) identify a feature derived for a physical quantity p and an aggregate function a.

$$\text{Aggregate}(\{s_1, \ldots, s_n\}) = \sum_{a \in A} \left(\bigvee_{p \in P} s_{(p,a)} \right) \leq 1 \tag{5.11}$$

(D9) *Quantity-aggregate*: Constraint type (D8) can be refined to choose the aggregate function for each physical quantity independently. This new constraint type allows choosing different aggregate functions for different quantities.

$$\text{Quantity-aggregate}(\{s_1, \ldots, s_n\}) = \bigwedge_{p \in P} \left(\sum_{a \in A} s_{(p,a)} \leq 1 \right) \tag{5.12}$$

(D10) *Aggregate-or-original*: Aggregates describe the same physical quantities as the features they aggregate. Thus, we hypothesize that selecting original features or aggregates for each quantity should suffice. To formalize this notion, let subscript (p, l) identify a feature derived for a physical quantity p and a slip system $l \in \{1, \ldots, 12\}$.

$$\text{Aggregate-or-original}(\{s_1, \ldots, s_n\}) = \bigwedge_{p \in P} \left(\left(\bigvee_{a \in A} s_{(p,a)} \right) + \left(\bigvee_{l \in \{1,\ldots,12\}} s_{(p,l)} \right) \leq 1 \right) \tag{5.13}$$

(D11) *Mixed*: Various combinations of the previous constraint types above are possible, though some constraint types are redundant or compete with each other. For comparison, we consider the union of (D4), (D5), (D6), (D7), (D9), and (D10).

$$\begin{aligned} \text{Mixed}(\{s_1, \ldots, s_n\}) = \; & \text{Quantity-Schmid-group-representative}(\{s_1, \ldots, s_n\}) \wedge \\ & \text{Plastic-strain-tensor}(\{s_1, \ldots, s_n\}) \wedge \\ & \text{Dislocation-density}(\{s_1, \ldots, s_n\}) \wedge \\ & \text{Plastic-strain-rate}(\{s_1, \ldots, s_n\}) \wedge \\ & \text{Quantity-aggregate}(\{s_1, \ldots, s_n\}) \wedge \\ & \text{Aggregate-or-original}(\{s_1, \ldots, s_n\}) \end{aligned} \tag{5.14}$$

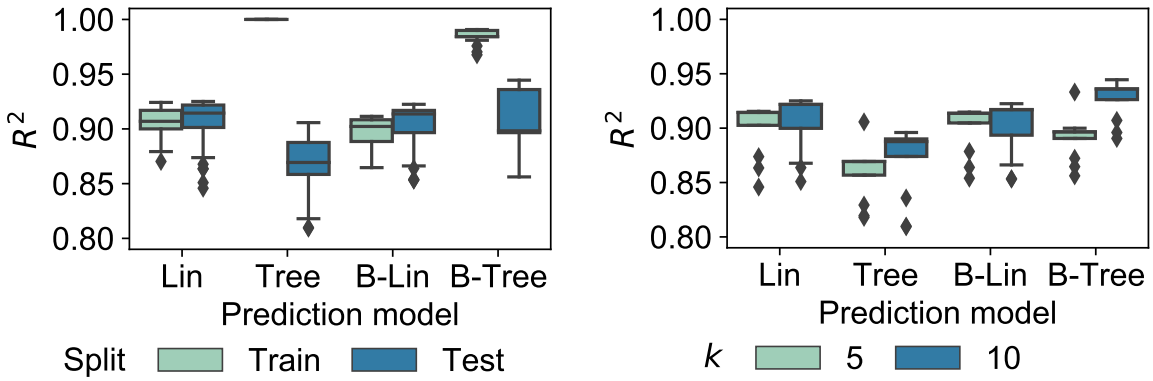
(D12) *Unconstrained*: As a reference point, we evaluate the setting without domain-specific constraints. This definition is the same as (T10) in the previous chapter.

$$\text{Unconstrained}(\{s_1, \ldots, s_n\}) = 1 \tag{5.15}$$

Combining domain-independent and domain-specific constraint types We create two sets of constraints from the domain-independent constraint types (I1), (I2), and (I3). Both sets involve all three types, but one uses the feature-set cardinality $k = 5$, and the other $k = 10$. We combine each of these two constraint sets with each of the twelve domain-specific constraint types. Thus, we get 24 constraint sets overall. In the following evaluation, we mainly mention the names of the domain-specific constraint types, but the domain-independent constraint types are always active as well.

5.2.5. Evaluation Metrics

To evaluate our hypotheses expressed as constraints, we focus on selected features and feature-set quality, i.e., objective value and prediction performance. In particular, we analyze how these outcomes differ between hypotheses. Technically, we could use the complete set of evaluation metrics from the study with generated constraints (cf. Section 4.3.5). However, we only have 24 experimental runs here, so statements on the relationship between evaluation metrics would be less generalizable.



(a) All data, by split.

(b) Only test data, by cardinality k.

Figure 5.1.: Distribution of prediction performance over constraint types, by prediction model.

5.2.6. Prediction

We use the same prediction models as in the study with generated constraints (cf. Section 4.3.4). As our current dataset has a temporal dimension, we apply a time-based 80:20 holdout split instead of cross-validation, i.e., all data objects in the training data are from earlier time steps than in the test data. However, our predictions do not go into the future. Instead, we predict the target variable from feature values at the same time step.

5.2.7. Implementation and Execution

As in the previous study (cf. Section 4.3.7), we implemented our experimental pipeline in Python 3.7, using *scikit-learn* [176] and *xgboost* [41] for machine learning and the SMT solver *Z3* [28, 50] for optimization. All code is available online (cf. Section 1.4). We organized the constrained-feature-selection functionality as a Python package.

Our experimental pipeline parallelizes over constraint types, while each of these experimental tasks runs single-threaded. We ran the pipeline on a server with 128 GB RAM and an *AMD EPYC 7551* CPU, having 32 physical cores and a base clock of 2.0 GHz. With this hardware, the parallelized pipeline run took less than one hour.

5.3. Evaluation

In this section, we evaluate our case study in materials science. In particular, we analyze feature-set quality (cf. Section 5.3.1) and selected features (cf. Section 5.3.2). Finally, we summarize key findings (cf. Section 5.3.3).

5.3.1. Feature-Set Quality

Prediction performance Figure 5.1a shows the prediction performance over all 24 experimental runs. All four prediction models perform well, even simple linear regression. The twelve domain-specific constraint types cause some but not substantial variation in prediction performance. Figure 5.1b shows that models with five features tend to perform only slightly worse than models with ten features. I.e., one can make good predictions with a small set of features, which renders the scenario suitable for feature selection.

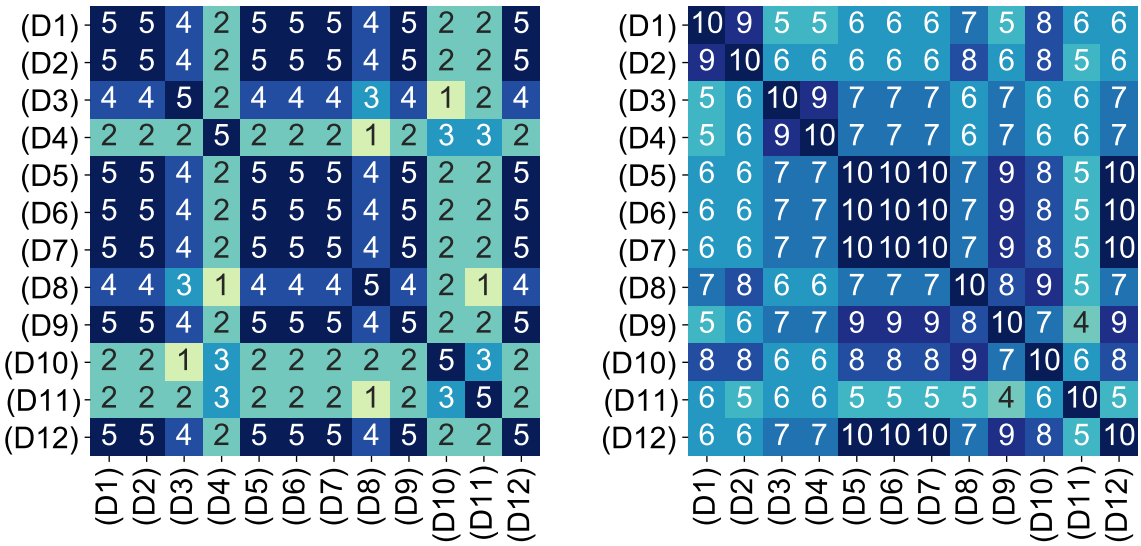
Objective value The objective value, i.e., summed univariate feature quality (cf. Section 5.2.3), is also high and has a relatively low variance over constraint types. In particular the objective value is in $[3.79, 4.04]$ for $k = 5$ features and in $[6.45, 7.48]$ for $k = 10$. The theoretical upper bound on each feature's quality is 1. In our dataset, the maximum feature quality is 0.92, and the average feature quality is 0.27. Thus, the objective values achieved are considerably higher than the expected values, i.e., $0.27 \cdot 5 = 1.35$ and $0.27 \cdot 10 = 2.7$.

The constraint type *Mixed* (D11) yields the lowest objective value for both cardinalities. This type combines six other constraint types and thus restricts the search space more than the others. Without this constraint type, the ranges of the objective value narrow down to $[3.92, 4.04]$ and $[7.03, 7.48]$. I.e., our domain-specific constraint types have relatively little impact on the objective value. In contrast, cardinality does have a noticeable impact, as can be expected from the linear objective. Nevertheless, the size of the increase indicates that the sixth to tenth selected features still have relatively high quality. This result should give way to alternative feature sets with similar quality, as we analyze later.

Comparing hypotheses As none of the analyzed constraint types significantly decrease feature-set quality, none of the underlying hypotheses seem to be invalidated. Further, the modest variation of feature-set quality between hypotheses makes it challenging to draw conclusions regarding individual hypotheses. A more detailed, domain-specific analysis of the hypotheses may be beneficial but is beyond the scope of this dissertation.

5.3.2. Selected Features

For all constraint types, the selected feature sets contain several dislocation-density-related features. This observation is expected, as the target variable quantifies reactions of dislocations. Figure 5.2 shows how many selected features are the same when comparing results with different constraint types. For $k = 5$ (cf. Figure 5.2a), several constraint types share more than half of their selected features. In particular, six constraint types yield the same feature set as the reference case, i.e., *Unconstrained* (D12). We see two reasons for such 'inactive' constraints. First, the constraints may already be consistent with the unconstrained result even if they refer to its features. For example, *Unconstrained* (D12) with $k = 5$ only selects dislocation-density features from one slip-system group, so the constraint types *Schmid-group* (D1) and *Quantity-Schmid-group* (D2) are satisfied already.



(a) Cardinality $k = 5$.

(b) Cardinality $k = 10$.

Figure 5.2.: Number of common features between resulting feature sets for different constraint types.

Second, some constraint types may not affect any of the selected features in the reference case. For example, *Plastic-strain-rate* (D7) and *Plastic-strain-tensor* (D5) refer to physical quantities that have low quality in the prediction scenario. In future work, one might apply an iterative approach to avoid inactive constraints: inspect the unconstrained result, formulate constraints, inspect the results, etc.

The picture becomes slightly more diverse for $k = 10$ (cf. Figure 5.2b). Most constraint types still share at least half the selected features. However, there are fewer cases where all features are the same since the reference case violates more constraint types now. Still, as observed in the previous section, the feature-set quality is pretty similar between the constraint types. For example, *Mixed* (D11) swaps 3/5 or 5/10 features compared to *Unconstrained* (D12). At the same time, the objective value only drops by 6.2% or 13.8%, which is a much smaller share. This observation indicates that constraints may yield alternative feature sets that differ from the reference case but still have a similar quality.

5.3.3. Summary

We observed that our domain-specific constraint types had a relatively small impact on feature-set quality. In other words, our analysis did not yield evidence against our hypotheses. However, constraints resulted in features being replaced with alternatives of similar quality, motivating a principled approach for alternative feature selection (cf. Chapter 6). Also, selecting only a few features sufficed to reach a high prediction performance. Additionally, several constraint types were already satisfied in the solution of the unconstrained reference case. Thus, we recommend an iterative approach of inspecting solutions and formulating new constraints instead of formulating all constraints upfront.

6. Finding Alternative Feature Sets

6.1. Overview

Scope Our case study in Chapter 5 showed that differently composed feature sets of similar quality may exist. However, these particular alternative feature sets were only a side-effect of employing different domain-specific constraints. Nevertheless, obtaining such alternative solutions is generally desirable for users (cf. Section 1.2.2). In this chapter, we formally introduce alternative feature selection in a domain-independent manner. This new problem definition instantiates constrained feature selection (cf. Chapter 4) with particular constraint types. Users can employ these predefined constraint types and configure them with user-friendly parameters.

Contributions Our contribution in this chapter is fivefold.

(1) We formalize alternative feature selection as an optimization problem, using 0-1 integer linear constraints to express alternative feature sets. This approach is orthogonal to the feature-selection method so that users can choose the latter according to their needs. Additionally, users may add further constraints on feature sets, e.g., to capture domain knowledge. We let users control the search for alternatives with two parameters, i.e., the number of alternatives and a dissimilarity threshold. For multiple alternatives, we consider sequential and simultaneous search.

(2) We discuss how to solve this optimization problem. To that end, we describe how to integrate different categories of conventional feature-selection methods into the optimization problem's objective function. In particular, we outline solver-based search methods for white-box and black-box optimization.

(3) We analyze the time complexity of the problem. We show $\mathcal{NP}$-hardness, even for a simple notion of feature-set quality, i.e., univariate feature qualities.

(4) We propose heuristic search methods for univariate feature qualities. We show that, under certain conditions, the optimization problem resides in the complexity class $\mathcal{APX}$, i.e., a constant-factor approximation exists.

(5) We conduct comprehensive experiments with 30 binary-classification datasets from the Penn Machine Learning Benchmarks (PMLB) [173, 186] and five feature-selection methods. We evaluate feature-set quality and runtime to investigate our search methods for alternatives and their user parameters. Section 6.4.4 summarizes key results.

Materials We publish all our code and experimental data online (cf. Section 1.4).

Prior works The content of this chapter bases on the following prior works:

- Jakob Bach. *Finding Optimal Diverse Feature Sets with Alternative Feature Selection.* arXiv:2307.11607v2 [cs.LG]. 2024. DOI: 10.48550/arXiv.2307.11607
- Jakob Bach and Klemens Böhm. "Alternative feature selection with user control". In: *Int. J. Data Sci. Anal.* (2024). DOI: 10.1007/s41060-024-00527-8

Chapter outline The remainder of this chapter is structured as follows: Section 6.2 describes and analyzes alternative feature selection. Section 6.3 outlines our experimental design. Section 6.4 presents the corresponding experimental results.

6.2. Alternative Feature Selection

In this section, we present the problem and approaches for alternative feature selection. First, we define the overall structure of the optimization problem (cf. Section 6.2.1). Second, we formalize alternatives via constraints (cf. Section 6.2.2). Third, we discuss objective functions for different feature-set quality measures and describe how to solve the resulting optimization problem (cf. Section 6.2.3). Fourth, we analyze the problem's time complexity (cf. Section 6.2.4). Fifth, we propose and analyze heuristic search methods for univariate feature qualities in the objective (cf. Section 6.2.5).

6.2.1. Optimization Problem

Alternative feature selection has two goals. First, the quality of an alternative feature set should be high, as in conventional feature selection (cf. Definition 1). Second, an alternative feature set should differ from one or more other feature set(s). There are several ways to combine these two goals in an optimization problem:

First, one can consider both goals as objectives, obtaining an unconstrained multi-objective problem. Second, one can define constraints for both, feature-set quality and being alternative, searching for feasible solutions instead of optimizing. Third, one can consider being alternative as objective and constrain feature-set quality, e.g., with a lower bound. Fourth, one can treat feature-set quality as objective and enforce alternatives with constraints.

We pursue the last formulation, i.e., optimizing feature-set quality subject to being alternative. This formulation is a special case of constrained feature selection (cf. Definition 4) and keeps the original objective of feature selection. Users need not set a threshold on feature-set quality but control how alternative the feature sets must be instead. We obtain the following optimization problem for a single alternative feature set F_s:

$$\begin{aligned} \max_{s} \quad & Q(s, X, y) \\ \text{subject to:} \quad & F_s \text{ being alternative} \end{aligned} \tag{6.1}$$

In the following, we discuss different objective functions $Q(s, X, y)$ (cf. Section 6.2.3) and suitable constraints for feature sets *being alternative* (cf. Section 6.2.2). As in the previous chapter, we also typically limit the feature-set size $|F_s|$ to a user-defined value $k \in \mathbb{N}$, which adds a further, simple constraint (cf. Equation 4.3) to the optimization problem.

6.2.2. Constraints – Defining Alternatives

In this section, we formalize alternative feature sets. First, we discuss the base case where an individual feature set is an alternative to another one (cf. Section 6.2.2.1). Second, we extend this notion to multiple alternatives, considering sequential and simultaneous search as two different problems (cf. Section 6.2.2.2). Our notion of alternatives is independent of the feature-selection method. We provide two parameters, i.e., a dissimilarity threshold τ and the number of alternatives a, allowing users to control the search for alternatives.

6.2.2.1. Single Alternative

We consider a feature set an alternative to another if it differs sufficiently. Formally, we express this notion with a set-dissimilarity measure [43, 56]. Such measures typically assess set overlap and set sizes. E.g., a well-known set-dissimilarity measure is the Jaccard distance, which is defined as follows for the (feature) sets F' and F'':

$$d_{\text{Jacc}}(F', F'') = 1 - \frac{|F' \cap F''|}{|F' \cup F''|} = 1 - \frac{|F' \cap F''|}{|F'| + |F''| - |F' \cap F''|} \tag{6.2}$$

In this dissertation, we use a dissimilarity measure based on the Dice coefficient:

$$d_{\text{Dice}}(F', F'') = 1 - \frac{2 \cdot |F' \cap F''|}{|F'| + |F''|} \tag{6.3}$$

Generally, we only have mild assumptions on the set-dissimilarity measure $d(\cdot)$. Our subsequent definitions, examples, and propositions assume symmetry, i.e., $d(F', F'') = d(F'', F')$, normalization $d(\cdot) \in [0, 1]$, and that $d(\cdot) = 1$ implies an empty intersection of the two sets. In particular, $d(\cdot)$ does not need to be a metric but can also be a semi-metric [223] like $d_{\text{Dice}}(\cdot)$. In contrast to metrics, semi-metrics may violate the triangle inequality.

We leverage the set-dissimilarity measure $d(\cdot)$ for the following definition:

Definition 5 (Single alternative)**.** *Given a symmetric set-dissimilarity measure $d(\cdot) \in [0, 1]$ with $d(\cdot) = 1$ implying no set overlap, and a dissimilarity threshold $\tau \in [0, 1]$, a feature set F' is an alternative to a feature set F'' (and vice versa) if $d(F', F'') \geq \tau$.*

The threshold τ controls how dissimilar alternative feature sets must be. A larger τ may reduce feature-set quality more, but the exact impact depends on the dataset and is unclear a priori. Also, only users can decide which drop in feature-set quality is acceptable as a trade-off for obtaining alternatives. Thus, we leave τ as a user parameter. For dissimilarity measures $d(\cdot)$ with range $[0, 1]$, like the Dice dissimilarity (cf. Equation 6.3) or Jaccard distance (cf. Equation 6.2), τ has a user-friendly interpretation: $\tau = 0$ allows identical feature sets, while $\tau = 1$ implies zero overlap. Users can adjust τ to find an acceptable quality-dissimilarity trade-off, e.g., with a binary search over τ's range.

When implementing Definition 5, the following proposition gives way to using a broad range of solvers to tackle the related optimization problem:

Proposition 7 (Linearity of constraints for alternatives)**.** *Using the Dice dissimilarity (cf. Equation 6.3), alternative feature sets (cf. Definition 5) can be expressed with 0-1 integer linear constraints.*

Proof. We re-arrange terms in the Dice dissimilarity (cf. Equation 6.3) to eliminate the quotient of feature-set sizes:

$$\begin{aligned} d_{\text{Dice}}(F', F'') = \quad & 1 - \frac{2 \cdot |F' \cap F''|}{|F'| + |F''|} \geq \tau \\ \Leftrightarrow \quad & |F' \cap F''| \leq \frac{1 - \tau}{2} \cdot (|F'| + |F''|) \end{aligned} \tag{6.4}$$

Next, we express the set sizes in terms of the feature-selection vector s:

$$\begin{aligned} |F_s| &= \sum_{j=1}^{n} s_j \\ |F_{s'} \cap F_{s''}| &= \sum_{j=1}^{n} s'_j \cdot s''_j \end{aligned} \tag{6.5}$$

Finally, we replace each product $s'_j \cdot s''_j$ with an auxiliary variable t_j, bound by additional constraints, to linearize it [155]:

$$\begin{aligned} t_j &\leq s'_j \\ t_j &\leq s''_j \\ 1 + t_j &\geq s'_j + s''_j \\ t_j &\in \{0, 1\} \end{aligned} \tag{6.6}$$

Combining Equations 6.4, 6.5, and 6.6, we obtain a set of constraints that only involve linear expressions of binary decision variables. In particular, there are only sum expressions and multiplications with constants but no products between variables. If one feature set is known, i.e., either s' or s'' is fixed, Equation 6.5 is already linear without Equation 6.6. □

Given a suitable objective function, which we discuss in Section 6.2.3, linear constraints allow using a broad range of solvers. Alternatively, one could also encode these constraints into propositional logic (SAT) [213] to enable using another category of solvers.

Table 6.1.: Number of variables and constraints for a alternatives ($a + 1$ feature sets overall) and n features.

	Sequential search		Simultaneous search
	l-th Alternative	Summed	
Decision variables s	n	$(a+1) \cdot n$	$(a+1) \cdot n$
Linearization variables t	0	0	$\frac{a \cdot (a+1) \cdot n}{2}$
Alternative constraints	l	$\frac{a \cdot (a+1)}{2}$	$\frac{a \cdot (a+1)}{2}$
Linearization constraints	0	0	$\frac{3 \cdot a \cdot (a+1) \cdot n}{2}$

If the set sizes $|F'|$ and $|F''|$ are constant, e.g., a user-defined $k \in \mathbb{N}$, Equation 6.4 implies that the threshold τ has a linear relationship to the maximum number of overlapping features $|F' \cap F''|$. This correspondence makes τ easy to interpret, so we use the Dice dissimilarity in the following. In contrast, the Jaccard distance leads to a non-linear relationship between τ and the overlap size (cf. Definition 5 with Equation 6.2):

$$\begin{aligned} d_{\text{Jacc}}(F', F'') = \quad & 1 - \frac{|F' \cap F''|}{|F'| + |F''| - |F' \cap F''|} \geq \tau \\ \Leftrightarrow \quad & |F' \cap F''| \leq \frac{1-\tau}{2-\tau} \cdot (|F'| + |F''|) \end{aligned} \tag{6.7}$$

Further, if $|F'| = |F''|$, as in our experiments, the Dice dissimilarity becomes identical to several other set-dissimilarity measures [56]. The parameter τ then directly expresses which fraction of features in one set needs to differ from the other set and vice versa:

$$d_{\text{Dice}}(F', F'') \geq \tau \Leftrightarrow |F' \cap F''| \leq (1-\tau) \cdot |F'| = (1-\tau) \cdot |F''| \tag{6.8}$$

Thus, if users are uncertain how to choose τ and $|F'|$ is reasonably small, they can try out all values of $\tau \in \{l/|F'|\}$ with $l \in \{1, \ldots, |F'|\}$. In particular, these $|F'|$ unique values of τ suffice to produce all distinct solutions that one could obtain with an arbitrary $\tau \in (0, 1]$.

6.2.2.2. Multiple Alternatives

If users desire multiple alternative feature sets rather than only one, we can determine these alternatives sequentially or simultaneously. The number of alternatives $a \in \mathbb{N}_0$ is a parameter to be set by the user. The overall number of feature sets is $a + 1$ since we deem one feature set the *original* one. Table 6.1 shows the sizes of the two search problems.

Sequential-search problem In the sequential-search problem, users obtain several alternatives iteratively, with one feature set per iteration. We constrain this new set to be an alternative to all previously found ones, which are given in a set $\mathbb{F}$:

Definition 6 (Sequential alternative)**.** *A feature set F'' is an alternative to a set of feature sets $\mathbb{F}$ (and vice versa) if F'' is a single alternative (cf. Definition 5) to each $F' \in \mathbb{F}$.*

One could also think of less strict constraints, e.g., only bounding the average dissimilarity to all previously found feature sets. However, definitions like the latter may allow some feature sets to overlap heavily or even be identical if others are very dissimilar. Thus, we require pairwise dissimilarity in Definition 6. Combining Equation 6.1 with Definition 6, we obtain the following optimization problem for each iteration of the search:

$$\begin{aligned} \max_{s} \quad & Q(s, X, y) \\ \text{subject to:} \quad & \forall F' \in \mathbb{F} : \; d(F_s, F') \geq \tau \end{aligned} \tag{6.9}$$

The full textual problem definition corresponding to Equation 6.9 is the following:

Definition 7 (Sequential-search problem for one alternative feature set)**.** *Given*

- *a dataset* $X \in \mathbb{R}^{m \times n}$ *with prediction target* $y \in Y^m$*,*
- *a set* $\mathbb{F}$ *of existing feature sets for* X*,*
- *a symmetric set-dissimilarity measure* $d(\cdot) \in [0, 1]$ *with* $d(\cdot) = 1 \rightarrow$ *no set overlap,*
- *and a dissimilarity threshold* $\tau \in [0, 1]$*,*

sequential search for one alternative feature set is the problem of making feature-selection decisions $s \in \{0, 1\}^n$ *that maximize a given notion of feature-set quality* $Q(s, X, y)$ *while making the corresponding feature set* F_s *a sequential alternative to* $\mathbb{F}$ *(cf. Definition 6).*

We solve this optimization problem repeatedly, thereby optimizing only one feature set's quality at once. $\mathbb{F} = \emptyset$ in the first iteration yields the *original* feature set, i.e., the best unconstrained feature set. Each alternative enlarges $\mathbb{F}$ and adds one constraint, but the number of decision variables remains the same. Further, we do not need to introduce linearization variables (cf. Equation 6.6) since all feature sets except one are fixed. Thus, the runtime of a solver-based sequential search should scale well with the number of alternatives. Additional runtime gains may arise if the solver keeps a state between iterations and can warm-start. Further, users do not need to define the number of alternatives a priori but can stop after each iteration once the feature-set quality is too low. As a caveat, the stepwise optimization may yield alternatives with significantly different quality.

Simultaneous-search problem In the simultaneous-search problem, users obtain multiple alternatives at once, so they need to decide on the number of alternatives a beforehand. We use pairwise dissimilarity constraints for alternatives again:

Definition 8 (Simultaneous alternatives)**.** *A set of feature sets* $\mathbb{F}$ *contains simultaneous alternatives if each feature set* $F' \in \mathbb{F}$ *is a single alternative (cf. Definition 5) to each other feature set* $F'' \in \mathbb{F}$ *with* $F' \neq F''$*.*

Combining Equation 6.1 with Definition 8, we obtain the following optimization problem for $a + 1$ feature sets, i.e., including the original feature set:

$$\begin{aligned} \max_{s^{(0)}, \ldots, s^{(a)}} \quad & \operatorname*{agg}_{l \in \{0, \ldots, a\}} Q(s^{(l)}, X, y) \\ \text{subject to:} \quad & \forall l_1, l_2 \in \{0, \ldots, a\}, \; l_1 \neq l_2 : \; d(F_{s^{(l_1)}}, F_{s^{(l_2)}}) \geq \tau \end{aligned} \tag{6.10}$$

The full textual problem definition corresponding to Equation 6.10 is the following:

Definition 9 (Simultaneous-search problem for alternative feature sets)**.** *Given*

- *a dataset* $X \in \mathbb{R}^{m \times n}$ *with prediction target* $y \in Y^m$*,*
- *the number of alternatives* $a \in \mathbb{N}_0$*,*
- *an aggregation operator* $agg(\cdot) : \mathbb{R}^{a+1} \to \mathbb{R}$ *for feature-set qualities,*
- *a symmetric set-dissimilarity measure* $d(\cdot) \in [0, 1]$ *with* $d(\cdot) = 1 \to$ *no set overlap,*
- *and a dissimilarity threshold* $\tau \in [0, 1]$*,*

simultaneous search for alternative feature sets is the problem of making feature-selection decisions $s^{(l)} \in \{0, 1\}^n$ *for* $l \in \{0, \ldots, a\}$ *that maximize a given notion of feature-set quality* $Q(s, X, y)$ *aggregated over the alternatives with* $\text{agg}_{l \in \{0,\ldots,a\}} Q(s^{(l)}, X, y)$ *while making the corresponding feature sets* $\mathbb{F} = \{F_{s^{(0)}}, \ldots, F_{s^{(a)}}\}$ *simultaneous alternatives (cf. Definition 8).*

In contrast to the sequential case (cf. Definition 7), the problem requires $a + 1$ instead of one decision vector s of length n, and a modified objective function. The operator $\text{agg}(\cdot)$ defines how to aggregate the feature-set qualities of the alternatives, which we discuss later. Runtime-wise, exact simultaneous search should scale worse with the number of alternatives than exact sequential search, as it tackles one large optimization problem instead of multiple smaller ones. Besides requiring more decision variables, we need to introduce an auxiliary variable for each feature and pair of alternatives if we want to obtain linear constraints (cf. Equation 6.6 and Table 6.1). Quality-wise, simultaneous search may have an advantage over sequential search due to optimizing alternatives globally rather than greedily. Also, the quality can be more evenly distributed over the alternatives, as opposed to the dropping quality over the course of the sequential procedure. As a downside, there are no intermediate steps where users could interrupt the search.

Sum-aggregation One straightforward way to aggregate the qualities of feature sets in the simultaneous-search objective is to sum them up, which we call *sum-aggregation*:

$$\max_{s^{(0)},\ldots,s^{(a)}} \sum_{l=0}^{a} Q(s^{(l)}, X, y) \tag{6.11}$$

While this objective fosters a high average feature-set quality, it does not guarantee that the alternatives have similar quality:

Example 1 (Sum-aggregation)**.** *Consider* $n = 6$ *features with univariate feature qualities (cf. Equation 2.1)* $q = (9, 8, 7, 3, 2, 1)$*, feature-set size* $k = 3$*, number of alternatives* $a = 2$*, the Dice dissimilarity (cf. Equation 6.3) as* $d(\cdot)$*, and dissimilarity threshold* $\tau = 0.5$*, which permits an overlap of one feature between sets here (cf. Equation 6.8). Exact sequential search (cf. Definition 7) yields the selection* $s^{(0)} = (1, 1, 1, 0, 0, 0)$*,* $s^{(1)} = (1, 0, 0, 1, 1, 0)$*, and* $s^{(2)} = (0, 1, 0, 1, 0, 1)$*, with a summed quality of* $24 + 14 + 12 = 50$*. One possible exact simultaneous-search (cf. Definition 9) solution consists of the feature sets* $s^{(0)} = (1, 1, 0, 1, 0, 0)$*,* $s^{(1)} = (1, 0, 1, 0, 1, 0)$*, and* $s^{(2)} = (0, 1, 1, 0, 0, 1)$*, with a summed quality of* $20 + 18 + 16 = 54$*. Another possible exact simultaneous-search solution is* $s^{(0)} = (1, 1, 0, 0, 0, 1)$*,* $s^{(1)} = (1, 0, 1, 0, 1, 0)$*, and* $s^{(2)} = (0, 1, 1, 1, 0, 0)$*, with a summed quality of* $18 + 18 + 18 = 54$*.*

This example yields several insights. First, exact sequential search yields worse quality than exact simultaneous search here, i.e., 50 vs. 54. Second, the feature-set qualities of the sequential solution, i.e., 24, 14, and 12, differ significantly. Third, an exact simultaneous search can admit multiple solutions whose feature-set quality is differently balanced. I.e., the solution with feature-set qualities 18, 18, and 18 is more balanced than the one with 20, 18, and 16. However, both solutions are equally optimal for sum-aggregation.

Min-aggregation To actively foster balanced feature-set qualities in simultaneous search, we propose *min-aggregation* in the objective:

$$\max_{s^{(0)},\ldots,s^{(a)}} \min_{l\in\{0,\ldots,a\}} Q(s^{(l)}, X, y) \tag{6.12}$$

This formulation maximizes the quality of the worst alternative. Thereby, it incentivizes all alternatives to have high quality and implicitly balances their quality. In the terminology of social choice theory, it uses an egalitarian rule instead of a utilitarian one [159].

Optimizing with either sum-aggregation or min-aggregation does not necessarily optimize the other. Example 1 already showed a solution optimizing sum-aggregation but not min-aggregation. In the following, we demonstrate the other direction:

Example 2 (Min-aggregation)**.** *Consider $n = 6$ features with univariate feature qualities (cf. Equation 2.1) $q = (11, 10, 6, 5, 4, 1)$, feature-set size $k = 3$, number of alternatives $a = 1$, the Dice dissimilarity (cf. Equation 6.3) as $d(\cdot)$, and dissimilarity threshold $\tau = 0.5$, which permits an overlap of one feature between sets here (cf. Equation 6.8). One simultaneous-search (cf. Definition 9) solution with min-aggregation (cf. Equation 6.12) is $s^{(0)} = (1, 1, 0, 0, 1, 0)$ and $s^{(1)} = (1, 0, 1, 1, 0, 0)$, with a summed quality of $25 + 22 = 47$. Another solution is $s^{(0)} = (1, 1, 0, 0, 0, 1)$ and $s^{(1)} = (1, 0, 1, 1, 0, 0)$, with a summed quality of $22 + 22 = 44$.*

While both solutions have the same minimum feature-set quality, only the first solution optimizes sum-aggregation. In particular, min-aggregation permits reducing the quality of feature sets as long as the latter remains above the minimum quality of all sets.

From the technical perspective, Equation 6.12 has the disadvantage of being non-linear regarding the decision variables $s^{(0)}, \ldots, s^{(a)}$. However, we can linearize it with an auxiliary variable $Q_{\min}$ and one constraint per feature set:

$$\begin{aligned} \max_{s^{(0)},\ldots,s^{(a)}} \quad & & Q_{\min} \\ \text{subject to:} \quad & \forall l \in \{0,\ldots,a\} : & Q_{\min} \leq Q(s^{(l)}, X, y) \\ & & Q_{\min} \in \mathbb{R} \end{aligned} \tag{6.13}$$

As we maximize $Q_{\min}$, this variable will implicitly assume the actual minimum value of $Q(s^{(l)}, X, y)$ with equality since the solution would not be optimal otherwise. This situation relieves us from introducing further auxiliary variables that are usually necessary when linearizing maximum or minimum expressions [155].

Further approaches for balancing quality Min-aggregation provides no control or guarantee of how much the feature-set qualities will actually differ between alternatives. One can alleviate this issue by adapting the objective or constraints. First, related work on Multi-Way Number Partitioning also uses other objectives for balancing [114, 123]. E.g., one could minimize the difference between maximum and minimum feature-set quality. Second, one could use sum-aggregation but constrain the minimum or maximum quality of sets or the difference between them. However, such constraint-based approaches introduce one or several quality-threshold parameters that are difficult to determine a priori. Third, one could treat balancing qualities as another objective besides maximizing the summed quality. One can then optimize two objectives simultaneously, filtering results for Pareto-optimal solutions or optimizing a weighted combination of the two objectives. In both cases, users may need to define an acceptable trade-off between the objectives.

6.2.3. Objective Functions – Finding Alternatives

In this section, we discuss how to find alternative feature sets. In particular, we describe how to address the different categories of feature-set quality measures from Section 2.1.2 in the optimization problem from Section 6.2.1. We consider white-box (cf. Section 6.2.3.1), black-box (cf. Section 6.2.3.2), and embedded approaches (cf. Section 6.2.3.3).

6.2.3.1. White-Box Optimization

If the structure of the alternative-feature-selection problem is sufficiently simple, one can employ a solver for white-box optimization (cf. Section 4.2.3). We already showed that our notion of alternative feature sets results in 0-1 integer linear constraints (cf. Proposition 7). We now discuss several feature-selection methods with objectives, i.e., feature-set quality functions $Q(s, X, y)$, that admit formulating a 0-1 integer linear problem.

Univariate filter feature selection For univariate filter feature selection (cf. Equation 2.1), the objective function is linear by default. Due to its simplicity, this objective admits various formalizations and search methods. Appendix A.1.1 specifies the complete 0-1 integer linear optimization problem, including the constraints for alternatives. Section 6.2.5 proposes heuristic search methods. As another encoding for solver-based search, one could formulate a weighted Max One problem [106], i.e., a special type of MaxSAT problem [15, 131]. In particular, Equation 2.1 is a sum of weighted binary variables, and a cardinality encoding [207] can turn the constraints for alternatives into SAT formulas.

Further, the monotonicity of the univariate objective enables speeding up arbitrary search methods if the user-defined feature-set sizes k and the number of alternatives a are small. In particular, replacing one selected feature with another feature of lower quality cannot increase the objective value. Sum-aggregation (cf. Equation 6.11) and min-aggregation (cf. Equation 6.12) for the simultaneous-search problem are monotonic as well. Thus, assuming $(a + 1) \cdot k < n$, it suffices to use the $(a + 1) \cdot k$ highest feature qualities when

searching for an optimal solution with $a+1$ feature sets. As the remaining feature qualities cannot improve the objective, one can drop them before optimization. We call this step *pre-selection*. While there may also be optimal solutions using the dropped features, their objective value cannot be higher. Such solutions may arise in case of multiple identical qualities or for min-aggregation. Also, the optimal solution may not contain all pre-selected features, i.e., pre-selection over-approximates the set of finally selected features.

Post-hoc feature importance Technically, one can also insert the values of post-hoc feature-importance scores (cf. Section 2.1.2.4) as univariate feature qualities $q(X_{\cdot j}, y)$ into Equation 2.1. However, such post-hoc importance scores typically evaluate the quality of each feature in the presence of the dataset's other features, i.e., the scores are not truly univariate. Thus, treating pre-computed post-hoc importance scores as univariate feature qualities in the optimization objective is a simplification and may not faithfully represent the actual feature qualities in a particular subset of selected features [63].

FCBF While the original FCBF (cf. Section 2.1.2.1) is an algorithm, we formulate a constrained optimization problem to enable a solver-based optimization for alternatives:

$$\begin{aligned} \max_{s} \quad & Q_{\text{FCBF}}(s, X, y) = \sum_{j=1}^{n} q(X_{\cdot j}, y) \cdot s_j \\ \text{subject to:} \quad & \forall j_1, j_2 \in \{1, \ldots, n\},\ j_1 \neq j_2,\ (*) : s_{j_1} + s_{j_2} \leq 1 \\ \text{with } (*): \quad & q(X_{\cdot j_1}, y) \leq q(X_{\cdot j_2}, X_{\cdot j_1}) \end{aligned} \tag{6.14}$$

We drop the original FCBF's threshold on feature-target correlation and maximize the latter instead, as in the univariate-filter case. Further, we keep FCBF's constraints on feature-feature correlation. In particular, we prevent the simultaneous selection of two features if the correlation between them is at least as high as one of the features' correlation to the target. Since the condition $q(X_{\cdot j_1}, y) \leq q(X_{\cdot j_2}, X_{\cdot j_1})$ in Equation 6.14 does not depend on the decision variables s, one can check whether it holds before optimization and add the corresponding linear constraint $s_{j_1} + s_{j_2} \leq 1$ only for feature pairs where it is needed.

mRMR For user-defined feature-set size $k = \sum_{j=1}^{n} s_j$, mRMR's objective (cf. Equation 2.2) yields a quadratic-programming problem [167, 185]. Replacing each product term $s_{j_1} \cdot s_{j_2}$ according to Equation 6.6 makes the problem linear. However, there is a more efficient linearization [164, 165], which we further adapt as follows:

$$\begin{aligned} \max_{s} \quad & Q_{\text{mRMR}}(s, X, y) = \frac{\sum_{j=1}^{n} q(X_{\cdot j}, y) \cdot s_j}{k} - \frac{\sum_{j=1}^{n} z_j}{k \cdot (k-1)} \\ \text{subject to:} \quad \forall j_1 : \quad & A_{j_1} := \sum_{j_2 \neq j_1} q(X_{\cdot j_1}, X_{\cdot j_2}) \cdot s_{j_2} \\ \forall j : \quad & z_j \geq M \cdot (s_j - 1) + A_j \\ \forall j : \quad & z_j \in \mathbb{R}_{\geq 0} \\ \text{with indices:} \quad & j, j_1, j_2 \in \{1, \ldots, n\} \end{aligned} \tag{6.15}$$

Here, A_{j_1} is the sum of all redundancy terms related to the feature with index j_1, i.e., the summed dependency value between this feature and all other selected features. Thus, one can linearize with one real-valued auxiliary variable z_j for each feature instead of one new binary variable for each pair of features. A_j does not need a separate variable but can be directly inserted in the subsequent constraint, so we wrote ':=' instead of '='. Since redundancy should be minimized, z_j assumes the value of A_j with equality if the feature with index j is selected ($s_j = 1$) and is zero otherwise ($s_j = 0$). To this end, M is a large positive value that deactivates the constraint $z_j \geq A_j$ if $s_j = 0$.

Since Equation 6.15 assumes the feature-set size $k \in \mathbb{N}$ to be user-defined before optimization, it requires fewer auxiliary variables and constraints than the more general formulation in [164, 165]. Additionally, in accordance with [167], we assign a value of zero to the self-redundancy terms $q(X_{\cdot j}, X_{\cdot j})$, effectively excluding them from the objective function. Thus, the redundancy term uses $k \cdot (k-1)$ instead of k^2 for averaging.

6.2.3.2. Black-Box Optimization

Overview If feature-set quality does not have an expression suitable for white-box optimization, one may treat it as a black-box function when searching for alternatives. This situation applies to wrapper feature-selection methods (cf. Section 2.1.2.2), which use prediction models to assess feature-set quality. One can optimize such black-box functions with search heuristics that systematically iterate over candidate feature sets. However, search heuristics often assume an unconstrained search space and may propose candidate feature sets that are not alternative enough. We see different ways to address this issue:

First, one may enumerate all possible feature sets that are alternative enough. For example, one can iterate over all possible feature sets and remove the invalid ones or use a solver to enumerate valid alternatives directly. Both approaches are usually very inefficient due to the vast number of potential alternatives. Sampling only some alternatives may improve efficiency but affect quality. Further, uniform sampling from a constrained space is a computationally hard problem [58]. Second, one can phrase alternative feature selection as a multi-objective problem, so there are no hard constraints anymore and one could apply a standard multi-objective black-box search method. However, as Section 6.2.1 explains, we pursue a single-objective formulation with constraints. Third, one can adapt an existing search heuristic to consider constraints. One idea is to prevent the search from producing invalid feature sets or at least make the latter less likely, e.g., with a penalty in the objective function. Another idea is to 'repair' feature sets in the search that violate constraints, e.g., replacing them with the most similar valid feature sets. Such solver-assisted search approaches are common in search methods for software feature models [79, 89, 222], and our following method for wrapper feature selection falls into this category as well.

Greedy Wrapper For wrapper feature selection in our experiments, we propose a novel hill-climbing procedure, displayed in Algorithm 6. Unlike standard hill climbing for feature selection [112], our procedure observes constraints. First, the algorithm uses a solver to

Algorithm 6: *Greedy Wrapper* for alternative feature selection.

Input: Dataset $X \in \mathbb{R}^{m \times n}$,
Prediction target $y \in Y^m$,
Quality function $Q(S, X, y)$ for sets of feature sets,
Set C of constraints for alternatives,
Maximum number of iterations $max\_iters \in \mathbb{N}$
Output: Set of feature-selection decision vectors $S = \{s^{(0)}, \ldots, s^{(a)}\}$

```
1  S^opt ← solve(C)                                          // Initial alternatives
2  iters ← 1                                  // Number of iterations = solver calls
3  if S^opt = ∅ then return ∅                          // No valid alternatives exist
4  j_1 ← 1                                    // Indices of features to be swapped
5  j_2 ← j_1 + 1
6  while iters < max_iters and j_1 < n do
7      S ← solve(Equation 6.17)                                        // Try swap
8      iters ← iters + 1
9      if S ≠ ∅ and Q(S, X, y) > Q(S^opt, X, y) then          // Swap if improved
10         S^opt ← S
11         j_1 ← 1                                    // Reset swap-feature indices
12         j_2 ← j_1 + 1
13     else if j_2 < n then                  // Try next swap; advance one index
14         j_2 ← j_2 + 1
15     else                                // Try next swap; advance both indices
16         j_1 ← j_1 + 1
17         j_2 ← j_1 + 1
18 return S^opt
```

find one solution that is alternative enough for the set of constraints C (Line 1) and stores it as the currently best solution S^{opt}. Thus, the algorithm has a valid starting point and can always return a solution unless there are no valid solutions at all (Line 3). Note that the solution is not only one feature-selection decision vector s but a set S of them, to enable simultaneous search. For sequential search, $|S| = 1$ and $a = 0$. Also, we adapt the notion of feature-set quality $Q(S, X, y)$ in this algorithm to support a set of feature sets, encompassing the aggregation operator $\text{agg}(\cdot)$ for simultaneous search (cf. Definition 9).

Next, the algorithm tries 'swapping' two features, i.e., selecting them if they were deselected or deselecting them if they were selected (Line 7). The corresponding swap indices j_1 and j_2 start at 1 and 2, respectively (Lines 4–5). For simultaneous search, we swap the affected features in each alternative. As the swap may violate constraints, the algorithm calls the solver to find the solution S that is closest to the currently best one S^{opt} while satisfying the swap constraints and the constraints for alternatives C. To this end, we measure the similarity between feature-selection decisions with the Hamming distance [43], i.e.,

how many values of decision variables differ between S and S^{opt}. Overall, we define the corresponding maximization problem for Line 7 of Algorithm 6 as follows:

$$\begin{aligned} \max_{s^{(0)},\ldots,s^{(a)}} \quad & \text{sim}(S, S^{\text{opt}}) = \sum_{l=0}^{a} \sum_{j=1}^{n} \left(s_j^{(l)} \leftrightarrow s_j^{(\text{opt},\, l)} \right) \\ \text{subject to:} \quad & C \\ & \forall l \in \{0, \ldots, a\} : \quad s_{j_1}^{(l)} \leftrightarrow \neg s_{j_1}^{(\text{opt},\, l)} \\ & \forall l \in \{0, \ldots, a\} : \quad s_{j_2}^{(l)} \leftrightarrow \neg s_{j_2}^{(\text{opt},\, l)} \end{aligned} \tag{6.16}$$

The values of $s^{(\text{opt},\, l)}$ as well as j_1 and j_2 in this problem are fixed based on Algorithm 6, while $s^{(l)}$ remains variable. Thus, we obtain a 0-1 integer linear program:

$$\begin{aligned} \max_{s^{(0)},\ldots,s^{(a)}} \quad & \text{sim}(S, S^{\text{opt}}) = \sum_{l=0}^{a} \left(\sum_{\substack{j \in \{1,\ldots,n\} \\ s_j^{(\text{opt},\, l)} = 1}} s_j^{(l)} + \sum_{\substack{j \in \{1,\ldots,n\} \\ s_j^{(\text{opt},\, l)} = 0}} \left(1 - s_j^{(l)} \right) \right) \\ \text{subject to:} \quad & C \\ & \forall l \in \{0, \ldots, a\} : \quad s_{j_1}^{(l)} = 1 - s_{j_1}^{(\text{opt},\, l)} \\ & \forall l \in \{0, \ldots, a\} : \quad s_{j_2}^{(l)} = 1 - s_{j_2}^{(\text{opt},\, l)} \end{aligned} \tag{6.17}$$

If a solution S for Equation 6.17 exists and its quality $Q(S, X, y)$ improves upon the currently best solution S^{opt}, the algorithm proceeds with the new solution, attempting again to swap the first and second features (Lines 10–12). Otherwise, it tries to swap another pair of features (Lines 13–17). Specifically, we assess only one solution per swap instead of exhaustively enumerating and evaluating all valid solutions involving the swap.

The algorithm terminates if it reaches a local optimum, i.e., no swap leads to an improvement, or a fixed number of iterations *max_iters* (Line 6). We define the iteration count as the number of solver calls, i.e., attempts to generate valid alternatives. This number also bounds the number of prediction models trained. However, we only train a model for valid solutions (Line 9), and not all solver invocations may yield one.

6.2.3.3. Embedding Alternatives

If feature selection is embedded into a prediction model (cf. Section 2.1.2.3), there is no general approach for finding alternative feature sets. Instead, one would also need to embed the search for alternatives into model training. Thus, we leave the formulation of specific approaches open for future work. For example, one could adapt the training of decision trees to not split on a feature if the resulting feature set of the tree was too similar to a given feature set. As another example, there are various formal encodings of prediction models, e.g., as SAT formulas [160, 193, 227], where 'training' already uses a solver. In such representations, one may directly add constraints for alternatives.

6.2.4. Time Complexity

In this section, we analyze the time complexity of alternative feature selection. In particular, we study the scalability regarding the number of features $n \in \mathbb{N}$, also considering the feature-set size $k \in \mathbb{N}$ and the number of alternatives $a \in \mathbb{N}_0$. Section 6.2.4.1 discusses exhaustive search for arbitrary feature-selection methods, while Section 6.2.4.2 examines the univariate objective (cf. Equation 2.1) in detail. Section 6.2.4.3 summarizes key results.

6.2.4.1. Exhaustive Search for Arbitrary Feature-Selection Methods

An exhaustive search over the entire search space is a simple but inefficient approach to find alternatives, which provides an upper bound for the time complexity.

Sequential search Sequential search for alternatives (cf. Definition 7) optimizes feature sets one at a time, like conventional feature selection (cf. Definition 1). The constraints for alternatives put an extra cost on each solution candidate. Constraint checking involves iterating over all existing feature sets and features (cf. Equation A.1), which entails a cost of $O(a \cdot n)$ for each alternative and $O((a+1)^2 \cdot n)$ for the whole search with a alternatives. Combined with the search cost from Proposition 2, we obtain the following proposition:

Proposition 8 (Complexity of exhaustive sequential search)**.** *Exhaustive sequential search (cf. Definition 7) for $a \in \mathbb{N}_0$ alternative feature sets of size $k \in \mathbb{N}$ from n features has a time complexity of $O((a+1)^2 \cdot n^{k+1})$ without the cost of evaluating the objective.*

Thus, the runtime remains polynomial in n if k is a small constant $k \in O(1)$, which places the problem in the parameterized complexity class $\mathcal{XP}$, like conventional feature selection with a cardinality constraint (cf. Proposition 3). Due to the fixed k, only choosing $a \leq \binom{n}{k} \in O(n^k)$ admits valid alternatives and therefore does a not affect polynomiality.

Simultaneous search The simultaneous-search problem (cf. Definition 9) enlarges the search space since it optimizes $a+1$ feature sets at once. Thus, an exhaustive search over size-k feature sets iterates over $O((n^k)^{a+1}) = O(n^{k\cdot(a+1)})$ solution candidates. Including the cost of constraint checking, we arrive at the following proposition:

Proposition 9 (Complexity of exhaustive simultaneous search)**.** *Exhaustive simultaneous search (cf. Definition 9) for $a \in \mathbb{N}_0$ alternative feature sets of size $k \in \mathbb{N}$ from n features has a time complexity of $O((a+1)^2 \cdot n^{k\cdot(a+1)+1})$ without the cost of evaluating the objective.*

The scalability with n is worse than for exhaustive sequential search since the number of alternatives appears in the exponent now. The time complexity only remains polynomial in n if a and k are small and independent from n, i.e., $a \cdot k \in O(1)$:

Proposition 10 (Parameterized complexity of simultaneous-search problem)**.** *The simultaneous-search problem (cf. Definition 9) for $a \in \mathbb{N}$ alternative feature sets of size $k \in \mathbb{N}$ from n features resides in the parameterized complexity class $\mathcal{XP}$ for the parameter $a \cdot k$.*

6.2.4.2. Univariate Feature Qualities

Motivation While the assumption $a \cdot k \in O(1)$ leads to polynomial runtime regarding n for arbitrary feature-selection methods, the optimization problem can still be hard in general. We already established that various feature-selection methods allow us to phrase alternative feature selection as a 0-1 integer linear program (cf. Section 6.2.3.1). Integer Programming is $\mathcal{NP}$-complete in general, even for binary decision variables [68, 103]. However, alternative feature selection could still be easier since it only uses particular constraint types. Vice versa, Integer Programming assumes a particular problem encoding, i.e., the problem's input size contains each constraint. If we instead define our input size as the total encoding length of the objective function plus parameters a, k, and τ, the problem could be harder. In particular, increasing the number of alternatives a would increase the encoding length logarithmically but the cost of constraint checking quadratically.

Scenario In the following, we derive complexity results for *univariate feature qualities* (cf. Equation 2.1 and Appendix A.1.1). This feature-selection method arguably has the simplest objective function, where the quality of a feature set equals the sum of its constituent features' individual qualities. This simplicity eases the transformation from well-known $\mathcal{NP}$-hard problems. The feature qualities $q(X_{\cdot j}, y)$ can be pre-computed and are constants during optimization, so their computation does not affect the complexity.

Min-aggregation with complete partitioning To obtain our first $\mathcal{NP}$-hardness result, we start with three assumptions, which we will drop later: First, we use a dissimilarity threshold of $\tau = 1$, i.e., no overlap of feature sets. Second, all features must be part of one set. Third, we analyze the simultaneous-search problem (cf. Definition 9) with min-aggregation (cf. Equation 6.12). We call the combination of the first two assumptions, which implies $n = (a + 1) \cdot k$ if all sets have size k, a *complete partitioning*. This scenario differs from $a \cdot k \in O(1)$, which yielded polynomial runtime in Proposition 10.

Our complete-partitioning scenario is a variant of a problem called Multi-Way Number Partitioning [114] or Multiprocessor Scheduling [68]: Partition a multiset of n numbers into a fixed number a of subsets such that all subset sums are as equal as possible. One application is assigning tasks with different lengths to processors such that the load is balanced. Maximizing the minimum subset sum is one possible objective [114, 123]. This objective corresponds to min-aggregation (cf. Equation 6.12) for the simultaneous-search problem (cf. Definition 9). Since Multiprocessor Scheduling is $\mathcal{NP}$-complete, even for just two partitions [68], our problem is $\mathcal{NP}$-complete as well:

Proposition 11 (Complexity of simultaneous-search problem with min-aggregation, complete partitioning, and unconstrained feature-set size)**.** *Assuming univariate feature qualities (cf. Equation 2.1), a dissimilarity threshold $\tau = 1$, unconstrained feature-set sizes, and all n features have to be selected, the simultaneous-search problem (cf. Definition 9) for alternative feature sets with min-aggregation (cf. Equation 6.12) is $\mathcal{NP}$-complete.*

We can also drop assumptions on the problem definition while retaining $\mathcal{NP}$-hardness since the following more general problem still contains the previous special case:

Proposition 12 (Complexity of simultaneous-search problem with min-aggregation)**.** *The simultaneous-search problem (cf. Definition 9) for alternative feature sets with min-aggregation (cf. Equation 6.12) is $\mathcal{NP}$-hard.*

While Proposition 11 allowed arbitrary set sizes, the hardness result remains valid for fixed or bounded set sizes, known as Balanced Number Partitioning [149, 231] or K-Partitioning [14, 124] in the literature:

Proposition 13 (Complexity of simultaneous-search problem with min-aggregation, complete partitioning, and constrained feature-set size)**.** *Assuming univariate feature qualities (cf. Equation 2.1), a dissimilarity threshold $\tau = 1$, desired feature-set size $k \in \mathbb{N}$, and all n features have to be selected, the simultaneous-search problem (cf. Definition 9) for alternative feature sets with min-aggregation (cf. Equation 6.12) is $\mathcal{NP}$-complete.*

Min-aggregation with incomplete partitioning We now allow that some features are not part of any feature set while keeping the assumption of no feature-set overlap. The problem of finding such an *incomplete partitioning* is still $\mathcal{NP}$-complete in general (cf. Appendix A.1.2.1 for the proof):

Proposition 14 (Complexity of simultaneous-search problem with min-aggregation, incomplete partitioning, and constrained feature-set size)**.** *Assuming univariate feature qualities (cf. Equation 2.1), a dissimilarity threshold $\tau = 1$, desired feature-set size $k \in \mathbb{N}$, and* not *all n features have to be selected, the simultaneous-search problem (cf. Definition 9) for alternative feature sets with min-aggregation (cf. Equation 6.12) is $\mathcal{NP}$-complete.*

Min-aggregation with overlapping feature sets The problem with $\tau < 1$, i.e., allowing set overlap, is also $\mathcal{NP}$-hard in general (cf. Appendix A.1.2.2 for the proof):

Proposition 15 (Complexity of simultaneous-search problem with min-aggregation, $\tau < 1$, and constrained feature-set size)**.** *Assuming univariate feature qualities (cf. Equation 2.1), a dissimilarity threshold $\tau < 1$, and desired feature-set size $k \in \mathbb{N}$, the simultaneous-search problem (cf. Definition 9) for alternative feature sets with min-aggregation (cf. Equation 6.12) is $\mathcal{NP}$-hard.*

Sum-aggregation In contrast to the previous $\mathcal{NP}$-hardness results for min-aggregation, $\tau = 1$ admits polynomial-time solutions for sum-aggregation (cf. Equation 6.11) and sequential search (cf. Appendix A.1.2.3 for the proof):

Proposition 16 (Complexity of search problems with sum-aggregation and $\tau = 1$)**.** *Assuming univariate feature qualities (cf. Equation 2.1) and a dissimilarity threshold $\tau = 1$, the problems for (1) sequential search (cf. Definition 7) and (2) simultaneous search with sum-aggregation (cf. Definition 9 and Equation 6.11) have a time complexity of $O(n \cdot \log n)$.*

This feasibility result applies to an arbitrary number of alternatives a and arbitrary feature-set sizes. The key reason for polynomial runtime is that sum-aggregation does not require balancing the feature sets' qualities. Thus, $\tau = 1$ admits many solutions with the same summed feature-set quality. While at least one of these solutions also optimizes the objective with min-aggregation, most do not, so the latter problem can be harder.

6.2.4.3. Summary

We showed that the simultaneous-search problem for alternative feature sets is $\mathcal{NP}$-hard in general (cf. Proposition 12). We also placed it in the parameterized complexity class $\mathcal{XP}$ (cf. Proposition 10), having a and k as the parameters that drive the hardness of the problem. For univariate feature qualities and min-aggregation, we obtained more specific $\mathcal{NP}$-hardness results for (1) complete partitioning, i.e., $\tau = 1$ and $(a+1) \cdot k = n$ (cf. Proposition 13), (2) incomplete partitioning, i.e., $(a + 1) \cdot k < n$ (cf. Proposition 14) and (3) feature-set overlap, i.e., $\tau < 1$ (cf. Proposition 15). In contrast, we also inferred polynomial runtime for $\tau = 1$ with sum-aggregation or sequential search (cf. Proposition 16).

6.2.5. Heuristic Search for Univariate Feature Qualities

In this section, we propose heuristic search methods for alternative feature selection with univariate feature qualities (cf. Equation 2.1 and Appendix A.1.1) and the Dice dissimilarity (cf. Equation 6.3) as $d(\cdot)$. These heuristics complement the solver-based search discussed in Section 6.2.3.1. In particular, we describe *Greedy Replacement* (cf. Section 6.2.5.1), which is a sequential search method, and *Greedy Balancing* (cf. Section 6.2.5.2), which is a simultaneous search method. The proposed heuristics should be faster than exact optimization at the expense of lower feature-set quality.

6.2.5.1. Greedy Replacement

Algorithm Algorithm 7 outlines *Greedy Replacement*, which conducts a sequential search for alternatives. It retains a fixed subset of features in each alternative while iteratively replacing the remaining ones. We start by sorting the features decreasingly based on their qualities q_j (Line 1). For a fixed feature-set size k, a dissimilarity threshold τ, and using the Dice dissimilarity (cf. Equation 6.3), one subset with $\lfloor (1 - \tau) \cdot k \rfloor$ features can be contained in all alternatives without violating the dissimilarity threshold (cf. Equation 6.8). Thus, our algorithm indeed selects the $\lfloor (1-\tau) \cdot k \rfloor$ features with highest quality in each alternative $s^{(\cdot)}$ (Lines 2–7). We fill the remaining spots in the sets by iterating over the alternatives and remaining features (Lines 8–15). For each alternative, we select the $\lceil \tau \cdot k \rceil$ highest-quality features not used in any prior alternative, thereby satisfying the dissimilarity threshold. We continue this procedure until we reach the desired number of alternatives a or until there are not enough unused features to form further alternatives (Line 9).

Algorithm 7: *Greedy Replacement* for alternative feature selection.

Input: Univariate feature qualities $q \in \mathbb{R}^n$,
Feature-set size $k \in \mathbb{N}$,
Number of alternatives $a \in \mathbb{N}_0$,
Dissimilarity threshold $\tau \in (0, 1]$
Output: List of feature-selection decision vectors $s^{(\cdot)}$

```
1  indices ← sort_indices(q, order=descending)          // Order by qualities
2  s ← {0}^n                                  // Initial selection for all alternatives
3  feature_position ← 1                       // Index of index of current feature
4  while feature_position ≤ ⌊(1 − τ) · k⌋ do
5      j ← indices[feature_position]                    // Index feature by quality
6      s_j ← 1
7      feature_position ← feature_position + 1
8  l ← 0                                          // Number of current alternative
9  while l ≤ a and l ≤ (n−k)/⌈τ·k⌉ do
10     s^(l) ← s                                  // Select top ⌊(1 − τ) · k⌋ features
11     for _ ← 1 to ⌈τ · k⌉ do                   // Select remaining ⌈τ · k⌉ features
12         j ← indices[feature_position]
13         s_j^(l) ← 1
14         feature_position ← feature_position + 1
15     l ← l + 1
16 return s^(0), ..., s^(l)
```

Example 3 (Algorithm of *Greedy Replacement*)**.** *With $n = 10$ features, feature-set size $k = 5$, and $\tau = 0.4$, each feature set must differ by $\lceil \tau \cdot k \rceil = 2$ features from the other feature sets. The original feature set $s^{(0)}$ consists of the top $k = 5$ features regarding quality q_j. The first alternative $s^{(1)}$ consists of the top $\lfloor (1 - \tau) \cdot k \rfloor = 3$ features plus the sixth- and seventh-best feature. The second alternative $s^{(2)}$ consists of the top three features plus the eighth- and ninth-best one. The algorithm cannot continue beyond $l = 2$ since there are not enough unused features to form further alternatives in the same manner.*

In general, the l-th alternative consists of the top $\lfloor (1 - \tau) \cdot k \rfloor$ features plus the features $k + (l - 1) \cdot \lceil \tau \cdot k \rceil + 1$ to $k + l \cdot \lceil \tau \cdot k \rceil$ in descending quality order.

Time complexity Sorting n feature qualities (Line 1) has a time complexity of $O(n \cdot \log n)$. Next, the algorithm iterates over the features and processes each feature at most once. In particular, after selecting a feature in an alternative, *feature_position* increases by 1. The maximum value of this variable depends on a and k (Line 9) but cannot exceed the total number of features n. For each *feature_position*, the algorithm conducts a constant number of update operations (Lines 11–14). Assuming each array access takes $O(1)$, the total update cost is $O(n)$. Further, each alternative $s^{(l)}$ gets initialized as the selection s of

the top $\lfloor (1-\tau) \cdot k \rfloor$ features (Line 10), which the algorithm determines once before the main loop (Lines 2–7). Initializing a arrays costs $O(a \cdot n)$. Since the algorithm can only yield $a < n$ alternatives, the overall time complexity is $O(n^2)$, i.e., polynomial in n.

Quality While not optimizing exactly, *Greedy Replacement* still offers an approximation guarantee relative to exact search methods:

Proposition 17 (Approximation quality of *Greedy Replacement*)**.** *Assume non-negative univariate feature qualities of n features (cf. Equation 2.1), $a \in \mathbb{N}_0$ alternatives, the Dice dissimilarity (cf. Equation 6.3) as $d(\cdot)$, a dissimilarity threshold $\tau \in (0, 1]$, desired feature-set size $k \in \mathbb{N}$, and $k + a \cdot \lceil \tau \cdot k \rceil \leq n$. Under these conditions,* Greedy Replacement *reaches at least a fraction of $\frac{\lfloor (1-\tau) \cdot k \rfloor}{k}$ of the optimal objective values of the optimization problems for (1) sequential search (cf. Definition 7), (2) simultaneous search with sum-aggregation (cf. Definition 9 and Equation 6.11), and (3) simultaneous search with min-aggregation (cf. Definition 9 and Equation 6.12).*

Proof. For univariate feature qualities, the quality of a feature set is the sum of the qualities of the contained features. *Greedy Replacement* includes the $\lfloor (1-\tau) \cdot k \rfloor$ highest-quality features in each alternative of size k, while the remaining $\lceil \tau \cdot k \rceil$ features may have an arbitrary quality. In comparison, the optimal original, i.e., unconstrained, feature set of size k contains the top k features, which are the union of the top $\lfloor (1-\tau) \cdot k \rfloor$ features and the next-best $\lceil \tau \cdot k \rceil$ features. Due to quality sorting, each of the next-best $\lceil \tau \cdot k \rceil$ features has at most the quality of each of the top $\lfloor (1-\tau) \cdot k \rfloor$ features, i.e., contributes the same or less to the summed quality of the feature set. Hence, assuming non-negative qualities, each alternative yielded by *Greedy Replacement* has at least a quality of $\lfloor (1-\tau) \cdot k \rfloor / k$ relative to the optimal original feature set of size k since the $\lfloor (1-\tau) \cdot k \rfloor$ highest-quality features are part of both feature sets. Next, the optimal original feature set of size k upper-bounds the quality of any feature set of size k. Consequently, alternative feature sets found by exact sequential or simultaneous search can also not be better. Thus, the quality bound of the heuristic solution relative to the exact solution also applies to the minimum and sum of qualities over multiple alternative feature sets. □

In particular, *Greedy Replacement* yields a constant-factor approximation for the three optimization problems mentioned in Proposition 17. The condition $k + a \cdot \lceil \tau \cdot k \rceil \leq n$ describes scenarios where *Greedy Replacement* can yield all desired alternatives, i.e., does not run out of unused features. As the heuristic has polynomial runtime, alternative feature selection lies in the complexity class $\mathcal{APX}$ [107] under the specified conditions:

Proposition 18 (Approximation complexity of alternative feature selection)**.** *Assume non-negative univariate feature qualities of n features (cf. Equation 2.1), $a \in \mathbb{N}_0$ alternatives, the Dice dissimilarity (cf. Equation 6.3) as $d(\cdot)$, a dissimilarity threshold $\tau \in [0, 1]$, desired feature-set size $k \in \mathbb{N}$, and $k + a \cdot \lceil \tau \cdot k \rceil \leq n$. Under these conditions, the optimization problems for (1) sequential search (cf. Definition 7), (2) simultaneous search with sum-aggregation (cf. Definition 9 and Equation 6.11), and (3) simultaneous search with min-aggregation (cf. Definition 9 and Equation 6.12) reside in the complexity class $\mathcal{APX}$.*

For $\tau = 1$, *Greedy Replacement* even yields the same objective value as exact sequential search and exact simultaneous search with sum-aggregation since it becomes identical to a procedure we outlined in our complexity analysis earlier (cf. Proposition 16 and Appendix A.1.2.3). In contrast, for arbitrary τ, the following example shows that the heuristic can be worse than an exact sequential search for as few as $a = 2$ alternatives:

Example 4 (Quality of *Greedy Replacement* vs. exact search)**.** *Consider $n = 6$ features with univariate feature qualities $q = (9, 8, 7, 3, 2, 1)$, feature-set size $k = 2$, number of alternatives $a = 2$, the Dice dissimilarity (cf. Equation 6.3) as $d(\cdot)$, and dissimilarity threshold $\tau = 0.5$, which permits an overlap of one feature between sets here. Exact sequential search and exact simultaneous search, for min- and sum-aggregation, yield the selection $s^{(0)} = (1, 1, 0, 0, 0, 0)$, $s^{(1)} = (1, 0, 1, 0, 0, 0)$, and $s^{(2)} = (0, 1, 1, 0, 0, 0)$, with a summed quality of $17 + 16 + 15 = 48$.* Greedy Replacement *yields the selection $s^{(0)} = (1, 1, 0, 0, 0, 0)$, $s^{(1)} = (1, 0, 1, 0, 0, 0)$, and $s^{(2)} = (1, 0, 0, 1, 0, 0)$, with a summed quality of $17 + 16 + 12 = 45$.*

While all searches yield the same $s^{(0)}$ and $s^{(1)}$, the quality of $s^{(2)}$ is lower for the heuristic (12 vs. 15). In particular, by always selecting all the top $\lfloor (1 - \tau) \cdot k \rfloor$ features, the heuristic misses feature sets not involving all of them. For min-aggregation in simultaneous search, even $a = 1$ alternative suffices for the heuristic being potentially worse than exact search:

Example 5 (Quality of *Greedy Replacement* vs. min-aggregation)**.** *Consider $n = 6$ features with univariate feature qualities $q = (9, 8, 7, 3, 2, 1)$, feature-set size $k = 3$, number of alternatives $a = 1$, the Dice dissimilarity (cf. Equation 6.3) as $d(\cdot)$, and dissimilarity threshold $\tau = 0.5$, which permits an overlap of one feature between sets here. Exact simultaneous search with min-aggregation yields the selection $s^{(0)} = (1, 1, 0, 0, 1, 0)$ and $s^{(1)} = (1, 0, 1, 1, 0, 0)$, with a quality of $\min\{19, 19\} = 19$.* Greedy Replacement *and exact sequential search yield the selection $s^{(0)} = (1, 1, 1, 0, 0, 0)$ and $s^{(1)} = (1, 0, 0, 1, 1, 0)$, with a quality of $\min\{24, 14\} = 14$. Exact simultaneous search with sum-aggregation may yield either of these two solutions or the selection $s^{(0)} = (1, 1, 0, 1, 0, 0)$ and $s^{(1)} = (1, 0, 1, 0, 1, 0)$, all with the same sum-aggregated quality of 38 but different min-aggregated quality.*

In particular, *Greedy Replacement* is sequential, so it does not balance feature-set qualities. We alleviate this issue with *Greedy Balancing* (cf. Section 6.2.5.2).

Limitations Proposition 17 and Examples 4, 5 already showed the potential quality loss of *Greedy Replacement* compared to an exact search. Further, the heuristic only works as long as some features have not been part of any feature set yet, i.e., $k + a \cdot \lceil \tau \cdot k \rceil \leq n$. Thus, a smaller feature-set size k and dissimilarity threshold τ enable a higher number of alternatives a. Beyond the scope of this dissertation, [16] introduces *Greedy Depth Search*, which extends the procedure of *Greedy Replacement* to yield more alternatives but with a higher and more variable runtime per alternative. Another drawback is that *Greedy Replacement* assumes univariate feature qualities. For more complex objectives, quality-based feature ordering (Line 1) may be impossible or suboptimal. Further, *Greedy Replacement* cannot accommodate additional constraints on feature sets, e.g., based on domain knowledge. Finally, the heuristic assumes the same size k for all feature sets.

Algorithm 8: *Greedy Balancing* for alternative feature selection.

Input: Univariate feature qualities $q \in \mathbb{R}^n$,
Feature-set size $k \in \mathbb{N}$,
Number of alternatives $a \in \mathbb{N}_0$,
Dissimilarity threshold $\tau \in [0, 1]$
Output: List of feature-selection decision vectors $s^{(0)}, \ldots, s^{(a)}$

```
1  if ⌈τ · k⌉ · a + k > n then return ∅
2  indices ← sort_indices(q, order=descending)            // Order by qualities
3  for l ← 0 to a do                                      // Initial selection for all alternatives
4      s^(l) ← {0}^n
5  feature_position ← 1                                   // Index of index of current feature
6  while feature_position ≤ ⌊(1 − τ) · k⌋ do               // Select top features
7      j ← indices[feature_position]                      // Index feature by quality
8      for l ← 0 to a do                                  // Same features in all alternatives
9          s_j^(l) ← 1
10     feature_position ← feature_position + 1
11 for l ← 0 to a do
12     Q^(l) ← 0                                          // Relative quality of each alternative
13 while feature_position ≤ ⌈τ · k⌉ · a + k do             // Fill all positions
14     Q_min ← ∞                                          // Find alternative with lowest quality
15     l_min ← −1
16     for l ← 0 to a do
17         if Q^(l) < Q_min and Σ_{j=1}^n s_j^(l) < k then // Check cardinality
18             Q_min ← Q^(l)
19             l_min ← l
20     j ← indices[feature_position]                      // Index feature by quality
21     s_j^(l_min) ← 1                                    // Add to lowest-quality, non-full alternative
22     Q^(l_min) ← Q^(l_min) + q_j                        // Update quality of that alternative
23     feature_position ← feature_position + 1
24 return s^(0), ..., s^(a)
```

6.2.5.2. Greedy Balancing

Algorithm Algorithm 8 outlines *Greedy Balancing*, which modifies *Greedy Replacement* to obtain more balanced feature-set qualities via a simultaneous search. In particular, it tries to distribute the individual feature qualities evenly to alternatives. First, we check whether the algorithm should terminate early, i.e., whether the number of features n is not high enough to satisfy the desired user parameters k, a, and τ (Line 1). Next, we select the first $\lfloor (1 - \tau) \cdot k \rfloor$ features in each alternative like in *Greedy Replacement* (cf. Algorithm 7), i.e., we pick the features with the highest quality q_j (Lines 2–10). For the remaining spots

in the alternatives, we use a Longest Processing Time (LPT) heuristic (Lines 11–23), which is common for Multiprocessor Scheduling and Balanced Number Partitioning problems [14, 40, 124]. In particular, we continue iterating over features by decreasing quality. We assign each feature to the alternative that currently has the lowest summed quality $Q^{(l)}$ and whose size k has not been reached yet. We continue this procedure until all alternatives have reached size k (Line 13).

Example 6 (Algorithm of *Greedy Balancing*)**.** *Consider $n = 6$ features with univariate feature qualities $q = (9, 8, 7, 3, 2, 1)$, feature-set size $k = 4$, number of alternatives $a = 1$, and dissimilarity threshold $\tau = 0.5$. The features with qualities 9 and 8 become part of both feature sets, $s^{(0)}$ and $s^{(1)}$, since $\lfloor (1 - \tau) \cdot k \rfloor = 2$ (Lines 2–10). At this point, both alternatives have the same relative quality $Q^{(0)} = Q^{(1)} = 0$, which ignores the quality of the shared features. Now the LPT heuristic becomes active (Lines 11–23). The feature with quality 7 is added to $s^{(0)}$, which causes $Q^{(0)} > Q^{(1)}$ (i.e., $7 > 0$). Thus, the feature with quality 3 is added to $s^{(1)}$. As $Q^{(0)} > Q^{(1)}$ (i.e., $7 > 3$) still holds, the feature with quality 2 becomes part of $s^{(1)}$ as well. Because $s^{(1)}$ has reached size $k = 4$, the feature with quality 1 is added to $s^{(0)}$, even if the latter still has a higher relative quality (i.e., $7 > 5$). Now both alternatives have reached their desired size and $n = 6 = \lceil 0.5 \cdot 4 \rceil \cdot 1 + 4 = \lceil \tau \cdot k \rceil \cdot a + k$ (Line 13). Thus, the algorithm terminates. The solution consists of $s^{(0)} = (1, 1, 1, 0, 0, 1)$ and $s^{(1)} = (1, 1, 0, 1, 1, 0)$.*

Time complexity Like *Greedy Replacement*, *Greedy Balancing* has an upfront cost of $O(n \cdot \log n)$ for sorting feature qualities (Line 2) and then iterates over $O(n)$ *feature_position*s (Lines 6 and 13). For each *feature_position*, the algorithm iterates over a alternatives (Lines 8 and 16) and conducts a constant number of operations each, which yields total update costs of $O(a \cdot n)$. This figure assumes cardinality checks (Line 17) can be done in $O(1)$, e.g., by storing the current feature-set sizes. There is also a total cost of $O(a \cdot n)$ for array initialization (Lines 3–4). Since $a < n$ (Line 1), the overall time complexity of *Greedy Balancing* is $O(n^2)$, as for *Greedy Replacement*.

Quality *Greedy Balancing* selects the same features as *Greedy Replacement* and only changes their assignment to the feature sets. Hence, the quality guarantee of *Greedy Replacement* (cf. Proposition 17) holds here as well:

Proposition 19 (Approximation quality of *Greedy Balancing*)**.** *Assume non-negative univariate feature qualities of n features (cf. Equation 2.1), $a \in \mathbb{N}_0$ alternatives, the Dice dissimilarity (cf. Equation 6.3) as $d(\cdot)$, a dissimilarity threshold $\tau \in [0, 1]$, desired feature-set size $k \in \mathbb{N}$, and $k + a \cdot \lceil \tau \cdot k \rceil \leq n$. Under these conditions,* Greedy Balancing *reaches at least a fraction of $\frac{\lfloor (1-\tau) \cdot k \rfloor}{k}$ of the optimal objective values of the optimization problems for (1) sequential search (cf. Definition 7), (2) simultaneous search with sum-aggregation (cf. Definition 9 and Equation 6.11), and (3) simultaneous search with min-aggregation (cf. Definition 9 and Equation 6.12).*

For min-aggregation, *Greedy Balancing* can even beat exact sequential search, which does not try to balance qualities. Example 5 demonstrates such a case, where the heuristic would

yield the same solution as exact simultaneous search with min-aggregation. However, the heuristic can also be worse than exact sequential and simultaneous search, as Example 4 shows, where *Greedy Balancing* would yield the same solution as *Greedy Replacement*.

Limitations *Greedy Balancing* shares several limitations with *Greedy Replacement*, e.g., it may be worse than exact search, assumes univariate feature qualities, and does not work if the number of features n is too low relative to k, a, and τ. In the latter case, *Greedy Balancing* yields no solution due to its simultaneous nature, while *Greedy Replacement* yields at least some alternatives. However, one could easily adapt *Greedy Balancing* to yield the highest feasible number of alternatives. Further, *Greedy Balancing* has the advantage of more balanced feature-set qualities.

6.3. Experimental Design

In this section, we introduce our experimental design for alternative feature selection. After a brief overview of its goal and components (cf. Section 6.3.1), we describe its components in detail: evaluation metrics (cf. Section 6.3.2), methods (cf. Section 6.3.3), and datasets (cf. Section 6.3.4). Finally, we briefly outline our implementation (cf. Section 6.3.5).

6.3.1. Overview

We conduct experiments with 30 binary-classification datasets. As evaluation metrics, we consider feature-set quality and runtime. We compare five feature-selection methods, representing different notions of feature-set quality. Also, we train prediction models with the resulting feature sets and analyze prediction performance. To find alternatives, we consider simultaneous and sequential search, both with solver-based and heuristic search methods. We systematically vary the two user parameters for searching alternatives, i.e., the number of alternatives a and the dissimilarity threshold τ.

6.3.2. Evaluation Metrics

Feature-set quality We evaluate feature-set quality with two metrics. First, we report the *objective value* $Q(s, X, y)$ of the feature-selection methods, which guided the search for alternatives. Second, we train prediction models with the found feature sets. We report *prediction performance* in terms of the Matthews correlation coefficient (MCC) [145]. This coefficient is insensitive to class imbalance, reaches its maximum of 1 for perfect predictions, and is 0 for random guessing as well as constant predictions.

We conduct a stratified five-fold cross-validation. In particular, the search for alternatives and model training only use the training data, while we employ the test data for evaluation: For the test-set objective value, we compute the objective on the test set but with the

feature selection from the training set. For the test-set prediction performance, we predict on the test set but use a prediction model trained with these features on the training set.

Runtime We consider two metrics related to runtime.

First, we analyze the *optimization time*. For white-box feature-selection methods in solver-based search for alternatives, we sum the measured runtime of solver calls. We exclude the time for computing feature qualities and feature dependencies for the objective since one can compute these values once per dataset and then reuse them in each solver call. For *Greedy Wrapper* feature selection and the heuristic search methods for alternatives, we measure the runtime of the corresponding search algorithms. For *Greedy Wrapper*, this search procedure involves multiple solver calls and trainings of a prediction model.

Second, we examine the *optimization status*, which can take four values for the solver-based search. If the solver finished before reaching a timeout, it either found an *optimal* solution or proved the problem *infeasible*, i.e., no solution exists. If the solver reached its timeout, it either found a *feasible* solution without proving its optimality or found no valid solution, though one might exist, so the problem is *not solved*. For the heuristic search methods, we only use *not solved* and *feasible* as statuses, as these search methods are neither guaranteed to find the optimum nor do they prove infeasibility if they terminate early.

6.3.3. Methods

We employ multiple methods for making predictions (cf. Section 6.3.3.1), feature selection (cf. Section 6.3.3.2), and searching alternatives (cf. Section 6.3.3.3).

6.3.3.1. Prediction

As prediction models, we use decision trees [34], which admit learning complex, non-linear dependencies from the data. Preliminary experiments with random forests [33] and k-nearest neighbors yielded similar insights. We leave the trees' hyperparameters at their defaults, except for using information gain instead of Gini impurity as the split criterion, to be consistent with our filter feature-selection methods. Note that tree models also select features themselves, so they may not use all features from the alternative feature sets. However, this is not an issue for our study. We are interested in which performance the models achieve if limited to certain feature sets, not how they use each available feature.

6.3.3.2. Feature Selection (Objective Functions)

We search for alternatives under different notions of feature-set quality in the objective function. We choose five well-known feature-selection methods that are easy to parameterize and cover the different categories from Section 2.1.2 except *embedded*, as explained in Section 6.2.3.3. However, we use feature importance from an embedded method as post-hoc

importance scores. Four feature-selection methods allow a white-box formulation of the optimization problem, while *Greedy Wrapper* is black-box. With each feature-selection method, we employ Equation 4.3 to obtain fixed feature-set sizes of $k \in \{5, 10\}$.

Filter feature selection We evaluate three filter methods, all using mutual information [115] as the dependency measure $q(\cdot)$. This measure can capture arbitrary dependencies rather than, e.g., just linear ones. *MI* denotes a univariate filter (cf. Equation 2.1), while *FCBF* (cf. Equation 6.14) and *mRMR* (cf. Equation 6.15) are multivariate. We normalize the mutual-information values per dataset and cross-validation fold to improve comparability: For *FCBF* and *MI*, we scale the individual features' qualities with a constant such that the overall objective value is in $[0, 1]$. For *mRMR*, we min-max-normalize all mutual-information values to $[0, 1]$, so the overall objective is in $[-1, 1]$.

Wrapper feature selection As a wrapper method, we employ our hill-climbing procedure *Greedy Wrapper* (cf. Algorithm 6) with *max_iters* = 1000. To evaluate feature-set quality in the wrapper, we apply a stratified 80:20 holdout split and train decision trees. $Q(s, X, y)$ corresponds to the prediction performance in terms of MCC on the 20% validation part.

Post-hoc feature importance As a post-hoc importance measure called *Model Gain*, we use importance scores from *scikit-learn's* decision trees. There, importance expresses a feature's contribution towards optimizing the split criterion of the tree, for which we choose information gain. These importances are normalized to sum up to 1 by default. We plug them into Equation 2.1, i.e., treat them like univariate filter scores, though they actually originate from trees trained with all features and thus are not univariate.

6.3.3.3. Alternatives (Constraints)

Overview In our evaluation, we categorize search methods for alternatives in two dimensions that are orthogonal to each other: Solver-based vs. heuristic and sequential vs. simultaneous. Also, we analyze the impact of the user parameters a and τ.

Solver-based search methods For the four feature-selection methods with white-box objectives, we use the integer-programming solver *SCIP* [27] to solve the underlying optimization problems exactly. Given sufficient solving time, these alternatives are globally optimal. For *Greedy Wrapper*, the search procedure (Algorithm 6) is heuristic (though still solver-based, so we place it in this category) and might not cover the entire search space. There, the solver only assists in finding valid solutions but does not optimize quality.

For each feature selection method, we analyze *sequential* (cf. Definition 7) and *simultaneous* (cf. Definition 9) solver-based search for alternatives. For the latter, we employ sum-aggregation (cf. Equation 6.11) and min-aggregation (cf. Equation 6.12) in the objective. In figures and tables, we use the abbreviations *seq.*, *sim. (sum)*, and *sim. (min)*.

Heuristic search methods We also compare heuristic search methods, which do not use a solver (cf. Section 6.2.5). In particular, we employ *Greedy Replacement* (cf. Algorithm 7), which is a sequential search method, and *Greedy Balancing* (cf. Algorithm 8), which is a simultaneous search method. In figures and tables, we use the abbreviations *rep.* and *bal.*. Since these heuristics assume univariate feature qualities (cf. Equation 2.1), we only combine them with the univariate feature-selection methods *MI* and *Model Gain*.

Search parametrization We vary the parameters of the search systematically: We evaluate $a \in \{1, \ldots, 10\}$ alternatives for sequential search methods and $a \in \{1, \ldots, 5\}$ for simultaneous search methods due to the higher runtime of the latter. For the dissimilarity threshold τ, we analyze all possible sizes of the feature-set overlap in the Dice dissimilarity (cf. Equations 6.3 and 6.8). Thus, for $k = 5$, we consider $\tau \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$, corresponding to an overlap of four to zero features. For $k = 10$, we consider $\tau \in \{0.1, 0.2, \ldots, 1.0\}$. We exclude $\tau = 0$, which would allow returning duplicate feature sets.

Timeout In solver-based search, we employ a timeout to enable a large-scale evaluation and account for the high variance of solver runtime. In particular, we grant each solver call 60 s multiplied by the number of feature sets sought. Thus, solver-based sequential search conducts multiple solver calls with 60 s timeout each, while solver-based simultaneous search conducts one solver call with proportionally more time, e.g., 300 s for five feature sets. For 84% of the feature sets in our evaluation, the solver finished before the timeout.

Competitors for search methods As discussed in Section 3.2.1, related work pursues different objective functions, operates with different notions of alternatives, and may only target particular feature-selection methods. All these points prevent a meaningful comparison to our search methods. E.g., a feature set deemed alternative in related work may violate our constraints for alternatives. Further, we can still put the feature-set quality into perspective by comparing alternatives to each other. In particular, the original feature set, i.e., without constraints for alternatives, serves as a natural reference point.

6.3.4. Datasets

We use a variety of datasets from the Penn Machine Learning Benchmarks (PMLB) [173, 186]. To harmonize evaluation, we only consider binary-classification datasets, though alternative feature selection also works for regression and multi-class problems. We exclude datasets with less than 100 data objects since they may entail a high uncertainty when assessing feature-set quality. Also, we exclude datasets with less than 15 features to leave room for alternatives. Next, we exclude one dataset with 1000 features, which would dominate the overall runtime. Finally, we manually exclude datasets that seem to be duplicated or modified versions of other datasets. Consequently, we obtain 30 datasets with 106 to 9822 data objects and 15 to 168 features (cf. Table 6.2). The datasets do not contain any missing values. Categorical features have an ordinal encoding by default.

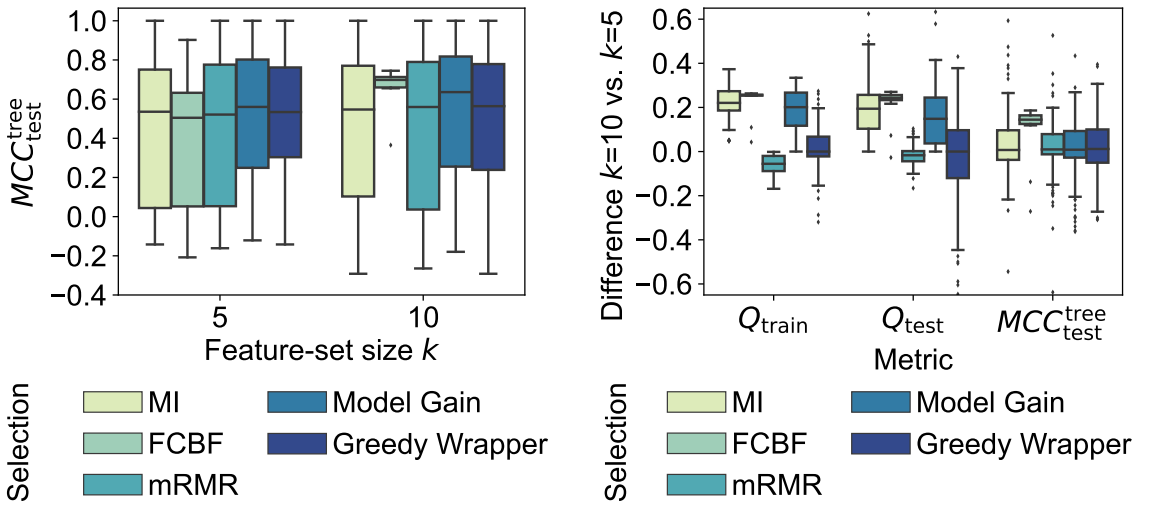
Table 6.2.: Datasets from PMLB used in our experiments. m denotes the number of data objects and n the number of features.

Dataset	m	n
backache	180	32
chess	3196	36
churn	5000	20
clean1	476	168
clean2	6598	168
coil2000	9822	85
credit_a	690	15
credit_g	1000	20
dis	3772	29
GAMETES_Epistasis_2_Way_20atts_0.1H_EDM_1_1	1600	20
GAMETES_Epistasis_2_Way_20atts_0.4H_EDM_1_1	1600	20
GAMETES_Epistasis_3_Way_20atts_0.2H_EDM_1_1	1600	20
GAMETES_Heterogeneity_20atts_1600_Het_0.4_0.2_50_EDM_2_001	1600	20
GAMETES_Heterogeneity_20atts_1600_Het_0.4_0.2_75_EDM_2_001	1600	20
hepatitis	155	19
Hill_Valley_with_noise	1212	100
horse_colic	368	22
house_votes_84	435	16
hypothyroid	3163	25
ionosphere	351	34
molecular_biology_promoters	106	57
mushroom	8124	22
ring	7400	20
sonar	208	60
spambase	4601	57
spect	267	22
spectf	349	44
tokyo1	959	44
twonorm	7400	20
wdbc	569	30

6.3.5. Implementation and Execution

We implemented our experimental pipeline in Python 3.8, using *scikit-learn* [176] for machine learning and the integer-programming solver *SCIP* [27] via the package *OR-Tools* [178] for solver-based search. All code is available online (cf. Section 1.4). We organized the alternative-feature-selection methods as a Python package to ease reuse.

Our experimental pipeline parallelizes over datasets, cross-validation folds, and feature-selection methods, while each of these experimental tasks runs single-threaded. We ran



(a) Test-set prediction performance by feature-set size k.

(b) Difference in feature-set quality between $k = 10$ and $k = 5$ by evaluation metric. Y-axis is truncated to improve readability.

Figure 6.1.: Distribution of feature-set quality over datasets and cross-validation folds, by feature-selection method. Results from the original feature sets of solver-based sequential search.

the pipeline on a server with 160 GB RAM and an *AMD EPYC 7551* CPU, having 32 physical cores and a base clock of 2.0 GHz. With this hardware, the parallelized pipeline run took approximately 249 hours, i.e., about 10.4 days.

6.4. Evaluation

In this section, we evaluate our experiments for alternative feature selection. After comparing feature-selection methods without constraints (cf. Section 6.4.1), we discuss the parametrization for searching alternatives: the search methods (cf. Section 6.4.2) and the two user parameters a and τ (cf. Section 6.4.3). Section 6.4.4 summarizes key findings.

6.4.1. Feature-Selection Methods

Prediction performance The five feature-selection methods in our experiments employ different objective functions $Q(s, X, y)$, so comparing objective values between them does not make sense. However, we can analyze the prediction performance of the obtained feature sets. Figure 6.1a displays the distribution of a decision tree's test-set MCC on the original feature sets, i.e., without constraints, for the feature-selection methods. The mean test-set MCC is 0.53 for *Model Gain* and *Greedy Wrapper*, 0.47 for *MI*, 0.46 for *mRMR*, and 0.43 for *FCBF*. Thus, the analyses of alternative feature sets in Sections 6.4.2 and 6.4.3 focus on *Model Gain* while still discussing the remaining feature-selection methods.

The univariate, model-free method *MI* keeps up surprisingly well with more sophisticated methods. It uses the same objective function as *Model Gain* but obtains its feature qualities from a bivariate dependency measure rather than a prediction model.

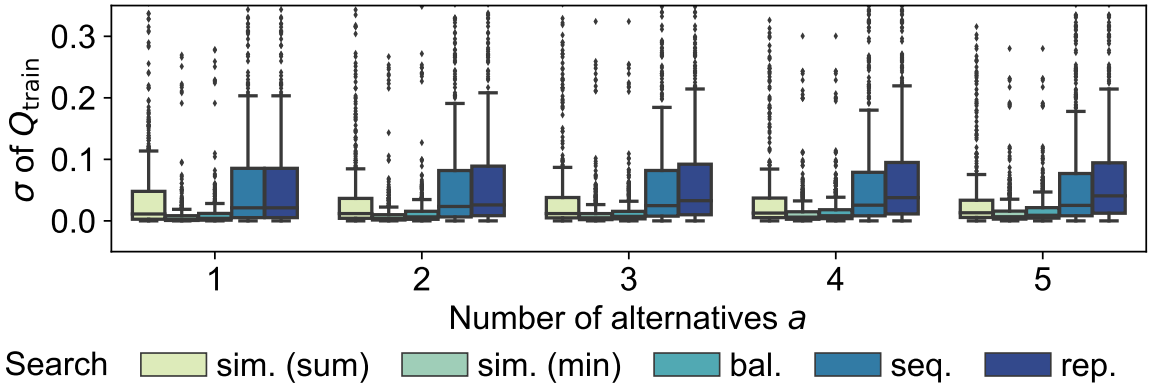
Greedy Wrapper uses prediction performance to assess feature-set quality but employs a search heuristic instead of optimizing globally. In particular, it performed 661 iterations on average to determine the original feature sets. However, the number of possible feature sets is higher, e.g., already $\binom{15}{5} = 3003$ for the lowest-dimensional dataset in our evaluation (cf. Table 6.2) and $k = 5$. The still high prediction performance comes at the expense of high runtime (cf. Table 6.5), so we prefer *Model Gain* for later evaluations.

FCBF's results may be taken with a grain of salt: The original feature set in solver-based sequential search is already infeasible, i.e., no solution satisfied the constraints, in 71% of the cases for *FCBF* but never for the other feature-selection methods. Over all solver-based search runs, even 89% of the feature sets were infeasible for *FCBF* but only 18% for *Model Gain*. In particular, the combination of feature-correlation constraints in our formulation of *FCBF* (cf. Equation 6.14) with a cardinality constraint (cf. Equation 4.3), i.e., enforcing a feature-set size k, may make the problem infeasible, especially if k gets larger.

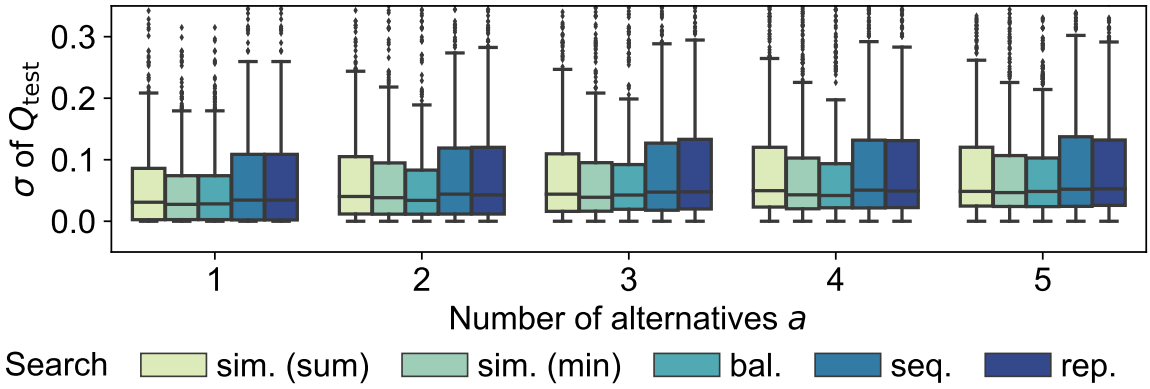
Influence of feature-set size k One could expect larger feature sets to exhibit a higher feature-set quality than smaller ones, but the picture in our experiments is more nuanced. In particular, quality may not increase proportionally with k or may even decrease. As Figure 6.1b shows for the original feature sets of solver-based sequential search, *MI* and *Model Gain* exhibit an increase of the training-set objective value Q_{train} from $k = 5$ to $k = 10$, i.e., the difference depicted in Figure 6.1b is positive. As these objectives are monotonic and the feature qualities are non-negative, a decrease in the training-set objective value is impossible. In contrast, *Greedy Wrapper* with its black-box objective does not necessarily benefit from more features. The latter insight also applies to *mRMR*, which normalizes its objective with the number of selected features and penalizes feature redundancy. For *FCBF*, the fraction of feasible feature sets changes considerably from $k = 5$ to $k = 10$, so the overall quality between these two settings should not be compared. As Figure 6.1b also displays, the benefit of larger feature sets is even less clear for prediction performance. In particular, all feature-selection methods except *FCBF* show a median difference in test-set MCC close to zero when comparing $k = 5$ to $k = 10$.

6.4.2. Search Methods for Alternatives

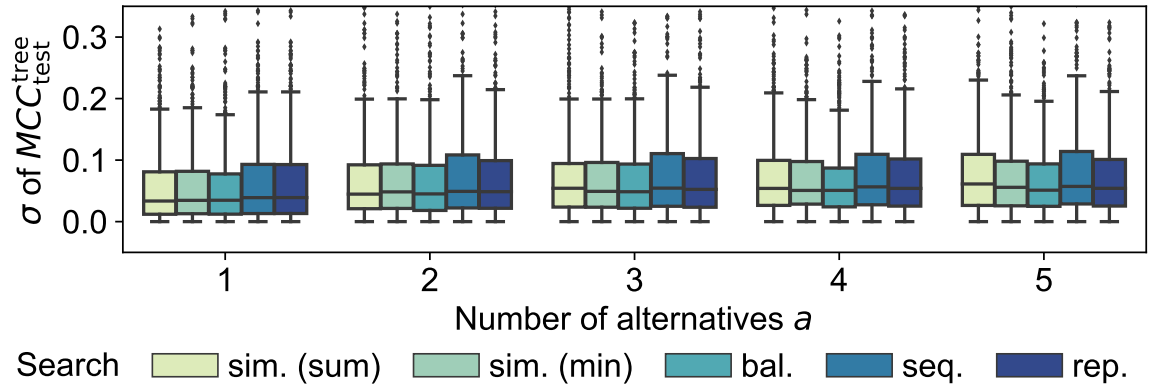
Variance in feature-set quality As expected, the search method influences how much the training-set objective value Q varies between multiple alternatives obtained for the same experimental settings, i.e., within one search run for alternatives. Figure 6.2a visualizes this result for *Model Gain* as the feature-selection method and $k = 5$. The figure shows how the standard deviation of the training-set objective value within individual search runs for alternatives is distributed over other experimental settings, e.g., datasets and cross-validation folds. In particular, the quality of multiple alternatives varies more if



(a) Training-set objective value.



(b) Test-set objective value.



(c) Test-set prediction performance.

Figure 6.2.: Distribution of standard deviation of feature-set quality within search runs over datasets, cross-validation-folds, and τ, by search method for alternatives and number of alternatives a. Results with *Model Gain* as the feature-selection method and $k = 5$. Excludes search settings where at least one combination of search method and a yielded no valid solution. Y-axes are truncated to improve readability.

they are found by solver-based sequential search rather than solver-based simultaneous search. For solver-based simultaneous search, min-aggregation yields considerably more homogeneous feature-set quality than sum-aggregation. These findings apply to all white-box feature-selection methods but not *Greedy Wrapper*.

The heuristic search method *Greedy Balancing* yields a small variance of training-set objective value within search runs, only slightly higher than for solver-based simultaneous search with min-aggregation. In contrast, *Greedy Replacement* rather mimics solver-based sequential search, having a substantial variance of quality. Additionally, the variance of *Greedy Replacement* noticeably grows with the number of alternatives a.

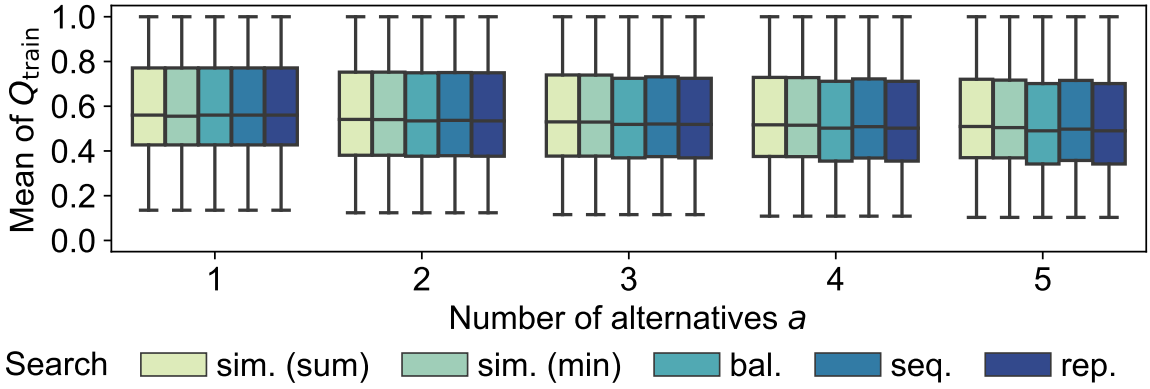
As Figures 6.2b and 6.2c show, the variance of feature-set quality differs considerably less between the search methods on the test set, for the objective value as well as prediction performance. This effect might result from overfitting: Even if the training-set quality is similar, some alternatives might generalize better than others. Thus, this variance caused by overfitting could alleviate the effect caused by the choice of search method.

Average value of feature-set quality While obtaining alternatives of homogeneous quality can be one goal of simultaneous search, another selling point would be obtaining alternatives of higher average quality than sequential search. However, this potential advantage rarely materialized in our experiments. In particular, Figure 6.3a compares the distribution of the mean training-set objective value in search runs with *Model Gain* as the feature-selection method and $k = 5$. We observe that all search methods yield very similar overall distributions of average feature-set quality. Comparing solver-based sequential and simultaneous search on each experimental setting separately and then aggregating also shows a mean quality difference close to zero. Further, outliers can occur in both directions, i.e., either solver-based search method may yield higher quality.

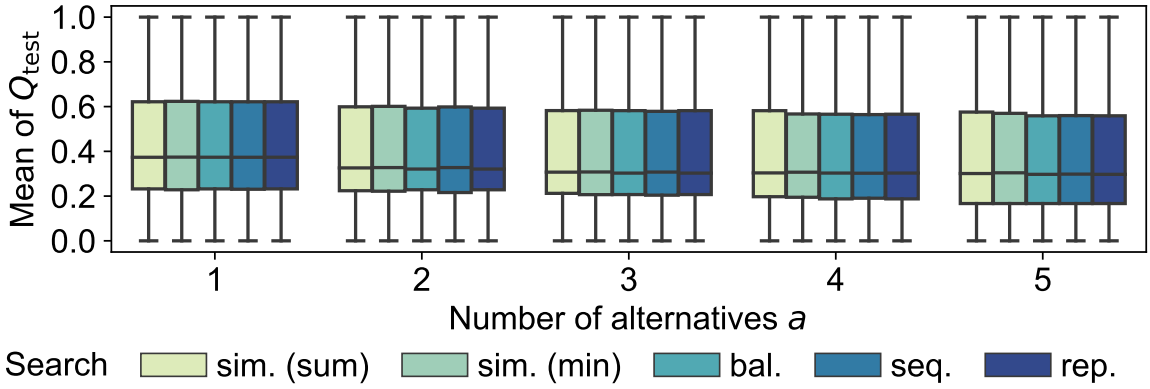
The mean test-set objective value in Figure 6.3b and the mean test-set prediction performance in Figure 6.3c also exhibit a negligible quality difference between the search methods. Finally, the other four feature-selection methods besides *Model Gain* do not show a general quality advantage of solver-based simultaneous search either.

Quality difference of heuristics We now look closer at the quality difference between solver-based and heuristic search. Figure 6.4 compares the mean training feature-set quality within search runs for each experimental setting separately. In particular, we compare solver-based simultaneous search with min-aggregation to *Greedy Balancing* (cf. Figures 6.4a and 6.4c) and solver-based sequential search to *Greedy Replacement* (cf. Figures 6.4b and 6.4d). Positive values in Figure 6.4 express that solver-based search yields higher quality; negative values favor heuristic search. The latter can occur if solver-based search yields suboptimal solutions due to timeouts, as we analyze later (cf. Table 6.3).

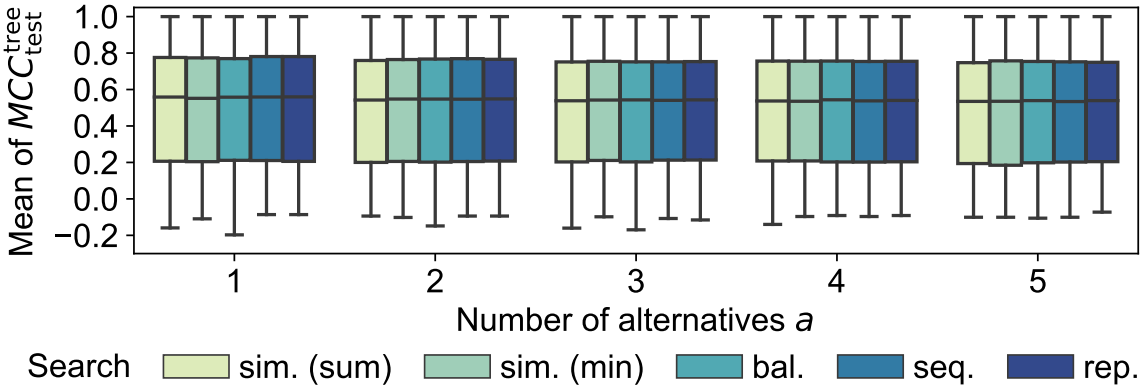
Figures 6.4a and 6.4b compare solver-based and heuristic search over the number of alternatives a. For $a = 1$, solver-based and heuristic search yield the same mean training feature-set quality except when timeouts occur. The more alternatives are desired, the more advantageous a solver-based search is quality-wise. As in prior analyses, the picture is less clear on the test set, which shows a smaller quality difference here. Further, the difference in mean training feature-set quality between *Greedy Balancing* and solver-based simultaneous search (cf. Figure 6.4a) grows faster with a than between *Greedy Replacement* and solver-based sequential search (cf. Figure 6.4b). As an explanation, consider that



(a) Training-set objective value.



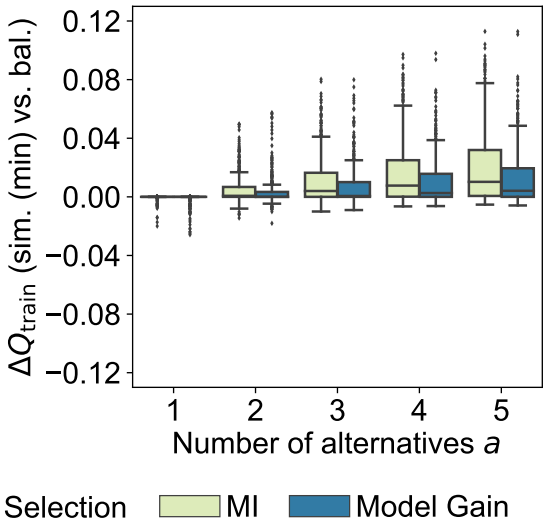
(b) Test-set objective value.



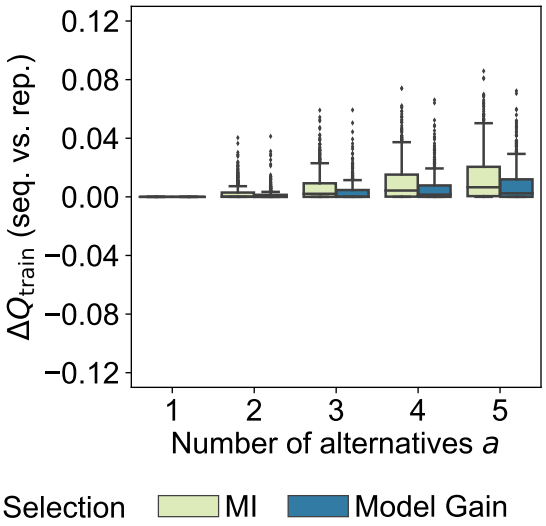
(c) Test-set prediction performance.

Figure 6.3.: Distribution of mean feature-set quality within search runs over datasets, cross-validation-folds, and τ, by search method for alternatives and number of alternatives a. Results with *Model Gain* as the feature-selection method and $k = 5$. Excludes search settings where at least one combination of search method and a yielded no valid solution. Y-axes are truncated to improve readability.

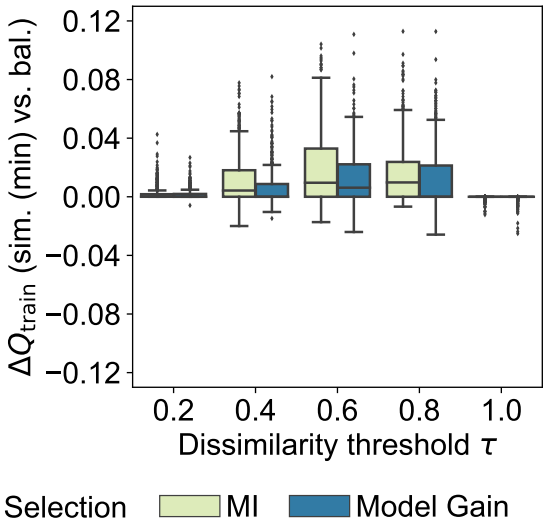
optimal simultaneous search may generally develop a quality advantage over optimal sequential search for more alternatives. *Greedy Balancing* selects the same features as *Greedy Replacement*, i.e., a sequential search heuristic, and only distributes them differently into feature sets, yielding the same mean feature-set quality.



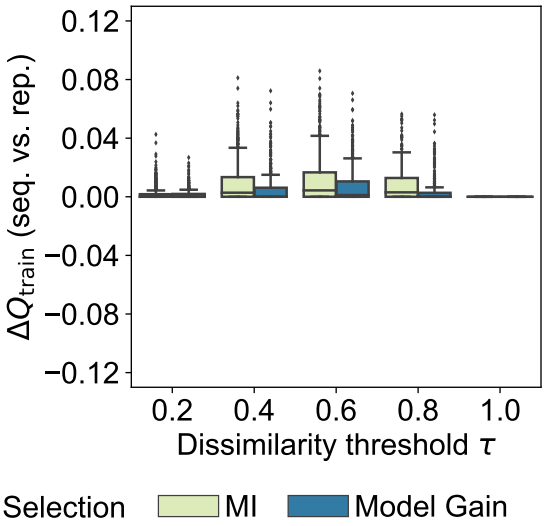
(a) Difference between solver-based simultaneous (min-aggregation) search and *Greedy Balancing*, by the number of alternatives a.



(b) Difference between solver-based sequential search and *Greedy Replacement*, by the number of alternatives a.



(c) Difference between solver-based simultaneous (min-aggregation) search and *Greedy Balancing*, by the dissimilarity threshold τ.



(d) Difference between solver-based sequential search and *Greedy Replacement*, by the dissimilarity threshold τ.

Figure 6.4.: Distribution of difference in mean training feature-set quality within a search run between solver-based and heuristic search methods over remaining experimental settings, by feature-selection method. Results with $k = 5$. Excludes search settings where at least one combination of search method and a yielded no valid solution.

Figures 6.4c and 6.4d compare solver-based and heuristic search over the dissimilarity threshold τ. Unlike for a, the difference in mean training feature-set quality between solver-based and heuristic search does not increase over the whole range of τ but shows an increase followed by a decrease. In particular, $\tau = 1$ allows the two heuristic search methods to reach the same mean training feature-set quality as the solver-based methods.

Table 6.3.: Frequency of optimization statuses (cf. Section 6.3.2) over datasets, cross-validation folds, a, and τ, by feature-selection method and search method for alternatives. Results with $k = 5$, $a \in \{1, 2, 3, 4, 5\}$, and excluding *Greedy Wrapper*, which does not use the solver for optimizing. Each row adds up to 100%.

Feature selection	Search	Optimization status			
		Infeasible	Not solved	Feasible	Optimal
FCBF	seq.	74.51%	0.00%	0.00%	25.49%
FCBF	sim. (min)	73.07%	0.00%	1.60%	25.33%
FCBF	sim. (sum)	73.07%	0.00%	2.19%	24.75%
MI	bal.	0.00%	9.20%	90.80%	0.00%
MI	rep.	0.00%	9.20%	90.80%	0.00%
MI	seq.	4.93%	0.00%	0.00%	95.07%
MI	sim. (min)	4.67%	0.00%	9.33%	86.00%
MI	sim. (sum)	4.67%	0.00%	3.04%	92.29%
Model Gain	bal.	0.00%	9.20%	90.80%	0.00%
Model Gain	rep.	0.00%	9.20%	90.80%	0.00%
Model Gain	seq.	4.93%	0.00%	0.00%	95.07%
Model Gain	sim. (min)	4.67%	0.00%	5.28%	90.05%
Model Gain	sim. (sum)	4.67%	0.00%	1.87%	93.47%
mRMR	seq.	4.88%	0.00%	9.55%	85.57%
mRMR	sim. (min)	4.67%	0.00%	48.64%	46.69%
mRMR	sim. (sum)	4.67%	0.00%	67.04%	28.29%

Table 6.4.: Frequency of optimization statuses (cf. Section 6.3.2) over datasets, cross-validation folds, feature-selection methods, and τ, by number of alternatives a. Results from solver-based simultaneous search with sum-aggregation, $k = 5$, and excluding *Greedy Wrapper*. Each row adds up to 100%.

a	Optimization status		
	Infeasible	Feasible	Optimal
1	16.10%	7.60%	76.30%
2	17.50%	13.27%	69.23%
3	20.00%	20.20%	59.80%
4	27.00%	21.43%	51.57%
5	28.23%	30.17%	41.60%

This observation corresponds to our theoretical result that optimizing the summed quality of alternatives with $\tau = 1$ admits polynomial-time algorithms (cf. Proposition 16).

Optimization status Suboptimal search results are one reason why solver-based simultaneous search fails to consistently beat solver-based sequential search quality-wise. For *Greedy Wrapper*, the search is heuristic per se and does not cover the entire search space. For all feature-selection methods, the solver can time out. Table 6.3 shows that solver-based simultaneous search has a higher likelihood of timeouts than solver-based sequential

Table 6.5.: Mean optimization time over datasets, cross-validation folds, a, and τ, by feature-selection method and search method for alternatives. Results with $k = 5$ and $a \in \{1, 2, 3, 4, 5\}$.

Feature selection	Optimization time				
	Bal.	Rep.	Seq.	Sim. (min)	Sim. (sum)
FCBF	—	—	0.22 s	11.41 s	12.62 s
Greedy Wrapper	—	—	52.62 s	68.39 s	70.36 s
MI	0.00 s	0.00 s	0.03 s	47.39 s	24.56 s
Model Gain	0.00 s	0.00 s	0.03 s	30.38 s	19.08 s
mRMR	—	—	33.59 s	156.00 s	189.25 s

search, likely due to the larger size of the optimization problem (cf. Table 6.1). In particular, for up to five alternatives and $k = 5$, all solver-based sequential searches for *FCBF*, *MI*, and *Model Gain* finished within the timeout, i.e., yielded the optimal feature set or ascertained infeasibility, while *mRMR* had about 10% timeouts. In contrast, for solver-based simultaneous search with sum-aggregation, all feature-selection methods experience timeouts: 1–3% of the searches for *FCBF*, *MI*, and *Model Gain*, and 67% of the searches for *mRMR* found a feasible solution but could not prove optimality. Such timeout-affected simultaneous solutions can be worse than optimal sequential solutions.

mRMR is especially prone to suboptimal solutions, likely because it has a more complex objective (cf. Equation 6.15) than *MI* and *Model Gain*. *FCBF* often results in infeasible optimization problems since its constraints, which prevent the selection of redundant features (cf. Equation 6.14), might prevent finding any valid feature set of size k. Min-aggregation instead of sum-aggregation in solver-based simultaneous search exhibits more timeouts for *MI* and *Model Gain* but less for *FCBF* and *mRMR*. Still, solver-based sequential search incurs fewer timeouts for all of these four feature-selection methods.

Also, note that the fraction of timeouts in solver-based simultaneous search strongly depends on the number of alternatives a, as Table 6.4 displays: For $k = 5$ and sum-aggregation, roughly 8% of the white-box searches timed out for $a = 1$, but 20% for $a = 3$ and 30% for $a = 5$. While we grant solver-based simultaneous searches proportionally more time for multiple alternatives (cf. Section 6.3.3.3), the observed increase in timeouts suggests that runtime increases super-proportionally with a, as we analyze later.

For the heuristic search methods, Table 6.3 shows that *Greedy Replacement* more often did not find a valid alternative (9.20%) than solver-based sequential search (4.93%). A similar phenomenon occurred for *Greedy Balancing* (9.20%) compared to solver-based simultaneous search (4.67%). Such a result can be expected since both heuristic search methods stop early as soon as each feature is part of at least one alternative.

Optimization time As Table 6.5 shows, solver-based sequential search is faster on average than solver-based simultaneous search for all five feature-selection methods. In particular, the difference is up to three orders of magnitude for the four white-box feature-selection

Table 6.6.: Mean optimization time over datasets, cross-validation folds, and τ, by number of alternatives and feature-selection method. Results from solver-based simultaneous search with sum-aggregation and $k = 5$.

a	Optimization time				
	FCBF	Greedy Wrapper	MI	Model Gain	mRMR
1	0.45 s	28.44 s	0.03 s	0.02 s	44.68 s
2	0.86 s	41.76 s	0.09 s	0.08 s	117.62 s
3	2.97 s	62.70 s	0.30 s	0.27 s	208.14 s
4	13.32 s	96.65 s	3.68 s	3.47 s	258.19 s
5	45.52 s	122.26 s	118.72 s	91.58 s	317.63 s

methods. Further, *FCBF*, *MI*, and *Model Gain* experience a dramatic increase in optimization time with the number of alternatives a in solver-based simultaneous search, as Table 6.6 displays. In contrast, the runtime increase is considerably less for solver-based sequential search, which shows an approximately linear trend with the number of alternatives.

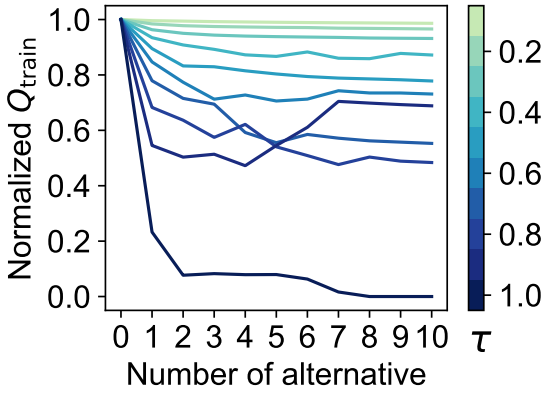
Table 6.5 also shows that the optimization time of the heuristic search methods for *MI* and *Model Gain* is negligible. In particular, *Greedy Replacement* and *Greedy Balancing* never took longer than 1 ms per search run for alternatives. These results highlight the runtime advantage of the heuristics, particularly of *Greedy Balancing* for simultaneous search.

An interesting question for practitioners is how the runtime relates to n, the number of features in the dataset. One could expect a positive correlation since the problem instance increases with n. Roughly speaking, this trend appears in our experimental data indeed. However, the observed trend is rather noisy, particularly for solver-based simultaneous search, and some higher-dimensional datasets even show lower average runtimes than lower-dimensional ones. This result indicates that other factors than n influence runtime as well, e.g., other experimental settings or the solver's internal search heuristics.

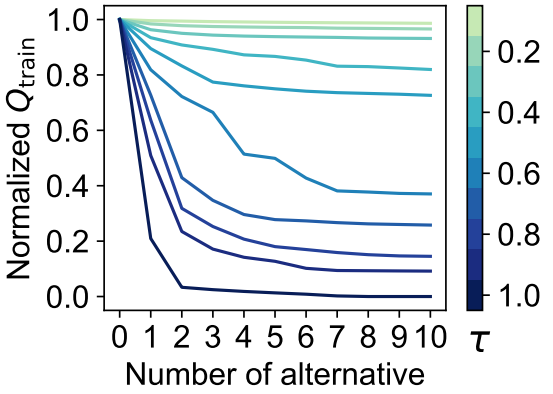
Based on all results described in this section, we focus on solver-based sequential search in the next section. In particular, it was significantly faster than solver-based simultaneous search while yielding similar feature-set quality.

6.4.3. User Parameters a And τ

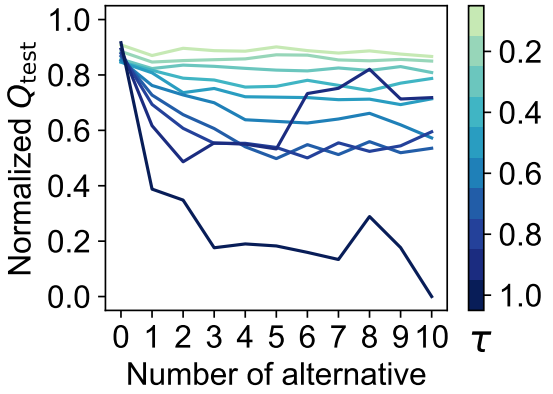
Feature-set quality Higher values of the two user parameters introduce more (for a) or stronger (for τ) constraints into the optimization problem of alternative feature selection. Thus, one would expect a corresponding decrease in feature-set quality. Figure 6.5 illustrates this trend for *Model Gain* as the feature-selection method and $k = 10$. Since the maximum feature-set quality varies among datasets, we max-normalize quality in this figure. In particular, we set the highest feature-set quality in each search run for alternatives to 1 and scale the other feature-set qualities accordingly. For prediction performance in terms of MCC, we shift its range from $[-1, 1]$ to $[0, 1]$ before normalization.



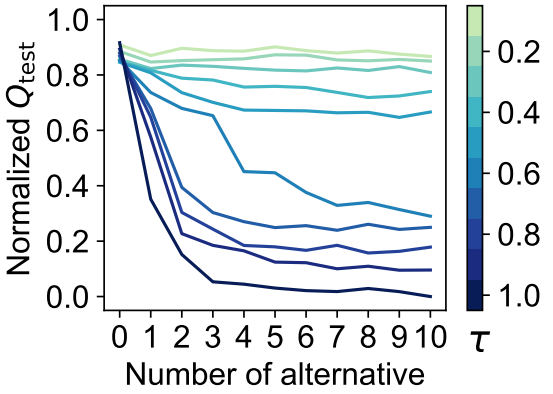
(a) Training-set objective value. Infeasible feature sets excluded.



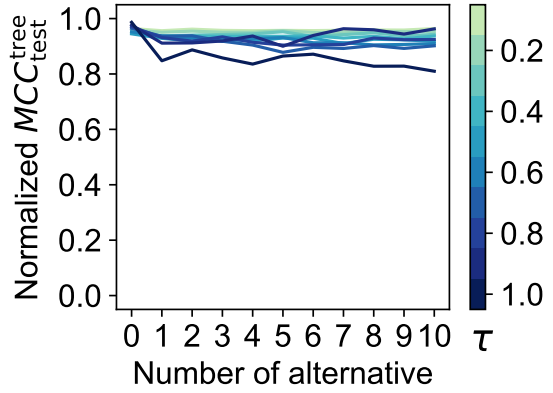
(b) Training-set objective value. Infeasible feature sets assigned a quality of 0.



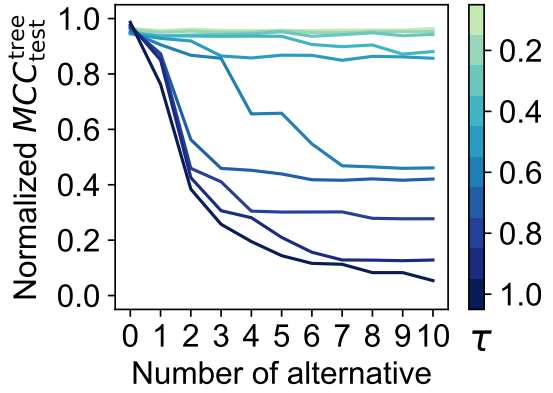
(c) Test-set objective value. Infeasible feature sets excluded.



(d) Test-set objective value. Infeasible feature sets assigned a quality of 0.

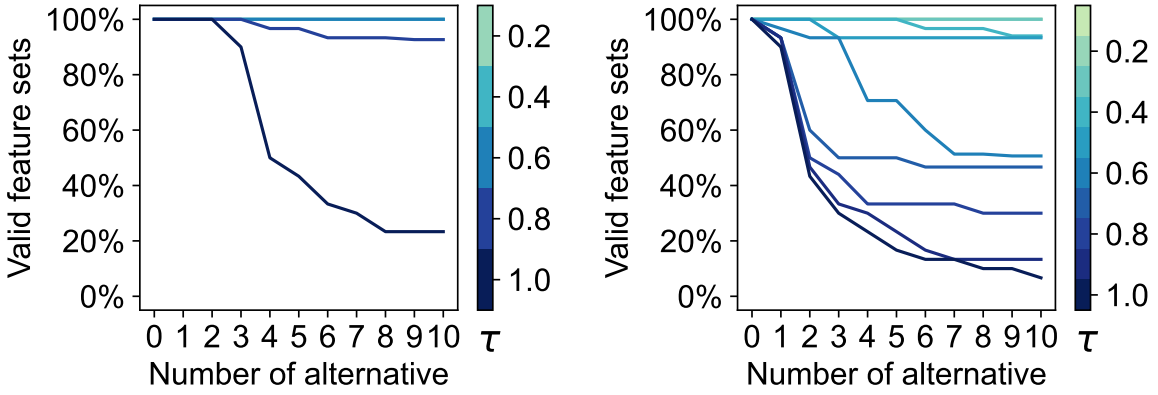


(e) Test-set prediction performance. Infeasible feature sets excluded.



(f) Test-set prediction performance. Infeasible feature sets assigned a quality of 0.

Figure 6.5.: Mean feature-set quality over datasets and cross-validation folds, max-normalized per search run for alternatives, by the number of alternative and dissimilarity threshold τ. Results from solver-based sequential search with *Model Gain* as the feature-selection method and $k = 10$.



(a) Feature-set size $k = 5$.

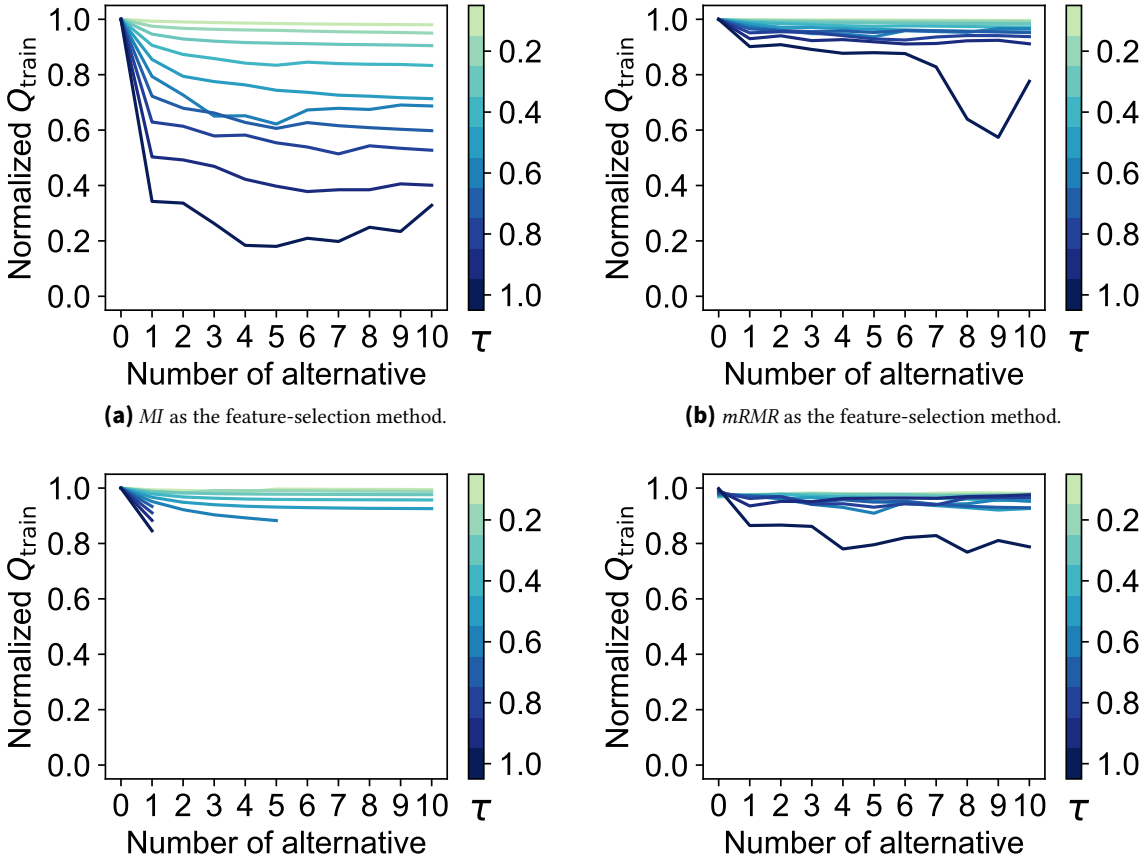
(b) Feature-set size $k = 10$.

Figure 6.6.: Frequency of optimization runs yielding a valid feature set over datasets and cross-validation folds, by the number of alternative and dissimilarity threshold τ. Results from solver-based sequential search with *Model Gain* as the feature-selection method.

Figure 6.5 shows that multiple alternatives may have a similar quality. Further, the training-set objective value (cf. Figure 6.5a) decreases most from the original feature set, i.e., the zeroth alternative, to the first alternative, but less beyond. Also, the decrease strongly depends on the dissimilarity threshold τ. For a low dissimilarity threshold like $\tau = 0.1$, the training-set objective value barely drops over the number of alternatives. Additionally, note that Figure 6.5 averages the normalized feature-set quality over multiple datasets. In our experiments, datasets with more features tend to experience a smaller decrease in quality over a and τ. As higher-dimensional datasets offer more options for alternatives, this observation makes sense. However, this effect is not guaranteed since datasets with many features could also contain many useless features instead of interesting alternatives.

The overall decrease in quality is slightly less pronounced for the test-set objective value (Figure 6.5c) than on the training set (Figure 6.5a) since overfitting might occur. In particular, the original feature set can even have lower test-set quality than the subsequent alternatives. The trend becomes even less clear for prediction performance, which varies little over a and τ in our experiments (cf. Figure 6.5e). In general, the optimization objective Q may only partially indicate actual prediction performance since the former may use a simplified feature-set quality criterion. Indeed, the correlation between optimization objective Q and prediction MCC is only weak to moderate in our experiments.

Optimization status The previous observations refer to the quality of the found feature sets. However, the more alternatives one desires and the more they should differ, the likelier an infeasible optimization problem is. Figure 6.6 visualizes the fraction of valid feature sets over the number of alternatives and dissimilarity threshold τ, showing the expected trend. Additionally, Figures 6.5b, 6.5d, and 6.5f display the same data as Figures 6.5a, 6.5c, and 6.5e but with the quality of infeasible feature sets set to zero instead of excluding these feature sets from evaluation. Consequently, the decrease in feature-set quality over a and τ is noticeably stronger. In contrast, if only considering valid feature sets, the mean quality in our experiments can increase over the number of alternatives, e.g., as visible in



(a) *MI* as the feature-selection method.

(b) *mRMR* as the feature-selection method.

(c) *FCBF* as the feature-selection method.

(d) *Greedy Wrapper* as the feature-selection method.

Figure 6.7.: Mean training-set objective value over datasets and cross-validation folds, max-normalized per search run for alternatives, by the number of alternative and dissimilarity threshold τ. Results from solver-based sequential search with $k = 10$. Infeasible feature sets excluded.

Figures 6.5a and 6.5c for $\tau = 0.9$. This counterintuitive phenomenon can occur because some datasets run out of valid feature sets sooner than others, so the average quality may be determined for different sets of datasets at each number of alternatives.

Influence of feature-selection method While we discussed *Model Gain* before, the decrease in objective value over a and τ occurs to different extents for the other feature-selection methods in our experiments, as Figure 6.7 displays. In this figure, we shifted the objectives of *Greedy Wrapper* and *mRMR* to $[0, 1]$ before max-normalization since their original range is $[-1, 1]$. *MI* (cf. Figure 6.7a) shows a similar behavior as *Model Gain* (cf. Figure 6.5a), which may result from both feature-selection methods using the same objective function, though with different feature qualities. In contrast, *mRMR* (cf. Figure 6.7b) exhibits a considerably smaller effect of increasing τ. For *FCBF* (cf. Figure 6.7c), the additional constraints on feature-feature correlation (cf. Equation 6.14) cause many infeasible results (cf. Table 6.3), so we cannot determine the average feature-set quality for some combinations of a and τ. For *Greedy Wrapper* (cf. Figure 6.7d), a and τ barely make any difference, which may be explained by the heuristic, inexact search procedure.

6.4.4. Summary

Feature-selection methods (cf. Section 6.4.1) Among the feature-selection methods, *Model Gain* yielded the best average prediction performance. The simple univariate *MI* also turned out competitive, while *Greedy Wrapper* and *mRMR* required high optimization times, and our constraint-based version of *FCBF* yielded many infeasible solutions. Selecting $k = 10$ instead of $k = 5$ features had only a small impact on prediction performance, so users may stick to smaller feature-set sizes if such a setting benefits interpretability.

Search methods for alternatives (cf. Section 6.4.2) Solver-based simultaneous search, particularly with min-aggregation, considerably reduced the variance of the training-set objective value over alternatives compared to solver-based sequential search, as we desired. However, results were less clear on the test set and when using prediction performance to measure feature-set quality. Further, the average quality of alternatives was similar to solver-based sequential search. In addition, the latter was considerably faster and led to fewer solver timeouts, particularly when increasing the number of alternatives. Also, sequential search allows users to stop searching after each alternative.

The heuristic search methods *Greedy Replacement* and *Greedy Balancing* for univariate feature qualities achieved a good feature-set quality relative to solver-based search, particularly for a low number of alternatives and on the test set. For a high number of alternatives, the training feature-set quality may differ more and the heuristics may stop early despite the existence of further alternatives. As a positive point, both the heuristics' runtime was negligible. Also, *Greedy Balancing* achieved a low variance of training-set objective value between alternatives, similar to solver-based simultaneous search with min-aggregation.

User parameters a and τ (cf. Section 6.4.3) Feature-set quality tended to decrease with a higher number of alternatives a and dissimilarity threshold τ, so these parameters give users control over alternatives. The decrease was highest from the original feature set to the first alternative but smaller beyond, resulting in multiple alternatives of similar quality. Also, the decrease was more prominent on the training set than on the test set. Further, the strength of this decrease depended on the feature-selection method; *MI* and *Model Gain* showed the largest effect. Independent from the feature-selection method, the frequency of infeasible solutions increased with a and τ due to stronger constraints.

7. Discovering Sparse and Alternative Subgroup Descriptions

7.1. Overview

Scope In this chapter, we move to the field of subgroup discovery (cf. Definition 3). Building on our work regarding constrained feature selection (cf. Chapter 4), we investigate constraints on features in subgroup descriptions: We limit the number of selected features and search for alternative solutions, adapting the approach from Chapter 6. Both these constraint types make subgroup discovery more user-centric.

Contributions Our contribution in this chapter is fivefold:

(1) We formalize subgroup discovery as a Satisfiability Modulo Theories (SMT) optimization problem. This novel white-box formulation admits a solver-based search for subgroups and allows integrating constraints in a declarative manner.

(2) We formalize two constraint types for this optimization problem: feature-cardinality constraints and alternative subgroup descriptions. *Feature-cardinality constraints* limit the number of selected features, i.e., features used in subgroup descriptions. Thus, the subgroup descriptions become *sparse*, which increases their interpretability at the potential expense of subgroup quality. *Alternative subgroup descriptions* should use different features than a given subgroup description but cover a similar set of data objects. We allow users to control alternatives with two parameters, i.e., the number of alternatives and a dissimilarity threshold. We integrate both constraint types into our white-box problem formulation.

(3) We describe how to integrate these two constraint types into three existing heuristic search methods and two novel baselines for subgroup discovery. The latter are faster and simpler than the former, so they may serve as additional reference points for future experimental studies on subgroup discovery.

(4) We analyze the time complexity of the subgroup-discovery problem with each of these two constraint types and prove several $\mathcal{NP}$-completeness results.

(5) We conduct comprehensive experiments with 27 binary-classification datasets from the Penn Machine Learning Benchmarks (PMLB) [173, 186]. We compare solver-based and heuristic subgroup-discovery methods in different experimental scenarios: without constraints, with a feature-cardinality constraint, and for searching alternative subgroup

Algorithm 9: *MORS* for subgroup discovery.

Input: Dataset $X \in \mathbb{R}^{m \times n}$,
Prediction target $y \in \{0,1\}^m$
Output: Subgroup bounds $lb, ub \in \{\mathbb{R} \cup \{-\infty, +\infty\}\}^n$

for $j \leftarrow 1$ **to** n **do**
  $lb_j \leftarrow \min_{\substack{i \in \{1,\dots,m\} \\ y_i = 1}} X_{ij}$
  $ub_j \leftarrow \max_{\substack{i \in \{1,\dots,m\} \\ y_i = 1}} X_{ij}$
  if $lb_j = \min_{i \in \{1,\dots,m\}} X_{ij}$ **then** $lb_j \leftarrow -\infty$
  if $ub_j = \max_{i \in \{1,\dots,m\}} X_{ij}$ **then** $ub_j \leftarrow +\infty$
for $j \notin$ *get_permissible_feature_idxs(…)* **do**
  $(lb_j,\ ub_j) \leftarrow (-\infty, +\infty)$
return lb, ub

descriptions. In particular, we evaluate the runtime of subgroup discovery and the quality of the discovered subgroups. We also analyze how the subgroup quality in solver-based search depends on the timeout of the solver. Section 7.5.5 summarizes key results.

Materials We publish all our code and experimental data online (cf. Section 1.4).

Prior works The content of this chapter bases on the following prior work:

- Jakob Bach. *Using Constraints to Discover Sparse and Alternative Subgroup Descriptions.* arXiv:2406.01411v1 [cs.LG]. 2024. DOI: 10.48550/arXiv.2406.01411

Chapter outline The remainder of this chapter is structured as follows: Section 7.2 introduces two baselines for subgroup discovery. Section 7.3 describes and analyzes constrained subgroup discovery. Section 7.4 outlines our experimental design. Section 7.5 presents the corresponding experimental results.

7.2. Baselines

In this section, we propose and analyze two baselines for subgroup discovery, *MORS* (cf. Section 7.2.1) and *Random Search* (cf. Section 7.2.2). They are conceptually simpler than existing heuristic search methods (cf. Section 2.2.2) and serve as further reference points in our experiments. While they technically also are heuristics, we use the term *baselines* to refer to these two methods specifically.

7.2.1. MORS

This baseline builds on the following definition:

Definition 10 (Minimal Optimal-Recall Subgroup (MORS))**.** *Given a dataset $X \in \mathbb{R}^{m \times n}$ with prediction target $y \in \{0,1\}^m$, the* Minimal Optimal-Recall Subgroup (MORS) *is the subgroup (cf. Definition 2) whose lower and upper bounds of each feature correspond to the minimum and maximum value of that feature over all positive data objects (i.e., with $y_i = 1$) from the dataset X.*

The definition ensures that all positive data objects are contained in the subgroup. Thus, the evaluation metric *recall*, i.e., the fraction of positive data objects becoming subgroup members, reaches its *optimal* value of 1. At the same time, raising the lower bounds or lowering the upper bounds would exclude positive data objects from the subgroup. In this sense, the set of subgroup members is *minimal*. The corresponding subgroup description is unique and solves the following variant of the subgroup-discovery problem:

Definition 11 (Minimal-optimal-recall-subgroup discovery)**.** *Given a dataset $X \in \mathbb{R}^{m \times n}$ with prediction target $y \in \{0,1\}^m$,* minimal-optimal-recall-subgroup discovery *is the problem of finding a subgroup (cf. Definition 2) that contains as few negative data objects (i.e., with $y_i = 0$) as possible but all positive data objects (i.e., with $y_i = 1$) from the dataset X.*

I.e., the problem minimizes the number of false positives subject to producing no false negatives. Equivalently, it maximizes the number of true negatives, i.e., negative data objects excluded from the subgroup, subject to producing no false negatives.

Algorithm 9 outlines the procedure to determine the *MORS* bounds. Slightly deviating from Definition 10, but consistent to Algorithm 1, *MORS* replaces all non-excluding bounds with infinity (Lines 4–5). Further, if only certain features are permissible to be bounded, as we discuss later, we reset the bounds of the remaining features (Lines 6–7).

Since *MORS* only needs to iterate over all data objects and features once to determine the minima and maxima, the time complexity of this algorithm is $O(m \cdot n)$. This places minimal-optimal-recall-subgroup discovery in the complexity class $\mathcal{P}$:

Proposition 20 (Complexity of minimal-optimal-recall-subgroup discovery)**.** *The problem of minimal-optimal-recall-subgroup discovery (cf. Definition 11) can be solved in $O(m \cdot n)$.*

Interestingly, a kind of inverted problem definition, the Maximum Box problem, is $\mathcal{NP}$-hard [55]. The latter problem maximizes the number of true positives, i.e., positive data objects in the subgroup, subject to no false positives, i.e., optimal precision.

For complexity proofs later in this dissertation, we define another variant of the subgroup-discovery problem based on another particular type of subgroups [146]:

Definition 12 (Perfect subgroup)**.** *Given a dataset $X \in \mathbb{R}^{m \times n}$ with prediction target $y \in \{0,1\}^m$, a* perfect subgroup *is a subgroup (cf. Definition 2) that contains all positive data objects (i.e., with $y_i = 1$) but no negative data objects (i.e., with $y_i = 0$) from the dataset X.*

Algorithm 10: *Random Search* for subgroup discovery.

Input: Dataset $X \in \mathbb{R}^{m \times n}$,
Prediction target $y \in \{0,1\}^m$,
Subgroup-quality function $Q(lb, ub, X, y)$,
Number of iterations $n\_iters \in \mathbb{N}$
Output: Subgroup bounds $lb, ub \in \{\mathbb{R} \cup \{-\infty, +\infty\}\}^n$

$Q^{\text{opt}} \leftarrow -\infty$
for *iters* $\leftarrow 1$ **to** *n_iters* **do**
  for $j \leftarrow 1$ **to** n **do**
    $(lb_j,\ ub_j) \leftarrow (-\infty, +\infty)$
  for $j \in$ *get_permissible_feature_idxs(...)* **do**
    $(lb_j,\ ub_j) \leftarrow$ sample_uniformly(unique($X_{\cdot j}$))
  if $Q(lb, ub, X, y) > Q^{opt}$ **then**
    $Q^{\text{opt}} \leftarrow Q(lb, ub, X, y)$
    $(lb^{\text{opt}},\ ub^{\text{opt}}) \leftarrow (lb,\ ub)$
for $j \leftarrow 1$ **to** n **do**
  if $lb_j^{opt} = \min_{i \in \{1,\ldots,m\}} X_{ij}$ **then** $lb_j^{\text{opt}} \leftarrow -\infty$
  if $ub_j^{opt} = \max_{i \in \{1,\ldots,m\}} X_{ij}$ **then** $ub_j^{\text{opt}} \leftarrow +\infty$
return $lb^{opt},\ ub^{opt}$

Perfect subgroups reach the theoretical maximum WRAcc (cf. Equation 2.4) but do not exist in all datasets. Next, we define a corresponding search problem:

Definition 13 (Perfect-subgroup discovery)**.** *Given a dataset $X \in \mathbb{R}^{m \times n}$ with prediction target $y \in \{0,1\}^m$,* perfect-subgroup discovery *is the problem of finding a perfect subgroup (cf. Definition 12) if it exists or determining that it does not exist.*

Since *MORS* helps solving this problem in $O(m \cdot n)$, we obtain the following result:

Proposition 21 (Complexity of perfect-subgroup discovery)**.** *The problem of perfect-subgroup discovery (cf. Definition 13) can be solved in $O(m \cdot n)$.*

In particular, after *MORS* (cf. Algorithm 9) has found a subgroup, one only needs to check whether it contains any negative data objects. If it does not, it is perfect, otherwise no perfect subgroup exists. In particular, the *MORS* bounds cannot be tightened to exclude negative data objects without also excluding positives, thereby violating perfection.

7.2.2. Random Search

Algorithm 10 outlines a randomized search procedure that constitutes our second baseline. *Random Search* repeatedly generates and evaluates subgroups for a user-defined number

of iterations *n_iters* $\in \mathbb{N}$. Each generation step samples a lower bound and an upper bound uniformly random from the unique values for each permissible feature, leaving the remaining features unrestricted (Lines 3–6). The algorithm tracks the best generated subgroup so far over the iterations (Lines 7–9) and finally returns it. As for Algorithm 1, *Random Search* replaces all non-excluding bounds with infinity (Lines 10–12).

7.3. Constrained Subgroup Discovery

In this section, we discuss subgroup discovery with constraints. First, we frame subgroup discovery as an SMT optimization problem (cf. Section 7.3.1). Second, we formalize and analyze feature-cardinality constraints (cf. Section 7.3.2). Third, we formalize and analyze alternative subgroup descriptions (cf. Section 7.3.3).

7.3.1. SMT Encoding of Subgroup Discovery

To find optimal subgroups exactly, one can encode subgroup discovery as a white-box optimization problem and employ a solver. As for constrained feature selection, this approach allows directly integrating and combining a variety of constraint types in a declarative manner. Similar to Section 4.2, we propose an SMT encoding with linear real arithmetic (LRA) [24]. This formulation is straightforward given the problem definition (cf. Definition 3). Appendix A.2.1 describes an encoding as a mixed integer linear program (MILP), which performed worse in preliminary experiments. Beyond the scope of this dissertation, [18] outlines a MaxSAT encoding and how to handle categorical features.

The optimization problem consists of an objective function and constraints.

Objective function We use WRAcc as the objective, which should be maximized. In the formula for WRAcc (cf. Equation 2.3), m and m^+ are constants, while m_b and m_b^+ depend on the decision variables. The previously provided formula contains m_b in the numerator and denominator, but one can reformulate it by multiplying its two factors:

$$\text{WRAcc} = \frac{m_b^+}{m} - \frac{m_b \cdot m^+}{m^2} = \frac{m_b^+ \cdot m - m_b \cdot m^+}{m^2} \tag{7.1}$$

In this new expression, the denominators and the factor m^+ in the numerator are constant. Thus, the whole expression is linear in m_b^+ and m_b. We define these two quantities as linear expressions from binary decision variables $b \in \{0,1\}^m$ that denote subgroup membership. I.e., b_i expresses whether the i-th data object is in the subgroup.

$$\begin{aligned} m_b &:= \sum_{i=1}^{m} b_i \\ m_b^+ &:= \sum_{\substack{i \in \{1,\ldots,m\} \\ y_i = 1}} b_i \end{aligned} \tag{7.2}$$

Since the values of the target variable y are fixed, the expression for m_b^+ only sums over the positive data objects. Further, one may define m_b^+ and m_b as separate integer variables or directly insert their expressions into Equation 7.1. We chose the latter in our implementation and therefore wrote ':=' in Equation 7.2 instead of using a propositional operator like '↔'. Finally, the formula for nWRAcc (cf. Equation 2.5) is linear as well, having the same enumerator as Equation 7.1 and a different constant in the denominator.

Constraints The subgroup membership b_i of a data object depends on the bounds of the subgroup (cf. Definition 2). Thus, we define real-valued decision variables $lb, ub \in \{\mathbb{R} \cup \{-\infty, +\infty\}\}^n$ for the latter. In particular, each feature has a lower bound and an upper bound. The upper bounds naturally need to be at least as high as the lower bounds:

$$\forall j \in \{1, \ldots, n\} : \; lb_j \leq ub_j \tag{7.3}$$

A data object is a member of the subgroup if all its feature values are within the bounds:

$$\forall i \in \{1, \ldots, m\} : \; b_i \leftrightarrow \bigwedge_{j \in \{1,\ldots,n\}} \left(\left(X_{ij} \geq lb_j\right) \wedge \left(X_{ij} \leq ub_j\right) \right) \tag{7.4}$$

Instead of defining separate decision variables b_i and binding them to the bounds with an equivalence constraint, one could also insert the corresponding Boolean expression into the right-hand-side of Equation 7.2 directly. In particular, lb_j and ub_j are the only decision variables strictly necessary for the optimization problem. However, some constraint types on subgroups may use b_i as well, so it is helpful to have it defined explicitly.

Complete optimization problem Combining all prior definitions of decision variables, constraints, and the objective, we obtain the following SMT optimization problem:

$$\begin{aligned}
\max \quad & Q_{\text{WRAcc}} = \frac{m_b^+}{m} - \frac{m_b \cdot m^+}{m^2} \\
\text{s.t.:} \quad & m_b := \sum_{i=1}^{m} b_i \\
& m_b^+ := \sum_{\substack{i \in \{1,\ldots,m\} \\ y_i = 1}} b_i \\
\forall i \in \{1, \ldots, m\} \quad & b_i \leftrightarrow \bigwedge_{j \in \{1,\ldots,n\}} \left(\left(X_{ij} \geq lb_j\right) \wedge \left(X_{ij} \leq ub_j\right) \right) \\
\forall j \in \{1, \ldots, n\} \quad & lb_j \leq ub_j \\
& b \in \{0,1\}^m \\
& lb, ub \in \{\mathbb{R} \cup \{-\infty, +\infty\}\}^n
\end{aligned} \tag{7.5}$$

We refer to this optimization problem as *unconstrained subgroup discovery* in the following since it only contains constraints to define subgroup discovery per se but not the additional constraints on feature sets that we will impose later.

Post-processing In our implementation, we add a small post-processing step. In particular, we do not use the solver-determined values of the variables lb_j and ub_j but the minimum and maximum feature values of all data objects in the subgroup (i.e., with $b_i = 1$). Thus, we ensure that the bounds correspond to actual feature values, consistent with the bounds returned by heuristic search methods and baselines. Further, this choice avoids potential numerical issues caused by extracting floating-point values from the solver. If the subgroup does not contain any data objects, we use invalid bounds (i.e., $ub_j = -\infty < \infty = lb_j$) to ensure that the subgroup remains empty even for arbitrary new data objects.

7.3.2. Feature-Cardinality Constraints

In this section, we discuss feature-cardinality constraints for subgroup discovery. First, we motivate and formalize them (cf. Section 7.3.2.1). Next, we describe how to integrate them into our SMT encoding of subgroup discovery (cf. Section 7.3.2.2), heuristic search methods (cf. Section 7.3.2.3), and baselines (cf. Section 7.3.2.4). Finally, we analyze the time complexity of subgroup discovery with this constraint type (cf. Section 7.3.2.5).

7.3.2.1. Concept

Feature-cardinality constraints limit the number of features used in the subgroup description, rendering the latter less complex and easier to interpret [147]. To formalize this constraint type, we define feature selection in subgroup discovery as follows:

Definition 14 (Feature selection in subgroups)**.** *Given a dataset $X \in \mathbb{R}^{m \times n}$ and a subgroup (cf. Definition 2) with bounds $lb, ub \in \{\mathbb{R} \cup \{-\infty, +\infty\}\}^n$, Feature j is* selected *if the bounds exclude at least one data object of X from the subgroup, i.e., $\exists i \in \{1, \ldots, m\} : \big(X_{ij} < lb_j\big) \vee \big(X_{ij} > ub_j\big)$.*

The bounds of unselected features can be considered infinite, effectively removing these features from the subgroup description. The *feature cardinality* of the subgroup is the number of selected features. Related work (cf. Section 3.1.2) also uses the terms *depth* [147] or *length* [8, 88], though partly referring to the number of conditions in the subgroup description rather than selected features. I.e., if there is a lower and an upper bound for a feature, some related work counts this feature twice instead of once.

Feature-cardinality constraints employ a user-defined limit on feature cardinality:

Definition 15 (Feature-cardinality constraint)**.** *Given a cardinality threshold $k \in \mathbb{N}$, a* feature-cardinality constraint *for a subgroup (cf. Definition 2) requires the subgroup to have at most k features selected (cf. Definition 14).*

In some scenarios, selecting less than exactly k features may improve subgroup quality. Thus, we use an inequality rather than the equality (cf. Equation 4.3) we employed in our work on alternative feature selection (cf. Chapter 6).

7.3.2.2. SMT Encoding

We first need to encode whether a feature is selected or not. Thus, we introduce binary decision variables $s, s^{\text{lb}}, s^{\text{ub}} \in \{0,1\}^n$. A feature is selected if its bounds exclude at least one data object from the subgroup (cf. Definition 14), i.e., the lower bound is higher than the minimum feature value or the upper bound is lower than the maximum feature value:

$$\begin{aligned} \forall j: \quad & s_j^{\text{lb}} \leftrightarrow \left(lb_j > \min_{i \in \{1,\ldots,m\}} X_{ij} \right) \\ \forall j: \quad & s_j^{\text{ub}} \leftrightarrow \left(ub_j < \max_{i \in \{1,\ldots,m\}} X_{ij} \right) \\ \forall j: \quad & s_j \leftrightarrow \left(s_j^{\text{lb}} \vee s_j^{\text{ub}} \right) \\ \text{with index:} \quad & j \in \{1,\ldots,n\} \end{aligned} \tag{7.6}$$

In this equation, minimum and maximum feature values are constants that can be determined before formulating the optimization problem. Given the definition of s_j, setting an upper bound on the number of selected features (cf. Definition 15) is straightforward:

$$\sum_{j=1}^{n} s_j \leq k \tag{7.7}$$

Instead of defining the decision variables s_j, s_j^{lb}, and s_j^{ub}, one could also insert the corresponding expressions into Equation 7.7 directly. However, we will also use s_j for alternative subgroup descriptions (cf. Section 7.3.3.2), so we define corresponding variables.

The overall SMT encoding of subgroup discovery with a feature-cardinality constraint is the SMT encoding of unconstrained subgroup discovery (cf. Equation 7.5) supplemented by the variables and constraints from Equations 7.6 and 7.7. Our implementation also contains a post-processing step that sets non-excluding lower bounds (i.e., with $s_j^{\text{lb}} = 0$) to $-\infty$ and non-excluding upper bounds (i.e., with $s_j^{\text{ub}} = 0$) to $+\infty$. This step is consistent with the heuristic search methods and baselines (e.g., cf. Lines 21–23 in Algorithm 1).

7.3.2.3. Integration into Heuristic Search Methods

The feature-cardinality constraint (cf. Definition 15) is *antimonotonic* [163] in the feature selection: If a selected feature set satisfies the constraint, all its subsets also satisfy it. Vice versa, if a feature set violates the constraint, all its supersets also violate it. This property allows to easily integrate the constraint into the three heuristic search methods from Section 2.2.2, i.e., *PRIM* (cf. Algorithm 1), *Beam Search* (cf. Algorithms 2 and 3), and *Best Interval* (cf. Algorithms 2 and 4), which all iteratively enlarge the set of selected features. In particular, these search methods start with unrestricted subgroup bounds, i.e., an empty feature set, which satisfies the constraint for any $k \geq 0$. Each iteration may either add bounds on one further feature or refine the bounds on an already selected feature.

Thus, one can prevent generation of invalid solution candidates by defining the function *get_permissible_feature_idxs(…)* (cf. Line 7 in Algorithm 1 and Line 12 in Algorithm 2) as follows: If already k features are selected, only these features are permissible. If less than k features are selected, all features are permissible, as in the unconstrained search.

7.3.2.4. Integration into Baselines

MORS *MORS* calls the function *get_permissible_feature_idxs(…)* in Line 6 of Algorithm 9. To instantiate this function, we employ a univariate, quality-based heuristic for feature selection: For each feature, we evaluate what would happen if only this feature was restricted according to *MORS* (cf. Definition 10). In particular, we determine the number of false positives, i.e., negative data objects in the subgroup, defined by each feature's *MORS* bounds (Lines 2–3). We select the k features with the lowest number of false positives.

This heuristic is equivalent to selecting the features with the highest WRAcc for univariate *MORS* bounds: Due to *MORS*, not only m and m^+ are constant but also the number of positive subgroup members m_b^+, which equals m^+. Thus, one can rephrase Equation 2.3:

$$\mathrm{WRAcc}_{\mathrm{MORS}} = \frac{m_b^+}{m} - \frac{m_b \cdot m^+}{m^2} = \frac{m^+}{m} - \frac{m_b \cdot m^+}{m^2} = \frac{m^+}{m} \cdot \left(1 - \frac{m_b}{m}\right) \tag{7.8}$$

Thus, maximizing WRAcc corresponds to minimizing m_b/m, i.e., the relative frequency of the data objects in the subgroup. Since the number of positive data objects in the subgroup is fixed in *MORS*, this objective amounts to including as few negative data objects as possible in the subgroup, i.e., minimizing the number of false positives like our heuristic.

Random Search *Random Search* calls *get_permissible_feature_idxs(…)* in Line 5 of Algorithm 10). To observe a cardinality threshold k, we simply sample k out of n features uniformly random without replacement. The bounds for these features will be restricted in the next step of the algorithm, while all remaining features remain unrestricted.

7.3.2.5. Time Complexity

We analyze three aspects of time complexity: the size of the search space for exhaustive search, parameterized complexity, and $\mathcal{NP}$-hardness.

Exhaustive search Before addressing feature-cardinality constraints, we analyze the unconstrained case. In general, the search space of subgroup discovery depends on the number of candidates for lower and upper bounds. With m data objects, each real-valued feature may have up to m unique values. It suffices to treat these unique values as bound candidates since any bounds between feature values or outside the feature's range do not change the subgroup membership during optimization. Thus, there are $O(m^2)$ relevant lower-upper-bound combinations per feature. Since we need to combine bounds over all n features, the size of the search space is $O(m^{2n})$:

Proposition 22 (Complexity of exhaustive search for subgroup discovery)**.** *An exhaustive search for subgroup discovery (cf. Definition 3) needs to evaluate $O(m^{2n})$ subgroups.*

For each of these candidate subgroups, the cost of evaluating a quality function like WRAcc (cf. Equation 2.3) typically is $O(m \cdot n)$, i.e., requires a constant number of passes over the dataset and therefore has linear complexity in the dataset size. Further, the number of potential subgroups only is an upper bound: Efficient exhaustive search methods employ quality-based pruning to not evaluate all solution candidates explicitly [8].

Next, we adapt the result from Proposition 22 to feature-cardinality constraints. There are $\binom{n}{k} \leq n^k$ feature sets of size k with $O(m^{2k})$ bound candidates each:

Proposition 23 (Complexity of exhaustive search for subgroup discovery with feature–cardinality constraint)**.** *An exhaustive search for subgroup discovery (cf. Definition 3) with a feature-cardinality constraint (cf. Definition 15) needs to evaluate $O(n^k \cdot m^{2k})$ subgroups.*

For the special case $k = 1$, the size of the search space becomes $O(n \cdot m^2)$, which is implicitly leveraged by heuristic search methods that only consider updating the bounds of each feature separately instead of jointly (cf. Section 2.2.2).

Parameterized complexity If we consider the number of features n to be a small constant, the search-space size of unconstrained subgroup discovery (cf. Proposition 22) is polynomial in m and thereby also polynomial in the dataset size $m \cdot n$. Thus, the problem of subgroup discovery belongs to the parameterized complexity class $\mathcal{XP}$ [54]:

Proposition 24 (Parameterized complexity of subgroup discovery)**.** *The problem of subgroup discovery (cf. Definition 3) resides in the parameterized complexity class $\mathcal{XP}$ for the parameter n.*

In contrast, the complexity remains exponential in n if m is fixed. I.e., the number of features has an exponential impact on the size of the search space, while the number of data objects has a polynomial impact. With a feature-cardinality constraint, the problem retains its $\mathcal{XP}$ membership. Considering the size of the search space from Proposition 23, the parameter is k instead of n now:

Proposition 25 (Parameterized complexity of subgroup discovery with feature-cardinality constraint)**.** *The problem of subgroup discovery (cf. Definition 3) with a feature-cardinality constraint (cf. Definition 15) resides in the parameterized complexity class $\mathcal{XP}$ for the parameter k.*

NP-Hardness [29] showed $\mathcal{NP}$-hardness for finding a subgroup description with minimum feature cardinality that replicates the membership of a given subgroup. We transfer this result to optimizing subgroup quality under a feature-cardinality constraint. First, we tackle the search problem for perfect subgroups (cf. Appendix A.2.2.1 for the proof):

Proposition 26 (Complexity of perfect-subgroup discovery with feature-cardinality constraint)**.** *The problem of perfect-subgroup discovery (cf. Definition 13) with a feature-cardinality constraint (cf. Definition 15) is $\mathcal{NP}$-complete.*

This hardness result contrasts with the polynomial runtime of unconstrained perfect-subgroup discovery (cf. Proposition 21), which corresponds to a cardinality constraint with $k = n$. Generalizing Proposition 26, the optimization problem of subgroup discovery with a feature-cardinality constraint is $\mathcal{NP}$-complete under a reasonable assumption on the notion of subgroup quality (cf. Appendix A.2.2.2 for the proof):

Proposition 27 (Complexity of subgroup discovery with feature-cardinality constraint)**.** *Assuming a subgroup-quality function $Q(lb, ub, X, y)$ for which only perfect subgroups (cf. Definition 12) reach its maximal value, the problem of subgroup discovery (cf. Definition 3) with a feature-cardinality constraint (cf. Definition 15) is $\mathcal{NP}$-complete.*

WRAcc as the subgroup-quality function satisfies this assumption since only perfect subgroups yield the theoretical maximum WRAcc (cf. Equation 2.4).

7.3.3. Alternative Subgroup Descriptions

In this section, we propose the optimization problem of discovering alternative subgroup descriptions. First, we motivate and formalize the problem (cf. Section 7.3.3.1). Next, we describe how to phrase it within our SMT encoding of subgroup discovery (cf. Section 7.3.3.2), heuristic search methods (cf. Section 7.3.3.3), and baselines (cf. Section 7.3.3.4). Finally, we analyze the time complexity of this problem (cf. Section 7.3.3.5).

7.3.3.1. Concept

Overview For alternative subgroup descriptions, we assume to have an *original subgroup* given, with subgroup membership $b^{(0)} \in \{0,1\}^m$ of data objects and with feature selection $s^{(0)} \in \{0,1\}^n$. When searching alternatives, we do not optimize subgroup quality but the similarity to this original subgroup. We express the similarity in terms of subgroup membership. If this similarity is very high, then the subgroup quality should also be similar since evaluation metrics for subgroup quality typically base on subgroup membership.

Additionally, we constrain the new subgroup description to be alternative enough. We express this dissimilarity in terms of the subgroups' feature selection. The user chooses a dissimilarity threshold $\tau \in \mathbb{R}_{\geq 0}$ and can thereby control alternatives. Further, we recommend employing a feature-cardinality constraint (cf. Definition 15) when determining the original subgroup so there are sufficiently many features left for the alternative. The alternative may be feature-cardinality-constrained as well, to increase its interpretability.

In a nutshell, alternative subgroup descriptions should produce similar subgroup membership as the original subgroup but use different features.

Sequential search One can search for multiple alternative subgroup descriptions sequentially, as for alternative feature sets (cf. Section 6.2.2.2). After determining the original subgroup, each iteration yields one further alternative. Users may prescribe a number of alternatives $a \in \mathbb{N}$ a priori or stop the procedure whenever the alternatives are not interesting anymore, e.g., too dissimilar to the original subgroup. Each alternative should have a similar subgroup membership as the original subgroup but a dissimilar feature selection compared to all *existing subgroups* found in prior iterations:

Definition 16 (Alternative-subgroup-description discovery)**.** *Given*

- *a dataset* $X \in \mathbb{R}^{m \times n}$,
- $a \in \mathbb{N}$ *existing subgroups with subgroup membership* $b^{(l)} \in \{0,1\}^m$ *and feature selection* $s^{(l)} \in \{0,1\}^n$ *for* $l \in \{0, \dots, a-1\}$,
- *a similarity measure* $sim(\cdot)$ *for subgroup-membership vectors,*
- *a dissimilarity measure* $dis(\cdot)$ *for feature-selection vectors of subgroups,*
- *and a dissimilarity threshold* $\tau \in \mathbb{R}_{\geq 0}$,

alternative-subgroup-description discovery *is the problem of finding a subgroup (cf. Definition 2) with membership* $b^{(a)} \in \{0,1\}^m$ *and feature selection* $s^{(a)} \in \{0,1\}^n$ *that maximizes the subgroup-membership similarity* $sim(b^{(a)}, b^{(0)})$ *to the original subgroup while being dissimilar to all existing subgroups regarding the feature selection, i.e.,* $\forall l \in \{0, \dots, a-1\} : dis(s^{(a)}, s^{(l)}) \geq \tau$.

Compared to the conventional subgroup-discovery problem (cf. Definition 3), $b^{(0)}$ replaces the prediction target y, and $\mathrm{sim}(\cdot)$ replaces the subgroup-quality function $Q(lb, ub, X, y)$. In the following, we discuss $\mathrm{sim}(\cdot)$ and $\mathrm{dis}(\cdot)$.

Similarity in objective function There are various options to quantify the similarity of subgroup membership, i.e., between two binary vectors. For example, the Hamming distance counts how many vector entries differ [43]. We turn this distance into a similarity measure by counting identical vector entries. Further, we normalize the similarity with the vector length, i.e., number of data objects m, to obtain the following *normalized Hamming similarity* for two subgroup-membership vectors $b', b'' \in \{0,1\}^m$:

$$\mathrm{sim}_{\mathrm{nHamm}}(b', b'') = \frac{1}{m} \cdot \sum_{i=1}^{m} (b'_i = b''_i) \tag{7.9}$$

If either b' or b'' is constant, this similarity measure is linear in its remaining argument, as discussed later (cf. Equation 7.12). Further, if one considers one vector to be a prediction and the other to be the ground truth, Equation 7.9 equals prediction accuracy for classification. Another popular similarity measure for sets or binary vectors is the Jaccard index (cf. Equation 6.2), which relates the intersection of positive vector entries to their union:

$$\mathrm{sim}_{\mathrm{Jacc}}(b', b'') = \frac{\sum_{i=1}^{m} (b'_i \wedge b''_i)}{\sum_{i=1}^{m} (b'_i \vee b''_i)} \tag{7.10}$$

However, this similarity measure is not linear in b' and b'', which prevents its use in certain white-box solvers. Thus, we use the normalized Hamming similarity.

Dissimilarity in constraints For the dissimilarity between feature-selection vectors, we employ the following *deselection dissimilarity* with an adapted dissimilarity threshold:

$$\mathrm{dis}_{\mathrm{des}}(s^{\mathrm{new}}, s^{\mathrm{old}}) = \sum_{j=1}^{n} (\neg s_j^{\mathrm{new}} \wedge s_j^{\mathrm{old}}) \geq \min\left(\tau_{\mathrm{abs}},\ k^{\mathrm{old}}\right) \tag{7.11}$$

This dissimilarity counts how many previously selected features are *not* selected in the new subgroup description. These features may either be replaced by other features, or the total number of selected features may be reduced. The constraint ensures that at least $\tau_{\mathrm{abs}} \in \mathbb{N}_0$ features are deselected but never more than there were selected before (k^{old}), which would be infeasible. For maximum dissimilarity, none of the previously selected features may be selected again. This dissimilarity measure is asymmetric, i.e., $\mathrm{dis}_{\mathrm{des}}(s^{\mathrm{new}}, s^{\mathrm{old}}) \neq \mathrm{dis}_{\mathrm{des}}(s^{\mathrm{old}}, s^{\mathrm{new}})$, which would be an issue in a simultaneous search for alternatives but is acceptable in sequential search, where 'old' and 'new' are well-defined.

Compared to more common dissimilarity measures like the Jaccard distance or the Dice dissimilarity (cf. Equation 6.3), our deselection dissimilarity has two advantages: First, if s^{old} is constant, the dissimilarity is a simple sum and therefore linear in s^{new}, even if the number of newly selected features is unknown yet, while Jaccard distance and Dice dissimilarity involve a ratio in the latter case. This property is useful for solver-based search (cf. Section 7.3.3.2). Second, Equation 7.11 is antimonotonic in the new feature selection, which is useful for heuristic search (cf. Section 7.3.3.3). Using the Jaccard distance or Dice dissimilarity in the constraint violates this property. In particular, these dissimilarities can increase by selecting features that were not selected before, i.e., an invalid feature set can become valid instead of remaining invalid by selecting further features.

7.3.3.2. SMT Encoding

Slightly reformulating Equation 7.9 yields a linear objective function regarding $b^{(a)}$:

$$\begin{aligned} \mathrm{sim}_{\mathrm{nHamm}}(b^{(a)}, b^{(0)}) &= \frac{1}{m} \cdot \sum_{i=1}^{m} \left(b_i^{(a)} \leftrightarrow b_i^{(0)}\right) \\ &= \frac{1}{m} \cdot \Bigg(\sum_{\substack{i \in \{1,\ldots,m\} \\ b_i^{(0)}=1}} b_i^{(a)} + \sum_{\substack{i \in \{1,\ldots,m\} \\ b_i^{(0)}=0}} \neg b_i^{(a)} \Bigg) \end{aligned} \tag{7.12}$$

In particular, since $b^{(0)}$ is known and therefore constant, we employ the expression from the second line, i.e., without the logical equivalence operator. Instead, we compute two sums, one for data objects that are members of the original subgroup and one for non-members. The negated expression $\neg b_i^{(a)}$ may be expressed as $1 - b_i^{(a)}$.

To formulate the dissimilarity constraints, we leverage that the feature-selection vector $s^{(l)}$ and the corresponding number of selected features $k^{(l)}$ are known for all existing subgroups as well. Thus, we instantiate and adapt Equation 7.11 as follows:

$$\forall l \in \{0, \ldots, a-1\} : \ \mathrm{dis}_{\mathrm{des}}(s^{(a)}, s^{(l)}) = \sum_{\substack{j \in \{1,\ldots,n\} \\ s_j^{(l)}=1}} \neg s_j^{(a)} \geq \min\left(\tau_{\mathrm{abs}},\ k^{(l)}\right) \tag{7.13}$$

In particular, we only sum over features that were selected in the existing subgroup and check whether they are deselected now.

The overall SMT encoding of alternative-subgroup-description discovery (cf. Definition 16) combines the similarity objective (cf. Equation 7.12) and dissimilarity constraints (cf. Equation 7.13) with the previously introduced variables and constraints for bounds (cf. Equation 7.3), subgroup membership (cf. Equation 7.4), and feature selection (cf. Equation 7.6). Optionally, one may add a feature-cardinality constraint (cf. Equation 7.7).

7.3.3.3. Integration into Heuristic Search Methods

The situation is similar to integrating feature-cardinality constraints (cf. Section 7.3.2.3). In particular, the constraint for alternatives based on the deselection dissimilarity (cf. Equation 7.11) is antimonotonic. I.e., the constraint is satisfied for an empty feature set, and once it is violated for a feature set, it remains violated for any superset. Thus, the constraint type is suitable for heuristic search methods that iteratively enlarge the set of selected features, like *PRIM* (cf. Algorithm 1), *Beam Search* (cf. Algorithms 2 and 3), and *Best Interval* (cf. Algorithms 2 and 4). We only need to adapt the function *get_permissible_feature_idxs(…)* (cf. Line 7 in Algorithm 1 and Line 12 in Algorithm 2) to return the indices of all features that may be selected without violating the dissimilarity constraint (cf. Equation 7.11). In particular, once $k^{(l)} - \tau_{\mathrm{abs}}$ features from an existing subgroup with $k^{(l)}$ features are selected in the new subgroup, no further features from the former subgroup may be selected.

7.3.3.4. Integration into Baselines

Adapting our two baselines to alternative subgroup descriptions is less straightforward than to feature-cardinality constraints (cf. Section 7.3.2.4) since the optimization objective changes and the search space under the dissimilarity constraint is harder to describe. Thus, we did not implement concrete adaptations but still discuss possible ideas next.

MORS *MORS* (cf. Algorithm 9) is tailored to a particular objective, i.e., subgroup quality in terms of recall, which differs from the normalized Hamming similarity (cf. Equation 7.9) we use for alternative subgroup descriptions. For the dissimilarity constraint, we would like to enforce a valid feature set by implementing the function *get_permissible_feature_idxs(…)* in Line 6 of Algorithm 9 appropriately. The univariate, quality-based selection heuristic we proposed for feature-cardinality constraints (cf. Section 7.3.2.4) may produce an invalid solution. To alleviate this issue, we could adapt this heuristic as follows: Still order the features by their univariate quality and iteratively select them in this order, but check the dissimilarity constraint in each iteration and skip over features that violate it.

Random Search For *Random Search* (cf. Algorithm 10), changing the optimization objective is not an issue since the objective is treated as a black-box function for evaluating randomly generated subgroups (cf. Line 7 of Algorithm 10). For the dissimilarity constraint, we would like to implement *get_permissible_feature_idxs(…)* in Line 5 of Algorithm 10) by uniformly sampling from the constrained search space, which is a computationally hard problem in general [58]. We could also sample from the unconstrained space and then check the dissimilarity constraint, repeating sampling till a valid feature set is found. However, this strategy may produce a high fraction of invalid solution candidates. Another option would be using non-uniform sampling. E.g., we could only sample features not selected in any existing subgroup, which guarantees constraint satisfaction but does not cover the entire constrained search space since it ignores the allowed feature-set overlap.

7.3.3.5. Time Complexity

As for the feature-cardinality constraint (cf. Section 7.3.2.5), we analyze the size of the search space for exhaustive search, parameterized complexity, and $\mathcal{NP}$-hardness.

Exhaustive search The search for an alternative subgroup description can iterate over the same solution candidates as the search for an original subgroup description. Thus, the results from Propositions 22 and 23 remain valid:

Proposition 28 (Complexity of exhaustive search for alternative-subgroup-description discovery)**.** *An exhaustive search for alternative-subgroup-description discovery (cf. Definition 16) needs to evaluate $O(m^{2n})$ subgroups for each alternative in general or $O(n^k \cdot m^{2k})$ subgroups if a feature-cardinality constraint (cf. Definition 15) is employed.*

The evaluation of solution candidates differs from the original subgroup descriptions but has a similar time complexity, i.e., $O(m \cdot n + a \cdot n)$ instead of $O(m \cdot n)$. In particular, evaluating the similarity-based objective function (e.g., Equation 7.9) should typically have a cost of $O(m \cdot n)$, like subgroup-quality functions have. Checking the dissimilarity constraint (e.g., Equation 7.11) requires determining the selected features and computing the dissimilarity. The former (cf. Definition 14) runs in $O(n)$ if the minimum and maximum feature values of the dataset are precomputed. The latter should typically entail a cost of $O(n)$ per existing subgroup description for reasonably simple dissimilarity functions.

Parameterized complexity Due to the similar search space as for original subgroup descriptions, the parameterized-complexity results from Propositions 24 and 25 apply to discovering alternative subgroup descriptions as well:

Proposition 29 (Parameterized complexity of alternative-subgroup-description discovery)**.** *The problem of alternative-subgroup-description discovery (cf. Definition 16) resides in the parameterized complexity class $\mathcal{XP}$ for the parameter n in general and for the parameter k if a feature-cardinality constraint (cf. Definition 15) is employed.*

NP-Hardness We prove $\mathcal{NP}$-completeness for a special case of alternative-subgroup-description discovery (cf. Definition 16) first, leveraging the following definition:

Definition 17 (Perfect alternative subgroup description)**.** *Given*

- *a dataset* $X \in \mathbb{R}^{m \times n}$,
- *an original subgroup with subgroup membership* $b^{(0)} \in \{0,1\}^m$ *and feature selection* $s^{(0)} \in \{0,1\}^n$,
- *a dissimilarity measure* $dis(\cdot)$ *for feature-selection vectors of subgroups,*
- *and a dissimilarity threshold* $\tau \in \mathbb{R}_{\geq 0}$,

a perfect alternative subgroup description *defines a subgroup (cf. Definition 2) with membership* $b^{(a)} \in \{0,1\}^m$ *and feature selection* $s^{(a)} \in \{0,1\}^n$ *that exactly replicates the subgroup membership of the original subgroup, i.e.,* $b^{(a)} = b^{(0)}$*, while being dissimilar regarding the feature selection, i.e.,* $dis(s^{(a)}, s^{(0)}) \geq \tau$.

In particular, the subgroup-membership similarity is fixed here rather than an optimization objective. Similar to perfect subgroups (cf. Definition 12), perfect alternative subgroup descriptions only exist in some datasets. Next, we define a corresponding search problem:

Definition 18 (Perfect-alternative-subgroup-description discovery)**.** *Given*

- *a dataset* $X \in \mathbb{R}^{m \times n}$,
- *an original subgroup with subgroup membership* $b^{(0)} \in \{0,1\}^m$ *and feature selection* $s^{(0)} \in \{0,1\}^n$,
- *a dissimilarity measure* $dis(\cdot)$ *for feature-selection vectors of subgroups,*
- *and a dissimilarity threshold* $\tau \in \mathbb{R}_{\geq 0}$,

perfect-alternative-subgroup-description discovery *is the problem of finding a perfect alternative subgroup description (cf. Definition 17) if it exists or determining that it does not exist.*

We obtain the following hardness result for this search problem with a perfect original subgroup and under a reasonable assumption on the notion of feature-selection dissimilarity (cf. Appendix A.2.2.3 for the proof):

Proposition 30 (Complexity of perfect-alternative-subgroup-description discovery with feature-cardinality constraint and perfect original subgroup)**.** *Assuming*

- *a combination of a dissimilarity measure* $dis(\cdot)$ *and a dissimilarity threshold* $\tau \in \mathbb{R}_{\geq 0}$ *that prevents selecting any selected feature from the original subgroup description again,*
- *and the original subgroup being perfect (cf. Definition 12),*

the problem of perfect-alternative-subgroup-description discovery (cf. Definition 18) with a feature-cardinality constraint (cf. Definition 15) is $\mathcal{NP}$*-complete.*

Our deselection dissimilarity (cf. Equation 7.11) as $\mathrm{dis}(\cdot)$ satisfies the dissimilarity assumption if we choose a dissimilarity threshold $\tau_{\mathrm{abs}} \geq k^{\mathrm{old}}$. Other dissimilarity measures should typically also have a threshold value that enforces zero overlap between feature sets.

The problem naturally remains $\mathcal{NP}$-complete when dropping the two assumptions in Proposition 30. Nevertheless, we explicitly extend this result to imperfect original subgroups (cf. Appendix A.2.2.4 for the proof):

Proposition 31 (Complexity of perfect-alternative-subgroup-description discovery with feature-cardinality constraint and imperfect original subgroup)**.** *Assuming*

- *a combination of a dissimilarity measure* $\mathrm{dis}(\cdot)$ *and a dissimilarity threshold* $\tau \in \mathbb{R}_{\geq 0}$ *that prevents selecting any selected feature from the original subgroup description again,*
- *and the original subgroup* not *being perfect (cf. Definition 12),*

the problem of perfect-alternative-subgroup-description discovery (cf. Definition 18) with a feature-cardinality constraint (cf. Definition 15) is $\mathcal{NP}$*-complete.*

Finally, we switch from the search problem for perfect alternatives to the optimization problem. We establish $\mathcal{NP}$-completeness under a reasonable assumption on the notion of subgroup-membership similarity (cf. Appendix A.2.2.5 for the proof):

Proposition 32 (Complexity of alternative-subgroup-description discovery with feature–cardinality constraint)**.** *Assuming*

- *a combination of a dissimilarity measure* $\mathrm{dis}(\cdot)$ *and a dissimilarity threshold* $\tau \in \mathbb{R}_{\geq 0}$ *that prevents selecting any selected feature from the original subgroup description again,*
- *and a similarity measure* $\mathrm{sim}(\cdot)$ *for which only perfect alternative subgroup descriptions (cf. Definition 17) reach its maximal value regarding the original subgroup,*

the problem of alternative-subgroup-description discovery (cf. Definition 16) with a feature-cardinality constraint (cf. Definition 15) is $\mathcal{NP}$*-complete.*

Normalized Hamming similarity (cf. Equation 7.9) as $\mathrm{sim}(\cdot)$ satisfies the similarity assumption in Proposition 32 since only perfect alternative subgroup descriptions yield the theoretical maximum similarity of 1 to the original subgroup description.

7.4. Experimental Design

In this section, we introduce our experimental design for evaluating constrained subgroup discovery. After a brief overview (cf. Section 7.4.1), we describe the components of the experimental design in detail: subgroup-discovery methods (cf. Section 7.4.2), experimental scenarios (cf. Section 7.4.3), evaluation metrics (cf. Section 7.4.4), and datasets (cf. Section 7.4.5). Finally, we briefly outline our implementation (cf. Section 7.4.6).

7.4.1. Overview

We evaluate six subgroup-discovery methods on 27 binary-classification datasets. As evaluation metrics, we mainly consider subgroup quality in terms of nWRAcc and the methods' runtime. We analyze four *experimental scenarios*: First, we compare the subgroup-discovery methods without constraints. Second, we vary the timeout in solver-based search. Third, we introduce a feature-cardinality constraint and vary the cardinality threshold k. Fourth, we search for alternative subgroup descriptions and vary the number of alternatives a as well as the dissimilarity threshold τ_{abs}.

7.4.2. Subgroup-Discovery Methods

We employ six subgroup-discovery methods: A solver-based one using our SMT encoding, three heuristic search methods from related work, and our two baselines.

Solver-based search For solver-based search, denoted as *SMT*, we employ the solver *Z3* [28, 50] with our SMT encoding (cf. Equation 7.5). Unlike the other five subgroup-discovery methods, this method is exhaustive, i.e., it finds the global optimum if granted sufficient time. However, we set solver timeouts to control the runtime (cf. Section 7.4.3).

Heuristic search We evaluate three heuristic search methods from related work: *PRIM* (cf. Algorithm 1), *Beam Search* (cf. Algorithms 2 and 3), subsequently called *Beam*, and *Best Interval* (cf. Algorithms 2 and 4), subsequently called *BI*. For *PRIM*, we set the peeling fraction to $\alpha = 0.05$, consistent with other implementations [6, 116] and the range proposed by its authors [62]. We set the support threshold to $\beta_0 = 0$, so there is no constraint on the subgroup's size. For *Beam* and *BI*, we choose a beam width of $w = 10$, falling between default values used in other implementations [6, 129].

Baselines We also employ our baselines *MORS* (cf. Algorithm 9) and *Random Search* (cf. Algorithm 10), subsequently called *Random*. *MORS* is parameter-free. For *Random*, we set the number of iterations $n\_iters = 1000$.

7.4.3. Experimental Scenarios

Unconstrained subgroup discovery Our first experimental scenario (cf. Section 7.5.1 for results) compares all six subgroup-discovery methods without constraints. This comparison assesses the effectiveness of the solver-based search method *SMT* for conventional subgroup discovery and is a reference point for subsequent experiments with constraints. All methods except *MORS* use WRAcc (cf. Equation 2.3) as the subgroup-quality function $Q(lb, ub, X, y)$ for search. *MORS* optimizes its built-in objective (cf. Definition 11).

Solver timeouts Our second experimental scenario (cf. Section 7.5.2 for results) analyzes whether setting solver timeouts enables finding good solutions in a shorter time frame. If the solver does not finish optimization within a given timeout, we record the currently best solution at this time. We evaluate twelve exponentially scaled timeout values for *SMT*, i.e., {1 s, 2 s, 4 s, ..., 2048 s}. In the three other experimental scenarios, we employ the maximum, i.e., 2048 s. The timeout only applies to optimization, while our runtime measurements include preparation effort and may be higher. Since the heuristic search methods and baselines are significantly faster, we do not set timeouts for them.

Feature-cardinality constraints Our third experimental scenario (cf. Section 7.5.3 for results) analyzes feature-cardinality constraints (cf. Section 7.3.2) for all six subgroup-discovery methods. In particular, we evaluate $k \in \{1, 2, 3, 4, 5\}$ selected features. These values of k are upper bounds (cf. Equation 7.7), i.e., the subgroup-discovery methods may select fewer features if selecting more does not improve subgroup quality.

Alternative subgroup descriptions Our fourth experimental scenario (cf. Section 7.5.4 for results) studies alternative subgroup descriptions (cf. Section 7.3.3) for *SMT* and *Beam*, i.e., a solver-based and a heuristic search method. This scenario still optimizes WRAcc (cf. Equation 2.3) for the original subgroup but normalized Hamming similarity (cf. Equation 7.9) for the alternatives. The deselection dissimilarity (cf. Equation 7.11) ensures that the alternatives are dissimilar enough. We limit the feature cardinality to $k = 3$, which yields a high subgroup quality (cf. Section 7.5.3), and search for $a = 5$ alternatives with a dissimilarity threshold $\tau_{\text{abs}} \in \{1, 2, 3\}$. Since each dataset has $n \geq 20$ features (cf. Section 7.4.5), our choices of a, k, and τ ensure that there always is a valid alternative.

7.4.4. Evaluation Metrics

Subgroup quality We use *nWRAcc* (cf. Equation 2.5) to report subgroup quality. To analyze how well the subgroup descriptions generalize, we conduct a stratified five-fold cross-validation. In particular, we run the subgroup-discovery methods only on 80% of the data objects, while the remaining data objects serve as test data. We use the found subgroup description to determine subgroup membership for all data objects and compute *training-set nWRAcc* and *test-set nWRAcc* on the corresponding part of the data separately.

Subgroup similarity To evaluate alternative subgroup descriptions, we also consider their subgroup-membership similarity to the original subgroup. To this end, we use *normalized Hamming similarity* (cf. Equation 7.9) and *Jaccard similarity* (cf. Equation 7.10).

Runtime As *runtime*, we report the training time of the subgroup-discovery methods. In particular, we measure how long the search for a subgroup takes. For solver-based search, we also record whether the solver timed out.

Table 7.1.: Datasets from PMLB used in our experiments. m denotes the number of data objects and n the number of features. We replaced *GAMETES_Epistasis* with *GE_* and *GAMETES_Heterogeneity* with *GH_*. *Timeouts* indicates whether *SMT* timed out at least once with the highest timeout setting (2048 s), optimizing the original subgroup without (*Max k*) or with a feature-cardinality constraint (*Any k*).

Dataset	m	n	Timeouts	
			Max k	Any k
backache	180	32	No	No
chess	3196	36	No	No
churn	5000	20	Yes	Yes
clean1	476	168	No	No
clean2	6598	168	No	No
coil2000	9822	85	Yes	Yes
credit_g	1000	20	Yes	Yes
dis	3772	29	No	No
GE_2_Way_20atts_0.1H_EDM_1_1	1600	20	Yes	Yes
GE_2_Way_20atts_0.4H_EDM_1_1	1600	20	No	No
GE_3_Way_20atts_0.2H_EDM_1_1	1600	20	Yes	Yes
GH_20atts_1600_Het_0.4_0.2_50_EDM_2_001	1600	20	Yes	Yes
GH_20atts_1600_Het_0.4_0.2_75_EDM_2_001	1600	20	Yes	Yes
Hill_Valley_with_noise	1212	100	Yes	Yes
horse_colic	368	22	No	No
hypothyroid	3163	25	No	No
ionosphere	351	34	No	No
molecular_biology_promoters	106	57	No	No
mushroom	8124	22	No	No
ring	7400	20	Yes	Yes
sonar	208	60	No	Yes
spambase	4601	57	No	Yes
spect	267	22	No	No
spectf	349	44	No	Yes
tokyo1	959	44	No	Yes
twonorm	7400	20	Yes	Yes
wdbc	569	30	No	No

7.4.5. Datasets

As in Chapter 6, we use binary-classification datasets from the Penn Machine Learning Benchmarks (PMLB) [173, 186]. If the class labels occur with different frequencies, we encode the minority class as positive. To avoid prediction scenarios that may be too easy or do not have enough features for alternative subgroup descriptions, we only select datasets with at least 100 data objects and 20 features. Next, we exclude one dataset with 1000 features, which would dominate the overall runtime. Finally, we manually exclude datasets that seem to be duplicated or modified versions of others. These criteria yield 27 datasets

with 106 to 9822 data objects and 20 to 168 features (cf. Table 7.1). The datasets encode categorical features ordinally by default and do not contain missing values.

7.4.6. Implementation and Execution

We implemented our experimental pipeline in Python 3.8, using the SMT solver *Z3* [28, 50] for solver-based search. All code is available online (cf. Section 1.4). We organized the subgroup-discovery methods and evaluation metrics as a Python package to ease reuse.

Our experimental pipeline parallelizes over datasets, cross-validation folds, and subgroup-discovery methods, while each of these experimental tasks runs single-threaded. We ran the pipeline on a server with 160 GB RAM and an *AMD EPYC 7551* CPU, having 32 physical cores and a base clock of 2.0 GHz. The parallelized pipeline run took roughly 34 hours.

7.5. Evaluation

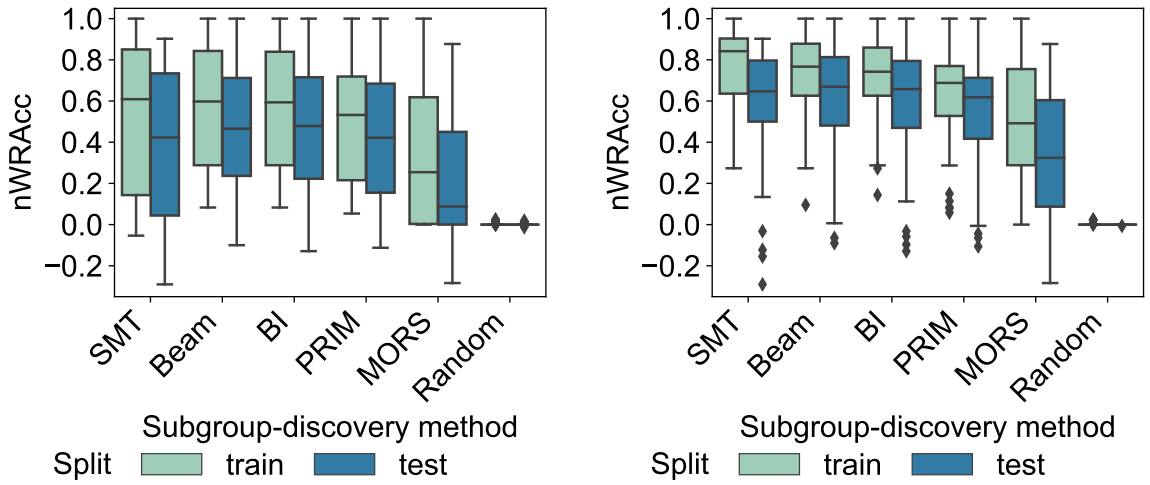
In this section, we evaluate our experiments for subgroup discovery. We cover our four experimental scenarios, i.e., unconstrained subgroup discovery (cf. Section 7.5.1), solver timeouts (cf. Section 7.5.2), feature-cardinality constraints (cf. Section 7.5.3), and alternative subgroup descriptions (cf. Section 7.5.4). Section 7.5.5 summarizes key findings.

7.5.1. Unconstrained Subgroup Discovery

In this section, we compare all six subgroup-discovery methods in the experimental scenario without constraints. *SMT* uses its default solver timeout of 2048 s.

Subgroup quality Figure 7.1a compares subgroup quality for the six subgroup-discovery methods. On the training set, the two heuristic search methods *Beam* and *BI* have roughly the same median nWRAcc as the solver-based search method *SMT*. In particular, the heuristics are even better than *SMT* on some datasets but worse on others. The former can happen when *SMT* yields suboptimal solutions due to timeouts, as we analyze later (cf. Section 7.5.2). However, even if we limit our analysis to the 17 datasets without *SMT* timeouts, *Beam* and *BI* are still remarkably close to the optimum quality (cf. Figure 7.1b). This result is not specific to *SMT* but also affects any other exhaustive search method. On the test set, *Beam* and *BI* are even better than *SMT* on median, also excluding timeout datasets, since their training-test nWRAcc difference is smaller. This result indicates that *Beam* and *BI* are less susceptible to overfitting, so their solutions generalize better. In detail, the average difference between training-set nWRAcc and test-set nWRAcc is 0.122 for *SMT*, 0.101 for *BI*, 0.095 for *Beam*, 0.094 for *MORS*, 0.068 for *PRIM*, and 0.001 for *Random*.

The heuristic search method *PRIM* yields worse subgroup quality than *Beam* and *BI*. Although it follows an iterative subgroup-refinement procedure like the latter two methods,



(a) All 27 datasets.

(b) 17 datasets without *SMT* timeouts.

Figure 7.1.: Distribution of subgroup quality over datasets and cross-validation folds, by subgroup-discovery method. Results from the unconstrained experimental scenario.

Table 7.2.: Aggregated runtime over datasets and cross-validation folds, by subgroup-discovery method. Results from the unconstrained experimental scenario.

(a) All 27 datasets.

Aggregate	BI	Beam	MORS	PRIM	Random	SMT
Mean	34.95 s	30.47 s	0.01 s	1.26 s	0.91 s	849.02 s
Standard deviation	103.61 s	85.69 s	0.00 s	1.51 s	0.95 s	929.60 s
Median	2.60 s	2.95 s	0.01 s	0.66 s	0.51 s	254.21 s

(b) 17 datasets without *SMT* timeouts.

Aggregate	BI	Beam	MORS	PRIM	Random	SMT
Mean	12.40 s	11.77 s	0.01 s	1.29 s	0.82 s	168.13 s
Standard deviation	21.17 s	20.47 s	0.00 s	1.62 s	0.89 s	243.11 s
Median	2.60 s	2.95 s	0.01 s	0.80 s	0.56 s	57.23 s

its refinement options are more limited. In particular, *PRIM* always has to remove a fixed fraction α of data objects from the subgroup (cf. Algorithm 1), while *Beam* and *BI* can remove more or less. Finally, all three heuristic search methods clearly beat the two baselines *MORS* and *Random*. While *Random* mostly yields the same quality as not restricting the subgroup at all, i.e., an nWRAcc of 0, *MORS* is considerably above 0 and thus a suitable baseline for future studies comparing subgroup-discovery methods.

Runtime Table 7.2 summarizes the runtimes of the subgroup-discovery methods. On average, *SMT* is an order of magnitude slower than *Beam* and *BI*, which are an order

Table 7.3.: Spearman correlation between runtime and metrics for dataset size, over datasets and cross-validation folds, by subgroup-discovery method. Results from the unconstrained experimental scenario, using the 17 datasets without *SMT* timeouts.

Method	Σn^u	$m \cdot n$	m	n
BI	0.95	0.51	0.32	0.67
Beam	0.96	0.49	0.30	0.66
MORS	0.27	0.57	0.51	0.26
PRIM	0.84	0.56	0.29	0.76
Random	0.58	0.69	0.42	0.77
SMT	0.39	0.73	0.70	0.23

of magnitude slower than *PRIM* and the baseline *Random*. The baseline *MORS* runs in negligible time and thus is a good tool for instantaneously obtaining a lower bound on subgroup quality. Taking subgroup quality into consideration, the heuristic search methods offer a good quality in a short time. Among the three heuristics, *PRIM* is the fastest but yields the lowest subgroup quality, so users should decide which runtime is acceptable.

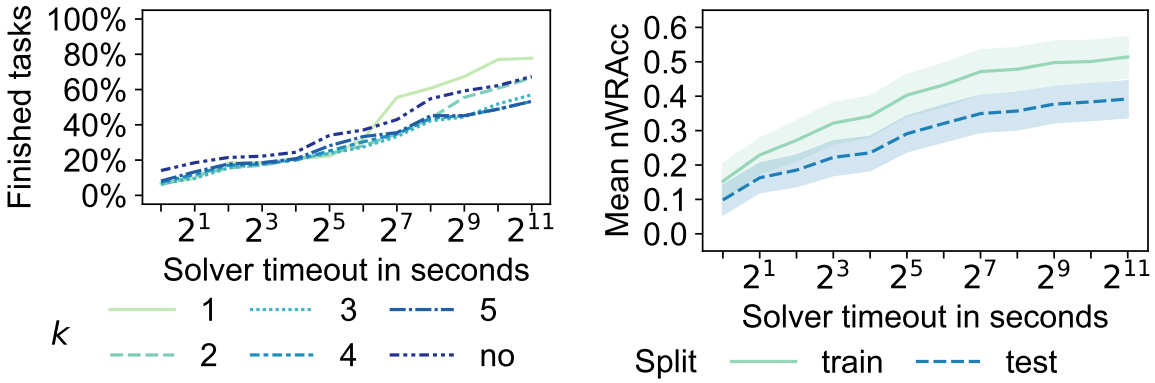
For *SMT*, the runtime not only comprises optimization but also formulating the optimization problem. Since the latter depends on the dataset size, e.g., involves $O(m)$ constraints with length $O(n)$ each to define the subgroup-membership variables b_i (cf. Equation 7.4), the preparation time can become considerable for large datasets. In our experiments, formulating the *SMT* problem took 45 s on average, with a maximum of 379 s. This average preparation time is already greater than the average total runtime of the heuristics.

To determine which factors influence runtime, we analyze the Spearman correlation between runtime and four simple metrics for dataset size. In particular, Table 7.3 considers the number of data objects m, the number of features n, the product of these two quantities $m \cdot n$, and the number of unique values per feature summed over the features Σn^u. For the three heuristic search methods, the latter metric shows a high correlation to runtime, while *SMT* exhibits the highest runtime correlation to $m \cdot n$.

7.5.2. Solver Timeouts

In this section, we evaluate the impact of solver timeouts for *SMT* search.

Finished tasks Figure 7.2a displays how many of the *SMT* optimization tasks for original subgroups finished within the evaluated solver timeouts. Besides the unconstrained tasks, we also consider different feature-cardinality thresholds, though the overall trend is the same. In particular, the number of finished tasks only increases slowly over time, and some tasks take orders of magnitude longer than others. E.g., in the unconstrained experimental scenario, 21.5% of the *SMT* tasks finished within 4 s, 24.4% within 16 s, 37.0% within 64 s, 54.8% within 256 s, and 62.2% within 1024 s. For the maximum setting of 2048 s, 67.4% of the *SMT* tasks finished, and 17 out of 27 datasets did not encounter timeouts (cf. Table 7.1).



(a) Frequency of finished *SMT* tasks over datasets and cross-validation folds, by feature cardinality k.

(b) Mean subgroup quality, with 95% confidence intervals based on datasets and cross-validation folds. Results from the unconstrained experimental scenario.

Figure 7.2.: Impact of solver timeouts for *SMT* as the subgroup-discovery method. Results from the search for original subgroups.

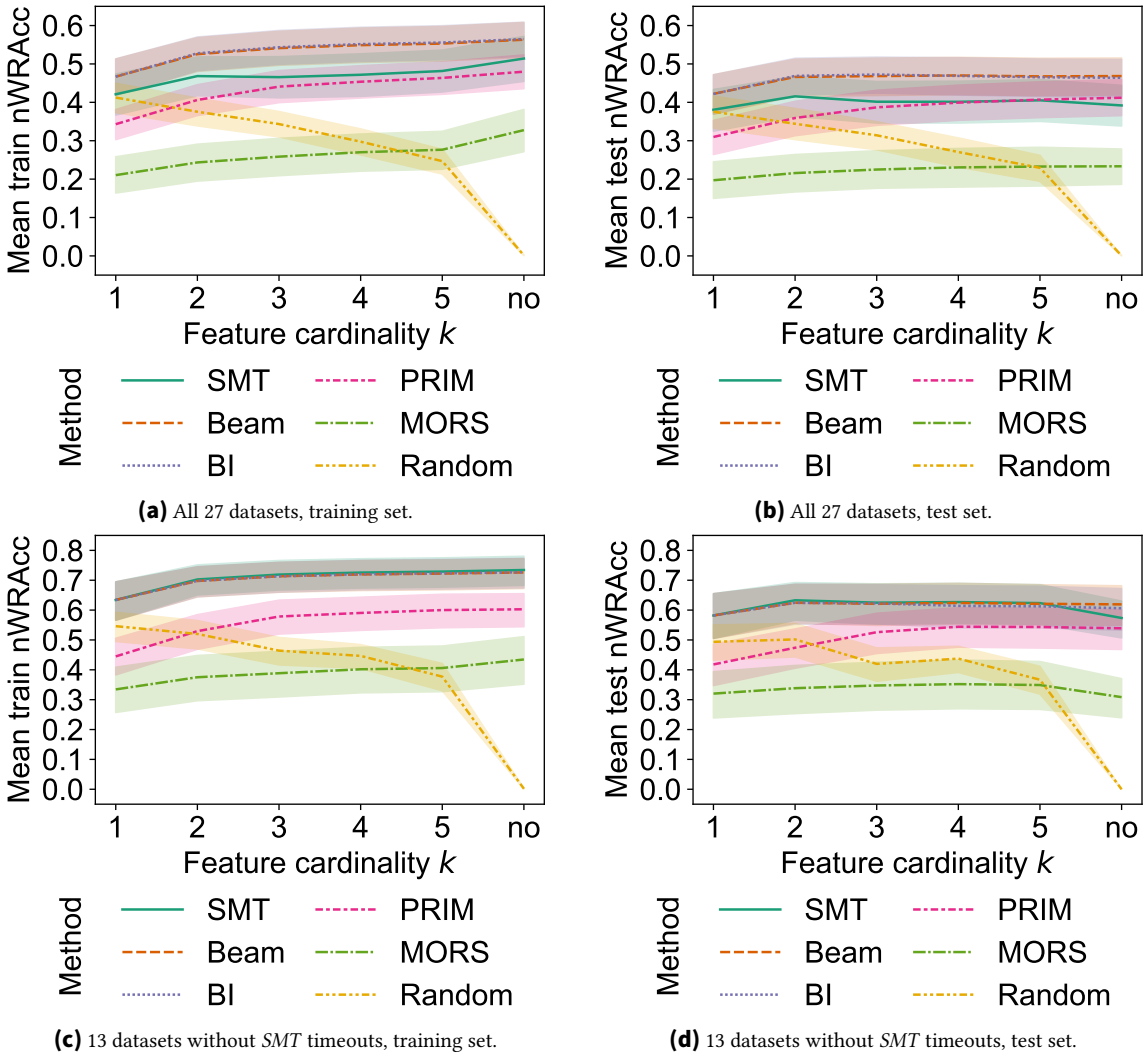
Subgroup quality Figure 7.2b visualizes the subgroup quality over solver timeouts for unconstrained *SMT* search. This plot uses the quality of the optimal solution for finished tasks and of the currently best solution for unfinished tasks. As for the number of finished tasks (cf. Figure 7.2a), the largest gains occur in the first minute. E.g., the mean test-set nWRAcc over datasets and cross-validation folds is 0.10 for 1 s, 0.19 for 4 s, 0.24 for 16 s, 0.32 for 64 s, and 0.39 for the maximum solver timeout of 2024 s. The main cause for this trend is that many tasks finish relatively early (cf. Figure 7.2a), and finished tasks cannot improve their quality for higher solver timeouts. In contrast, if we only consider the tasks where the solver did not finish even within the maximum solver timeout, the quality increase of the currently best solution over time is marginal.

Further, even *SMT* with a timeout does not compare favorably to fast heuristic search methods. E.g., with a solver timeout of 64 s, corresponding to an average overall runtime of 88 s, *SMT* achieves a mean training-set nWRAcc of 0.43, compared to 0.56 for *Beam* with an average runtime of 30 s (cf. Table 7.2a). Finally, setting a lower solver timeout decreases overfitting, i.e., the difference between training-set nWRAcc and test-set nWRAcc increases over time (cf. Figure 7.2b). However, since the test-set nWRAcc still increases with the timeout, choosing lower timeouts does not help quality-wise.

7.5.3. Feature-Cardinality Constraints

In this section, we compare all subgroup-discovery methods in the experimental scenario with feature-cardinality constraints. *SMT* uses its default solver timeout of 2048 s.

Subgroup quality Figure 7.3 displays the mean subgroup quality, averaging over datasets and cross-validation folds, for different values of the feature-cardinality threshold k. For most subgroup-discovery methods, mean training-set nWRAcc (cf. Figure 7.3a) increases



(a) All 27 datasets, training set.

(b) All 27 datasets, test set.

(c) 13 datasets without *SMT* timeouts, training set.

(d) 13 datasets without *SMT* timeouts, test set.

Figure 7.3.: Mean subgroup quality, with 95% confidence intervals based on datasets and cross-validation folds, by subgroup-discovery method and feature cardinality k. Results from the search for original subgroups.

with k, though the marginal utility decreases. In particular, even with $k = 1$, the mean nWRAcc is already clearly above 50% of the nWRAcc achieved with all features. Further, the quality increase between $k = 1$ and $k = 2$ is usually the largest. On the test set (cf. Figure 7.3b), the benefit of setting a larger k is even smaller. E.g., the mean test-set nWRAcc of *Beam*, *BI*, and *SMT* barely improves beyond $k = 2$. These results indicate that sparse subgroup descriptions already yield a high subgroup quality. The baseline *Random* even improves subgroup quality with lower k due to its design (cf. Algorithm 10): *Random* samples bounds independently for each feature. The more features are used in the subgroup description, the smaller the expected number of data objects in the subgroup becomes. However, the number of subgroup members is one factor in WRAcc (cf. Equation 2.3).

Figures 7.3a and 7.3b also reveal that the heuristic search methods *Beam* and *BI* still yield higher average subgroup quality than the solver-based search *SMT* due to timeouts, for any feature-cardinality setting. Even excluding the datasets with *SMT* timeouts (cf. Figures 7.3c

Table 7.4.: Mean runtime over datasets and cross-validation folds, by subgroup-discovery method and feature cardinality k. Results from the search for original subgroups.

k	BI	Beam	MORS	PRIM	Random	SMT
1	7.81 s	6.81 s	0.01 s	0.08 s	0.63 s	648.16 s
2	11.74 s	10.06 s	0.01 s	0.17 s	0.64 s	911.28 s
3	14.20 s	12.78 s	0.01 s	0.26 s	0.65 s	1091.75 s
4	16.68 s	14.65 s	0.01 s	0.35 s	0.66 s	1113.40 s
5	18.66 s	16.12 s	0.01 s	0.46 s	0.66 s	1117.39 s
no	34.95 s	30.47 s	0.01 s	1.26 s	0.91 s	849.02 s

and 7.3d), these two heuristics yield nearly the same average subgroup quality as *SMT* for constrained k and have an advantage on the test set with unconstrained k. The heuristic *PRIM* exhibits a larger increase of subgroup quality over k than *Beam* and *BI*, thereby narrowing the quality gap to the latter. The baseline *MORS* displays the least effect of k on mean test-set nWRAcc, showing very stable subgroup quality.

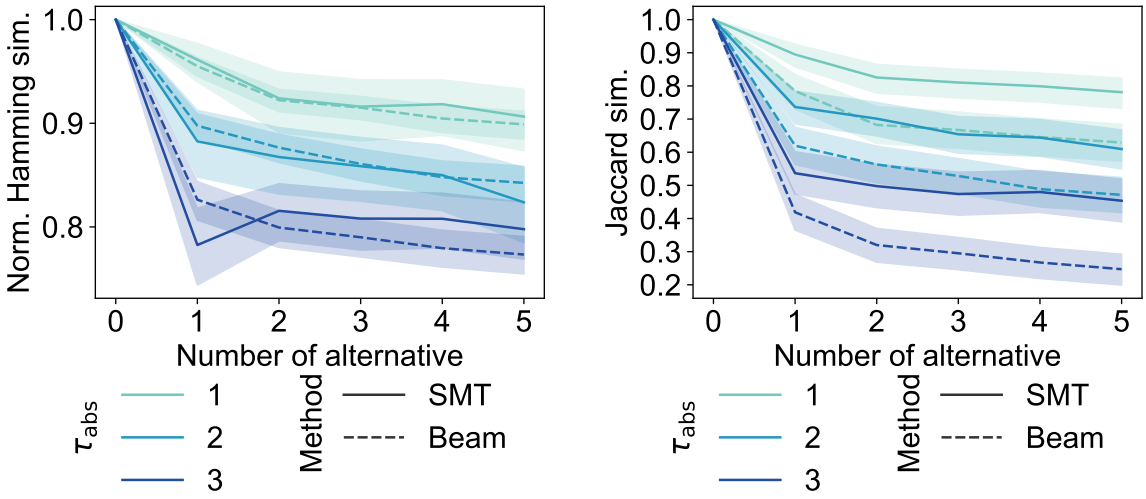
Finally, the results indicate that limiting k reduces overfitting. For example, *Beam* has a mean training-test nWRAcc difference of 0.095 without limiting k, 0.073 for $k = 3$, and 0.045 for $k = 1$. The increasing tendency to overfit with larger k explains why mean training-set nWRAcc increases more than mean test-set nWRAcc over k in Figure 7.3. *PRIM* shows the smallest increase of overfitting over k, *MORS* and *SMT* the largest.

Runtime As Table 7.4 displays, the heuristic search methods *Beam*, *BI*, and *PRIM* become faster the smaller k is. The baseline *Random* shows a similar trend, though less prominent, while *MORS* yields results instantaneously in any case. In contrast, the picture for the solver-based search method *SMT* is less clear. While its average runtime clearly increases from $k = 1$ till $k = 3$, it roughly remains constant for $k \in \{4, 5\}$ and even decreases without a feature-cardinality constraint, only remaining higher than for $k = 1$.

7.5.4. Alternative Subgroup Descriptions

In this section, we analyze alternative subgroup descriptions for *Beam* and *SMT*. Both employ a feature cardinality of $k = 3$. *SMT* uses its default solver timeout of 2048 s.

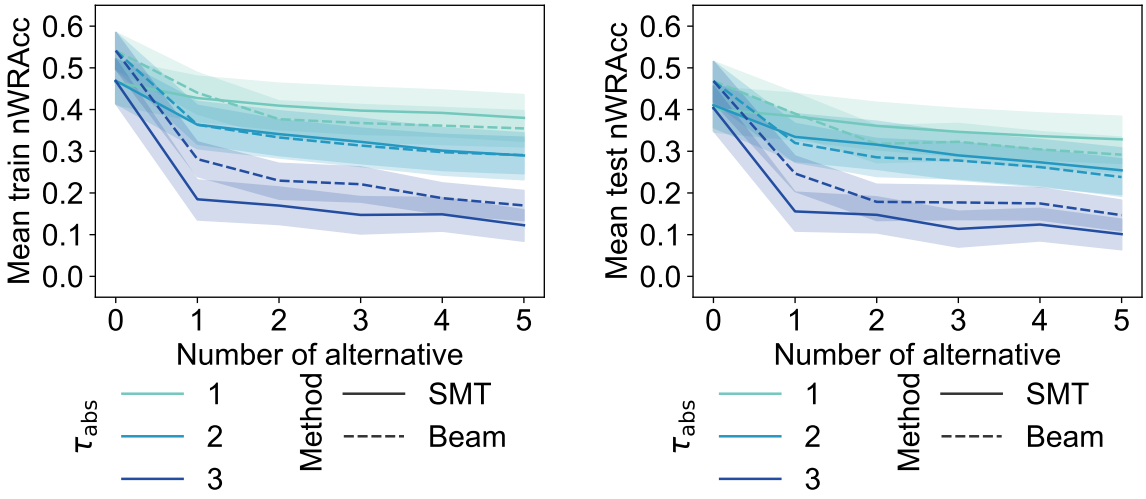
Subgroup similarity Figure 7.4 visualizes the average similarity between the original subgroups and those induced by alternative subgroup descriptions. As expected, the subgroup-membership similarity decreases the more alternatives one desires and the more the selected features in subgroup descriptions should differ. Further, the decrease is strongest from the original subgroup, i.e., the zeroth alternative, to the first alternative but smaller beyond. This observation indicates that one may find several alternative subgroup descriptions of comparable similarity to the original.



(a) Normalized Hamming similarity.

(b) Jaccard similarity.

Figure 7.4.: Mean subgroup similarity of alternative subgroup descriptions to the original subgroup, with 95% confidence intervals based on datasets and cross-validation folds, by subgroup-discovery method, number of alternative, and dissimilarity threshold τ_{abs}.



(a) Training set.

(b) Test set.

Figure 7.5.: Mean subgroup quality of alternative subgroup descriptions, with 95% confidence intervals based on datasets and cross-validation folds, by subgroup-discovery method, number of alternative, and dissimilarity threshold τ_{abs}.

These trends hold for both similarity measures, i.e., the normalized Hamming similarity we optimize (cf. Equation 7.9 and Figure 7.4a) as well as the Jaccard similarity (cf. Equation 7.10 and Figure 7.4b). The latter yields lower similarity values than the former since it ignores data objects that are not contained in either of the two compared subgroups. Further, the observed trends exist for the solver-based search method *SMT* as well as the heuristic search method *Beam*. *SMT* yields clearly more similar subgroups than *Beam* for the Jaccard similarity, while the normalized Hamming similarity does not show a clear winner.

Table 7.5.: Mean runtime over datasets and cross-validation folds, by subgroup-discovery method, dissimilarity threshold τ_{abs}, and number of alternative. Results from the search for alternative subgroup descriptions.

Method	τ_{abs}	Number of alternative					
		0	1	2	3	4	5
Beam	1	12.8 s	8.0 s	7.6 s	7.3 s	7.3 s	7.3 s
	2	12.8 s	7.7 s	7.4 s	7.2 s	7.0 s	6.8 s
	3	12.8 s	5.8 s	5.1 s	4.7 s	4.1 s	3.5 s
SMT	1	1091.7 s	166.0 s	221.5 s	239.6 s	258.1 s	277.9 s
	2	1105.2 s	377.5 s	463.5 s	537.5 s	599.4 s	658.3 s
	3	1107.4 s	869.1 s	670.8 s	597.6 s	588.1 s	557.6 s

Subgroup quality The average subgroup quality of alternative subgroup descriptions (cf. Figure 7.5) shows similar trends as subgroup similarity (cf. Figure 7.4) and the quality of alternative feature sets in our previous chapter (cf. Figure 6.5). In particular, quality decreases over the dissimilarity threshold τ_{abs} and over the number of alternatives a, with the largest decrease to the first alternative. For the highest dissimilarity threshold $\tau_{\text{abs}} = 3$, *Beam* consistently yields higher average quality than *SMT* for the original subgroup and each alternative. In contrast, the other two values of the dissimilarity threshold do not clearly favor either method. The observed trends on the test set (cf. Figure 7.5b) are very similar to those on the training set (cf. Figure 7.5a). For both subgroup-discovery methods, overfitting, as measured by the training-test difference in nWRAcc, is lower for the alternative subgroup descriptions than for the original subgroups. This phenomenon may result from the alternatives not directly optimizing subgroup quality.

Runtime Table 7.5 displays the average runtime for searching original subgroups and alternative subgroup descriptions. The search for alternatives is faster for both analyzed search methods, i.e., *Beam* and *SMT*. As for the original subgroups, *Beam* search for alternatives is one to two orders of magnitude faster than the solver-based *SMT* search. For *Beam*, runtime tends to decrease over the number of alternatives, while *SMT* shows a less clear behavior. In particular, its runtime increases over alternatives for $\tau_{\text{abs}} \in \{1, 2\}$, i.e., settings that allow reusing features from previous subgroup descriptions. In contrast, runtime decreases over alternatives for $\tau_{\text{abs}} = k = 3$, which forbids reusing any feature selected before. Finally, the number of *SMT* tasks finished within the solver timeout shows trends corresponding to the runtime. In particular, there are more finished tasks when searching for alternative subgroup descriptions than for original subgroups.

7.5.5. Summary

Unconstrained subgroup discovery (cf. Section 7.5.1) We recommend using heuristic search methods rather than solver-based search. In particular, *Beam* and *BI* were an order of magnitude faster than *SMT* and still yielded higher test-set subgroup quality since they

overfit less. The latter result not only impedes *SMT* but other exhaustive search methods as well. *PRIM* was faster than *Beam* and *BI* but yielded lower subgroup quality. Our novel baseline *MORS* provided instantaneous, non-trivial lower bounds for subgroup quality.

Solver timeouts (cf. Section 7.5.2) Setting larger solver timeouts showed a decreasing marginal utility regarding the number of finished *SMT* tasks and subgroup quality, i.e., most gains occurred within the first dozens of seconds. About half the *SMT* tasks that finished at all finished in under a minute. However, the average subgroup quality with this solver timeout was lower than for heuristic search methods with even lower runtime.

Feature-cardinality constraints (cf. Section 7.5.3) Using more features in subgroup descriptions showed a decreasing marginal utility regarding subgroup quality. For *Beam*, *BI*, and *SMT*, test-set subgroup quality was already close to the unconstrained scenario at $k = 2$, while *PRIM* benefited more from larger k. A smaller k made the heuristic search methods faster and generally reduced overfitting. *MORS* showed stable test-set subgroup quality regarding k, while *Random* even increased subgroup quality with smaller k.

Alternative subgroup descriptions (cf. Section 7.5.4) The heuristic *Beam* was one to two orders of magnitude faster than solver-based *SMT* when searching for alternative subgroup descriptions, while both search methods found alternatives faster than original subgroups. The quality and similarity of alternatives strongly depended on the number of alternatives a and the dissimilarity threshold τ_{abs}. The difference in quality and similarity between the original and the first alternative was higher than among the first few alternatives.

8. Conclusions

This dissertation focused on making feature selection more user-centric with the help of constraints. In particular, we tackled two research gaps that have not received sufficient attention in the literature before (cf. Section 1.2): integrating domain knowledge and finding alternative solutions. We addressed these two issues by formulating constraints on the selected feature sets. In particular, we made the following contributions:

In Chapter 4, we formally introduced the optimization problem of constrained feature selection. Our definition is orthogonal to the user's choice of a feature-selection method, as it retains the method's notion of feature-set quality. We proposed using an SMT solver for optimization, which supports a wide range of constraint types. Additionally, we systematically evaluated the impact of constraints on feature selection. Our comprehensive study generated constraints with varying characteristics and quantified the feature-selection results with multiple evaluation metrics. Section 4.4.4 summarizes central results.

In Chapter 5, we applied constrained feature selection in a domain-specific case study. We worked with materials scientists to express preferences and scientific hypotheses as constraints. Section 5.3.3 summarizes central results.

In Chapter 6, we introduced the optimization problem of alternative feature selection. We formalized alternatives via 0-1 integer linear constraints, which are independent of the feature-selection method and allow users to control the number and dissimilarity of alternatives. Additionally, we discussed how to combine these constraints with different categories of existing feature-selection methods. For a simple notion of feature-set quality, we showed that the optimization problem is $\mathcal{NP}$-hard but a constant-factor approximation exists, for which we proposed corresponding heuristics. Finally, we conducted extensive experiments with five feature-selection methods, five search methods for alternatives, and different values of the two user parameters. Section 6.4.4 summarizes central results.

In Chapter 7, we analyzed feature-selection-related constraints in subgroup discovery. We formalized subgroup discovery as an SMT optimization problem, focusing on two constraint types: limiting the number of features used in subgroups and searching for alternative subgroup descriptions. Further, we proved $\mathcal{NP}$-hardness of these two constrained optimization problems. We also showed how to integrate these constraint types into existing heuristic search methods for subgroup discovery. Finally, we evaluated heuristic and solver-based search in four experimental scenarios: unconstrained subgroup discovery, our two constraint types, and solver timeouts. Section 7.5.5 summarizes central results.

For all contributions, we made our code and experimental data available online (cf. Section 1.4). Finally, we will discuss directions for future work in the next chapter.

9. Future Work

In this section, we outline potential directions for future research.

Integrating more conventional methods Our notions of constraints and alternatives are orthogonal to conventional methods for feature selection and subgroup discovery. While we successfully showed how to integrate constraints, we could naturally only evaluate a combination with a few existing methods. Thus, there is room for integrating constraints into further methods. E.g., we implemented only one procedure to find alternatives for wrapper feature selection (cf. Section 6.2.3.2) and did not adapt embedded feature selection, which does not admit a generic solution procedure (cf. Section 6.2.3.3).

Further problem encodings and solvers In each chapter, we focused on one problem encoding and solver, i.e., SMT via *Z3* (cf. Chapters 4, 5, and 7) or 0-1 integer linear programming via *SCIP* (cf. Chapter 6). One could analyze further problem encodings, e.g., using MaxSAT, and compare additional solvers. For subgroup discovery, we assumed numerical features and a binary target (cf. Section 2.2.1). To support more datasets, one could adapt the encoding to multi-valued categorical features and continuous targets.

Heuristic search For alternative feature selection, we developed heuristic search methods and showed membership in the complexity class $\mathcal{APX}$ for univariate feature qualities (cf. Equation 2.1) under certain conditions (cf. Proposition 18). One could attempt to tighten the quality bounds we derived or show membership in a narrower complexity class. Additionally, one could develop heuristic search methods for other notions of feature-set quality. For subgroup discovery, we described how to integrate feature-cardinality constraints and alternative subgroup descriptions into heuristic search methods (cf. Sections 7.3.2.3 and 7.3.3.3). However, proving quality-approximation guarantees is still open.

Further definitions of alternatives One could vary the definition of alternatives, e.g., the set-dissimilarity measure (cf. Equations 6.3 and 7.11). However, one should ensure that the corresponding search method supports the desired change, e.g., requiring linearity for integer linear programming or antimonotonicity for specific heuristic search methods. For simultaneous alternatives, there are different options to aggregate the quality of feature sets (cf. Section 6.2.2.2). For subgroup discovery, one could analyze solver-based search for alternatives in the sense of covering different data objects, as used in related work (cf. Section 3.2.2), rather than covering the same data objects differently, as we did.

Soft constraints We only employed hard constraints, i.e., all constraints had to be satisfied. As we observed, this can result in reduced feature-set quality or even an infeasible solution. With soft constraints, users could attach individual penalties to each constraint violation and thereby specify how to trade off constraint satisfaction against feature-set quality. Alternatively, one could treat constraint satisfaction as another objective besides feature-set quality and apply multi-objective optimization. The set of Pareto-optimal solutions may allow users to compare different constraint-quality trade-offs.

Time complexity Our various $\mathcal{NP}$-hardness results made assumptions on the problem definition, e.g., parametrization of the search for alternatives. One could attempt to obtain further hardness results for other scenarios. For alternative feature selection, our analysis (cf. Section 6.2.4.2) assumed univariate feature qualities (cf. Equation 2.1). Results for other feature-selection methods may differ. In the univariate scenario, we obtained $\mathcal{NP}$-hardness for min-aggregation with feature-set overlap (cf. Proposition 15) and polynomial runtime for sum-aggregation without overlap (cf. Proposition 16). However, an analysis of sum-aggregation with overlap remains open. For subgroup discovery, we showed $\mathcal{NP}$-hardness of finding alternative subgroup descriptions (cf. Propositions 30, 31, and 32) but focused on scenarios without feature-set overlap. Additionally, our proofs assumed a feature-cardinality constraint (cf. Definition 15) when searching alternative subgroup descriptions. One could examine scenarios without this constraint type.

Case studies While we conducted a case study for constrained feature selection (cf. Chapters 5), our other experiments used generic benchmark datasets and quantitative evaluation metrics (cf. Chapters 4, 6, and 7). In particular, we focused on uncovering general trends rather than dataset-specific insights. Since all the methods introduced in this dissertation are domain-independent, one could also employ them in domain-specific case studies and interpret the corresponding results qualitatively, i.e., from the domain perspective.

User-friendly systems We made all our generally applicable methods available for reuse via Python packages. Also, we implemented specific evaluation routines to create figures and tables. However, users may prefer an application with a graphical user interface to run analyses and inspect results themselves. In particular, an interactive tool could ease trying out different constraints or varying the parameters for alternatives. For constrained feature selection, one would need to design a user-friendly way to formulate constraints. For all methods, runtime is an essential concern for interactivity. This point could be addressed with solver timeouts, fast heuristics, and runtime estimates shown to users.

Constrained feature engineering In this dissertation, we took the datasets as-is, i.e., assuming all features were engineered already. However, one could integrate constraints into feature engineering as well, steering the creation of new features rather than the selection of existing ones. For example, one could desire only to engineer features with specific characteristics or limit the combination of certain feature-engineering operators.

Bibliography

[1] Ankit Agrawal et al. “Exploration of data science techniques to predict fatigue strength of steel from composition and processing parameters”. In: *Integr. Mater. Manuf. Innovation* 3.1 (2014), pp. 90–108. DOI: 10.1186/2193-9772-3-8.

[2] Prachi Agrawal et al. “Metaheuristic Algorithms on Feature Selection: A Survey of One Decade of Research (2009-2019)”. In: *IEEE Access* 9 (2021), pp. 26766–26791. DOI: 10.1109/ACCESS.2021.3056407.

[3] Enrique Matos Alfonso and Norbert Manthey. “New CNF Features and Formula Classification”. In: *Proc. POS*. Vienna, Austria, 2014, pp. 57–71. DOI: 10.29007/b8t1.

[4] Noga Alon et al. “Approximation schemes for scheduling on parallel machines”. In: *J. Sched.* 1.1 (1998), pp. 55–66. DOI: 10.1002/(SICI)1099-1425(199806)1:1<55::AID-JOS2>3.0.CO;2-J.

[5] André Artelt and Barbara Hammer. ““Even if ...” – Diverse Semifactual Explanations of Reject”. In: *Proc. SSCI*. Singapore, Singapore, 2022, pp. 854–859. DOI: 10.1109/SSCI51031.2022.10022139.

[6] Vadim Arzamasov and Klemens Böhm. “REDS: Rule Extraction for Discovering Scenarios”. In: *Proc. SIGMOD*. Virtual Event, China, 2021, pp. 115–128. DOI: 10.1145/3448016.3457301.

[7] Vadim Arzamasov, Benjamin Jochum, and Klemens Böhm. *Pedagogical Rule Extraction to Learn Interpretable Models — an Empirical Study*. arXiv:2112.13285v2 [cs.LG]. 2022. DOI: 10.48550/arXiv.2112.13285.

[8] Martin Atzmueller. “Subgroup discovery”. In: *WIREs Data Min. Knowl. Disc.* 5.1 (2015), pp. 35–49. DOI: 10.1002/widm.1144.

[9] Martin Atzmueller and Florian Lemmerich. “Fast Subgroup Discovery for Continuous Target Concepts”. In: *Proc. ISMIS*. Prague, Czech Republic, 2009, pp. 35–44. DOI: 10.1007/978-3-642-04125-9_7.

[10] Martin Atzmueller and Frank Puppe. “A Methodological View on Knowledge-Intensive Subgroup Discovery”. In: *Proc. EKAW*. Poděbrady, Czech Republic, 2006, pp. 318–325. DOI: 10.1007/11891451_28.

[11] Martin Atzmueller and Frank Puppe. “SD-Map – A Fast Algorithm for Exhaustive Subgroup Discovery”. In: *Proc. PKDD*. Berlin, Germany, 2006, pp. 6–17. DOI: 10.1007/11871637_6.

[12] Martin Atzmueller, Frank Puppe, and Hans-Peter Buscher. "Exploiting Background Knowledge for Knowledge-Intensive Subgroup Discovery". In: *Proc. IJCAI*. Edinburgh, United Kingdom, 2005, pp. 647–652. URL: https://www.ijcai.org/Proceedings/05/Papers/1217.pdf.

[13] Martin Atzmueller and Dietmar Seipel. "Using Declarative Specifications of Domain Knowledge for Descriptive Data Mining". In: *Proc. INAP*. Würzburg, Germany, 2007, pp. 149–164. DOI: 10.1007/978-3-642-00675-3_10.

[14] Luitpold Babel, Hans Kellerer, and Vladimir Kotov. "The k-Partitioning Problem". In: *Math. Methods Oper. Res.* 47.1 (1998), pp. 59–82. DOI: 10.1007/BF01193837.

[15] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. "Maximum Satisfiability". In: *Handbook of Satisfiability*. 2nd ed. IOS Press, 2021. Chap. 24, pp. 929–991. DOI: 10.3233/FAIA201008.

[16] Jakob Bach. *Finding Optimal Diverse Feature Sets with Alternative Feature Selection*. arXiv:2307.11607v2 [cs.LG]. 2024. DOI: 10.48550/arXiv.2307.11607.

[17] Jakob Bach. *Leveraging Constraints for User-Centric Selection of Predictive Features*. AI Hub @ Karlsruhe. 2022. DOI: 10.5445/IR/1000151285.

[18] Jakob Bach. *Using Constraints to Discover Sparse and Alternative Subgroup Descriptions*. arXiv:2406.01411v1 [cs.LG]. 2024. DOI: 10.48550/arXiv.2406.01411.

[19] Jakob Bach and Klemens Böhm. "Alternative feature selection with user control". In: *Int. J. Data Sci. Anal.* (2024). DOI: 10.1007/s41060-024-00527-8.

[20] Jakob Bach et al. "An Empirical Evaluation of Constrained Feature Selection". In: *SN Comput. Sci.* 3.6 (2022). DOI: 10.1007/s42979-022-01338-z.

[21] Eric Bae and James Bailey. "COALA: A Novel Approach for the Extraction of an Alternate Clustering of High Quality and High Dissimilarity". In: *Proc. ICDM*. Hong Kong, China, 2006, pp. 53–62. DOI: 10.1109/ICDM.2006.37.

[22] Eric Bae, James Bailey, and Guozhu Dong. "A clustering comparison measure using density profiles and its application to the discovery of alternate clusterings". In: *Data Min. Knowl. Disc.* 21.3 (2010), pp. 427–471. DOI: 10.1007/s10618-009-0164-z.

[23] James Bailey. "Alternative Clustering Analysis: A Review". In: *Data Clustering: Algorithms and Applications*. 1st ed. CRC Press, 2014. Chap. 21, pp. 535–550. DOI: 10.1201/9781315373515.

[24] Clark Barrett and Cesare Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Model Checking*. 1st ed. Springer, 2018. Chap. 11, pp. 305–343. DOI: 10.1007/978-3-319-10575-8_11.

[25] Adnene Belfodil et al. "FSSD - A Fast and Efficient Algorithm for Subgroup Set Discovery". In: *Proc. DSAA*. Washington, DC, USA, 2019, pp. 91–99. DOI: 10.1109/DSAA.2019.00023.

[26] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. "Automated analysis of feature models 20 years later: A literature review". In: *Inf. Syst.* 35.6 (2010), pp. 615–636. DOI: 10.1016/j.is.2010.01.001.

[27] Ksenia Bestuzheva et al. *The SCIP Optimization Suite 8.0*. Tech. rep. Zuse Institute Berlin, 2021. URL: http://nbn-resolving.de/urn:nbn:de:0297-zib-85309.

[28] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. "*v*Z - An Optimizing SMT Solver". In: *Proc. TACAS*. London, United Kingdom, 2015, pp. 194–199. DOI: 10.1007/978-3-662-46681-0_14.

[29] Mario Boley and Henrik Grosskreutz. "Non-redundant Subgroup Discovery Using a Closure System". In: *Proc. ECML PKDD*. Bled, Slovenia, 2009, pp. 179–194. DOI: 10.1007/978-3-642-04180-8_29.

[30] Tibérius O. Bonates, Peter L. Hammer, and Alexander Kogan. "Maximum patterns in datasets". In: *Discrete Appl. Math.* 156.6 (2008), pp. 846–861. DOI: 10.1016/j.dam.2007.06.004.

[31] Giorgos Borboudakis and Ioannis Tsamardinos. "Extending greedy feature selection algorithms to multiple solutions". In: *Data Min. Knowl. Disc.* 35.4 (2021), pp. 1393–1434. DOI: 10.1007/s10618-020-00731-7.

[32] Guillaume Bosc et al. "Anytime discovery of a diverse set of patterns with Monte Carlo tree search". In: *Data Min. Knowl. Disc.* 32.3 (2018), pp. 604–650. DOI: 10.1007/s10618-017-0547-5.

[33] Leo Breiman. "Random Forests". In: *Mach. Learn.* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[34] Leo Breiman et al. *Classification and Regression Trees*. 1st ed. Chapman and Hall, 1984. DOI: 10.1201/9781315139470.

[35] Cristóbal J. Carmona, María J. del Jesus, and Francisco Herrera. "A unifying analysis for the supervised descriptive rule discovery via the weighted relative accuracy". In: *Knowledge-Based Syst.* 139 (2018), pp. 89–100. DOI: 10.1016/j.knosys.2017.10.015.

[36] Diogo V. Carvalho, Eduardo M. Pereira, and Jaime S. Cardoso. "Machine Learning Interpretability: A Survey on Methods and Metrics". In: *Electronics* 8.8 (2019). DOI: 10.3390/electronics8080832.

[37] Girish Chandrashekar and Ferat Sahin. "A survey on feature selection methods". In: *Comput. Electr. Eng.* 40.1 (2014), pp. 16–28. DOI: 10.1016/j.compeleceng.2013.11.024.

[38] Chandra Chekuri and Amit Kumar. "Maximum Coverage Problem with Group Budget Constraints and Applications". In: *Proc. APPROX*. Cambridge, MA, USA, 2004, pp. 72–83. DOI: 10.1007/978-3-540-27821-4_7.

[39] Lin Chen et al. "An Efficient PTAS for Parallel Machine Scheduling with Capacity Constraints". In: *Proc. COCOA*. Hong Kong, China, 2016, pp. 608–623. DOI: 10.1007/978-3-319-48749-6_44.

[40] Shi Ping Chen, Yong He, and Guohui Lin. "3-Partitioning Problems for Maximizing the Minimum Load". In: *J. Comb. Optim.* 6.1 (2002), pp. 67–80. DOI: 10.1023/A:1013370208101.

[41] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proc. KDD*. San Francisco, CA, USA, 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.

[42] Christopher M. Childs and Newell R. Washburn. "Embedding domain knowledge for machine learning of complex material systems". In: *MRS Commun.* 9.3 (2019), pp. 806–820. DOI: 10.1557/mrc.2019.90.

[43] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C. Tappert. "A Survey of Binary Similarity and Distance Measures". In: *J. Syst. Cybern. Inf.* 8.1 (2010), pp. 43–48. URL: http://www.iiisci.org/Journal/pdv/sci/pdfs/GS315JG.pdf.

[44] Stephen A. Cook. "The Complexity of Theorem-Proving Procedures". In: *Proc. STOC*. Shaker Heights, OH, USA, 1971, pp. 151–158. DOI: 10.1145/800157.805047.

[45] Ian Covert, Scott M. Lundberg, and Su-In Lee. "Understanding Global Feature Contributions With Additive Importance Measures". In: *Proc. NeurIPS*. Virtual-only Conference, 2020, pp. 17212–17223. URL: https://proceedings.neurips.cc/paper_files/paper/2020/hash/c7bf0b7c1a86d5eb3be2c722cf2cf746-Abstract.html.

[46] Giordano d'Aloisio, Antinisca Di Marco, and Giovanni Stilo. "Democratizing Quality-Based Machine Learning Development through Extended Feature Models". In: *Proc. FASE*. Paris, France, 2023, pp. 88–110. DOI: 10.1007/978-3-031-30826-0_5.

[47] Susanne Dandl et al. "Multi-Objective Counterfactual Explanations". In: *Proc. PPSN*. Leiden, The Netherlands, 2020, pp. 448–469. DOI: 10.1007/978-3-030-58112-1_31.

[48] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. "A Declarative Framework for Constrained Clustering". In: *Proc. ECML PKDD*. Prague, Czech Republic, 2013, pp. 419–434. DOI: 10.1007/978-3-642-40994-3_27.

[49] Thi-Bich-Hanh Dao and Christel Vrain. "A review on declarative approaches for constrained clustering". In: *Int. J. Approximate Reasoning* 171 (2024). DOI: 10.1016/j.ijar.2024.109135.

[50] Leonardo De Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". In: *Proc. TACAS*. Budapest, Hungary, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24.

[51] Bryan L. Deuermeyer, Donald K. Friesen, and Michael A. Langston. "Scheduling to Maximize the Minimum Processor Finish Time in a Multiprocessor System". In: *SIAM J. Algebraic Discrete Methods* 3.2 (1982), pp. 190–196. DOI: 10.1137/0603019.

[52] Daniel Deutch and Nave Frost. "Constraints-based Explanations of Classifications". In: *Proc. ICDE*. Macao, China, 2019, pp. 530–541. DOI: 10.1109/ICDE.2019.00054.

[53] Pradip Dhal and Chandrashekhar Azad. "A comprehensive survey on feature selection in the various fields of machine learning". In: *Appl. Intell.* 52.4 (2021), pp. 4543–4581. DOI: 10.1007/s10489-021-02550-9.

[54] Rodney G. Downey, Michael R. Fellows, and Ulrike Stege. "Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability". In: *Proc. DIMATIA-DIMACS*. Štiřín Castle, Czech Republic, 1997, pp. 49–99. DOI: 10.1090/dimacs/049/04.

[55] Jonathan Eckstein et al. "The Maximum Box Problem and its Application to Data Analysis". In: *Comput. Optim. Appl.* 23.3 (2002), pp. 285–298. DOI: 10.1023/A:1020546910706.

[56] Leo Egghe. "New Relations Between Similarity Measures for Vectors Based on Vector Norms". In: *J. Am. Soc. Inf. Sci. Technol.* 60.2 (2009), pp. 232–239. DOI: 10.1002/asi.20949.

[57] Christos Emmanouilidis et al. "Selecting Features in Neurofuzzy Modelling by Multiobjective Genetic Algorithms". In: *Proc. ICANN*. Edinburgh, United Kingdom, 1999, pp. 749–754. DOI: 10.1049/cp:19991201.

[58] Stefano Ermon, Carla Gomes, and Bart Selman. "Uniform Solution Sampling Using a Constraint Solver As an Oracle". In: *Proc. UAI*. Catalina Island, CA, USA, 2012, pp. 255–264. URL: https://www.auai.org/uai2012/papers/160.pdf.

[59] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. "All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously". In: *J. Mach. Learn. Res.* 20.177 (2019). URL: https://jmlr.org/papers/v20/18-760.html.

[60] Edouard Fouché, Florian Kalinke, and Klemens Böhm. "Efficient subspace search in data streams". In: *Inf. Syst.* 97 (2021). DOI: 10.1016/j.is.2020.101705.

[61] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *A note on the group lasso and a sparse group lasso*. arXiv:1001.0736v1 [math.ST]. 2010. DOI: 10.48550/arXiv.1001.0736.

[62] Jerome H. Friedman and Nicholas I. Fisher. "Bump hunting in high-dimensional data". In: *Stat. Comput.* 9.2 (1999), pp. 123–143. DOI: 10.1023/A:1008894516817.

[63] Daniel Fryer, Inga Strümke, and Hien Nguyen. "Shapley Values for Feature Selection: The Good, the Bad, and the Axioms". In: *IEEE Access* 9 (2021), pp. 144352–144360. DOI: 10.1109/ACCESS.2021.3119110.

[64] Esther Galbrun and Pauli Miettinen. *Redescription Mining*. 1st ed. Springer, 2017. DOI: 10.1007/978-3-319-72889-6.

[65] José A. Galindo et al. "Automated analysis of feature models: Quo vadis?" In: *Computing* 101.5 (2019), pp. 387–433. DOI: 10.1007/s00607-018-0646-1.

[66] Arianna Gallo, Pauli Miettinen, and Heikki Mannila. "Finding Subgroups having Several Descriptions: Algorithms for Redescription Mining". In: *Proc. SDM*. Atlanta, GA, USA, 2008, pp. 334–345. DOI: 10.1137/1.9781611972788.30.

[67] Dragan Gamberger and Nada Lavrač. "Expert-Guided Subgroup Discovery: Methodology and Application". In: *J. Artif. Intell. Res.* 17 (2002), pp. 501–527. DOI: 10.1613/jair.1089.

[68] Michael R. Garey and David S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. 24th ed. W. H. Freeman and Company, 2003. URL: https://www.worldcat.org/title/440655898.

[69] Leilani H. Gilpin et al. "Explaining Explanations: An Overview of Interpretability of Machine Learning". In: *Proc. DSAA*. Turin, Italy, 2018, pp. 80–89. DOI: 10.1109/DSAA.2018.00018.

[70] Niku Gorji and Sasha Rubin. "Sufficient Reasons for Classifier Decisions in the Presence of Domain Constraints". In: *Proc. AAAI*. Virtual Conference, 2022, pp. 5660–5667. DOI: 10.1609/aaai.v36i5.20507.

[71] Valerio Grossi, Andrea Romei, and Franco Turini. "Survey on using constraints in data mining". In: *Data Min. Knowl. Disc.* 31.2 (2017), pp. 424–464. DOI: 10.1007/s10618-016-0480-z.

[72] Henrik Großkreutz, Daniel Paurat, and Stefan Rüping. "An Enhanced Relevance Criterion For More Concise Supervised Pattern Discovery". In: *Proc. KDD*. Beijing, China, 2012, pp. 1442–1450. DOI: 10.1145/2339530.2339756.

[73] Henrik Grosskreutz and Stefan Rüping. "On subgroup discovery in numerical domains". In: *Data Min. Knowl. Disc.* 19.2 (2009), pp. 210–226. DOI: 10.1007/s10618-009-0136-3.

[74] William Christopher Groves. "Toward Automating and Systematizing the Use of Domain Knowledge in Feature Selection". PhD thesis. University of Minnesota, 2015. URL: https://hdl.handle.net/11299/175444.

[75] Riccardo Guidotti. "Counterfactual explanations and how to find them: literature review and benchmarking". In: *Data Min. Knowl. Disc.* (2022). DOI: 10.1007/s10618-022-00831-6.

[76] Stephan Günnemann et al. "Detection of Orthogonal Concepts in Subspaces of High Dimensional Data". In: *Proc. CIKM*. Hong Kong, China, 2009, pp. 1317–1326. DOI: 10.1145/1645953.1646120.

[77] Tias Guns, Siegfried Nijssen, and Luc De Raedt. "Itemset mining: A constraint programming perspective". In: *Artif. Intell.* 175.12-13 (2011), pp. 1951–1983. DOI: 10.1016/j.artint.2011.05.002.

[78] Tias Guns, Siegfried Nijssen, and Luc De Raedt. "k-Pattern Set Mining under Constraints". In: *IEEE Trans. Knowl. Data Eng.* 25.2 (2011), pp. 402–418. DOI: 10.1109/TKDE.2011.204.

[79] Jianmei Guo and Kai Shi. "To Preserve or Not to Preserve Invalid Solutions in Search-Based Software Engineering: A Case Study in Software Product Lines". In: *Proc. ICSE*. Gothenburg, Sweden, 2018, pp. 1027–1038. DOI: 10.1145/3180155.3180163.

[80] Jianmei Guo et al. "A genetic algorithm for optimized feature selection with resource constraints in software product lines". In: *J. Syst. Software* 84.12 (2011), pp. 2208–2221. DOI: 10.1016/j.jss.2011.06.026.

[81] Jianmei Guo et al. “Scaling Exact Multi-Objective Combinatorial Optimization by Parallelization”. In: *Proc. ASE*. Vasteras, Sweden, 2014, pp. 409–420. DOI: 10.1145/2642937.2642971.

[82] Jianmei Guo et al. “SMTIBEA: a hybrid multi-objective optimization algorithm for configuring large constrained software product lines”. In: *Software Syst. Model.* 18.2 (2019), pp. 1447–1466. DOI: 10.1007/s10270-017-0610-0.

[83] D. S. Guru et al. “An alternative framework for univariate filter based feature selection for text categorization”. In: *Pattern Recognit. Lett.* 103 (2018), pp. 23–31. DOI: 10.1016/j.patrec.2017.12.025.

[84] Isabelle Guyon and André Elisseeff. “An Introduction to Variable and Feature Selection”. In: *J. Mach. Learn. Res.* 3.Mar (2003), pp. 1157–1182. URL: https://www.jmlr.org/papers/v3/guyon03a.html.

[85] Mohamed Haouari and Mahdi Jemmali. “Maximizing the minimum completion time on parallel machines”. In: *4OR* 6.4 (2008), pp. 375–392. DOI: 10.1007/s10288-007-0053-5.

[86] Mark Harman et al. “Search based software engineering for software product line engineering: a survey and directions for future work”. In: *Proc. SPLC*. Florence, Italy, 2014, pp. 5–18. DOI: 10.1145/2648511.2648513.

[87] Yong He et al. “k-Partitioning Problems for Maximizing the Minimum Load”. In: *Comput. Math. Appl.* 46.10-11 (2003), pp. 1671–1681. DOI: 10.1016/S0898-1221(03)90201-X.

[88] Sumyea Helal. “Subgroup Discovery Algorithms: A Survey and Empirical Evaluation”. In: *J. Comput. Sci. Technol.* 31.3 (2016), pp. 561–576. DOI: 10.1007/s11390-016-1647-1.

[89] Christopher Henard et al. “Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines”. In: *Proc. ICSE*. Florence, Italy, 2015, pp. 517–528. DOI: 10.1109/ICSE.2015.69.

[90] Franciso Herrera et al. “An overview on subgroup discovery: foundations and applications”. In: *Knowl. Inf. Syst.* 29.3 (2011), pp. 495–525. DOI: 10.1007/s10115-010-0356-2.

[91] Samah Hijazi et al. “Active learning of constraints for weighted feature selection”. In: *Adv. Data Anal. Classif.* 15.2 (2021), pp. 337–377. DOI: 10.1007/s11634-020-00408-5.

[92] Juhua Hu and Jian Pei. “Subspace multi-clustering: a review”. In: *Knowl. Inf. Sys.* 56.2 (2018), pp. 257–284. DOI: 10.1007/s10115-017-1110-9.

[93] Tran Doan Huan, Arun Mannodi-Kanakkithodi, and Rampi Ramprasad. “Accelerated materials property predictions and design using motif-based fingerprints”. In: *Phys. Rev. B* 92.1 (2015). DOI: 10.1103/PhysRevB.92.014106.

[94] Dan Hudson and Martin Atzmueller. “Subgroup Discovery with SD4Py”. In: *Proc. ECAI Workshops*. Kraków, Poland, 2023, pp. 338–348. DOI: 10.1007/978-3-031-50396-2_19.

[95] Alexey Ignatiev et al. "Reasoning-Based Learning of Interpretable ML Models". In: *Proc. IJCAI*. Montreal, Canada, 2021, pp. 4458–4465. DOI: 10.24963/ijcai.2021/608.

[96] Giulio Imbalzano et al. "Automatic selection of atomic fingerprints and reference configurations for machine-learning potentials". In: *J. Chem. Phys.* 148.24 (2018). DOI: 10.1063/1.5024611.

[97] Laurent Jacob, Guillaume Obozinski, and Jean-Philippe Vert. "Group Lasso with Overlap and Graph Lasso". In: *Proc. ICML*. 2009, pp. 433–440. DOI: 10.1145/1553374.1553431.

[98] Rudolf Jagdhuber et al. "Cost-Constrained feature selection in binary classification: adaptations for greedy forward selection and genetic algorithms". In: *BMC Bioinf.* 21 (2020). DOI: 10.1186/s12859-020-3361-9.

[99] Sarthak Jain and Byron C. Wallace. "Attention is not Explanation". In: *Proc. NAACL-HLT*. Minneapolis, MN, USA, 2019, pp. 3543–3556. DOI: 10.18653/v1/N19-1357.

[100] Gareth James et al. "Linear Regression". In: *An Introduction to Statistical Learning with Applications in R*. Springer, 2013. Chap. 3, pp. 59–126. DOI: 10.1007/978-1-4614-7138-7_3.

[101] Jon Paul Janet and Heather J. Kulik. "Resolving Transition Metal Chemical Space: Feature Selection for Machine Learning and Structure–Property Relationships". In: *J. Phys. Chem. A* 121.46 (2017), pp. 8939–8954. DOI: 10.1021/acs.jpca.7b08750.

[102] Amir-Hossein Karimi et al. "Model-Agnostic Counterfactual Explanations for Consequential Decisions". In: *Proc. AISTATS*. Online, 2020, pp. 895–905. URL: https://proceedings.mlr.press/v108/karimi20a.html.

[103] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. 1st ed. Plenum Press, 1972. Chap. 9, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[104] Anuj Karpatne et al. "Theory-Guided Data Science: A New Paradigm for Scientific Discovery from Data". In: *IEEE Trans. Knowl. Data Eng.* 29.10 (2017), pp. 2318–2331. DOI: 10.1109/TKDE.2017.2720168.

[105] Hans Kellerer and Vladimir Kotov. "A 3/2-approximation algorithm for k_i-partitioning". In: *Oper. Res. Lett.* 39.5 (2011), pp. 359–362. DOI: 10.1016/j.orl.2011.06.005.

[106] Sanjeev Khanna, Madhu Sudan, and David P. Williamson. "A Complete Classification of the Approximability of Maximization Problems Derived from Boolean Constraint Satisfaction". In: *Proc. STOC*. El Paso, TX, USA, 1997, pp. 11–20. DOI: 10.1145/258533.258538.

[107] Sanjeev Khanna et al. "On Syntactic Versus Computational Views of Approximability". In: *SIAM J. Comput.* 28.1 (1998), pp. 164–191. DOI: 10.1137/S0097539795286612.

[108] Rami N. Khushaba, Ahmed Al-Ani, and Adel Al-Jumaily. "Feature subset selection using differential evolution and a statistical repair mechanism". In: *Expert Syst. Appl.* 38.9 (2011), pp. 11515–11526. DOI: 10.1016/j.eswa.2011.03.028.

[109] Been Kim, Rajiv Khanna, and Oluwasanmi Koyejo. "Examples are not Enough, Learn to Criticize! Criticism for Interpretability". In: *Proc. NIPS*. Barcelona, Spain, 2016. URL: https://proceedings.neurips.cc/paper_files/paper/2016/hash/5680522b8e2bb01943234bce7bf84534-Abstract.html.

[110] Mi-Young Kim et al. "A Multi-Component Framework for the Analysis and Design of Explainable Artificial Intelligence". In: *Mach. Learn. Knowl. Extr.* 3.4 (2021), pp. 900–921. DOI: 10.3390/make3040045.

[111] Gökberk Koçak et al. "Exploiting Incomparability in Solution Dominance: Improving General Purpose Constraint-Based Mining". In: *Proc. ECAI*. Santiago de Compostela, Spain, 2020, pp. 331–338. DOI: 10.3233/FAIA200110.

[112] Ron Kohavi and George H. John. "Wrappers for feature subset selection". In: *Artif. Intell.* 97.1-2 (1997), pp. 273–324. DOI: 10.1016/S0004-3702(97)00043-X.

[113] Richard E. Korf. "Multi-Way Number Partitioning". In: *Proc. IJCAI*. Pasadena, CA, USA, 2009, pp. 538–543. URL: https://www.ijcai.org/Proceedings/09/Papers/096.pdf.

[114] Richard E. Korf. "Objective Functions for Multi-Way Number Partitioning". In: *Proc. SoCS*. Atlanta, GA, USA, 2010, pp. 71–72. DOI: 10.1609/socs.v1i1.18172.

[115] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. "Estimating mutual information". In: *Phys. Rev. E* 69.6 (2004). DOI: 10.1103/PhysRevE.69.066138.

[116] Jan Kwakkel. "The Exploratory Modeling Workbench: An open source toolkit for exploratory modeling, scenario discovery, and (multi-objective) robust decision making". In: *Environ. Modell. Software* 96 (2017), pp. 239–250. DOI: 10.1016/j.envsoft.2017.06.054.

[117] Jan H. Kwakkel and Scott C. Cunningham. "Improving scenario discovery by bagging random boxes". In: *Technol. Forecasting Social Change* 111 (2016), pp. 124–134. DOI: 10.1016/j.techfore.2016.06.014.

[118] Gabriel Laberge et al. "Partial Order in Chaos: Consensus on Feature Attributions in the Rashomon Set". In: *J. Mach. Learn. Res.* 24.364 (2023). URL: http://jmlr.org/papers/v24/23-0149.html.

[119] Vincenzo Lagani et al. "Feature Selection with the R Package MXM: Discovering Statistically Equivalent Feature Subsets". In: *J. Stat. Software* 80.7 (2017). DOI: 10.18637/jss.v080.i07.

[120] Nada Lavrač, Peter Flach, and Blaz Zupan. "Rule Evaluation Measures: A Unifying View". In: *Proc. ILP*. Bled, Slovenia, 1999, pp. 174–185. DOI: 10.1007/3-540-48751-4_17.

[121] Nada Lavrač and Dragan Gamberger. "Relevancy in Constraint-Based Subgroup Discovery". In: *Proc. European Workshop on Inductive Databases and Constraint Based Mining*. Hinterzarten, Germany, 2006, pp. 243–266. DOI: 10.1007/11615576_12.

[122] Nada Lavrač et al. "Subgroup discovery with CN2-SD". In: *J. Mach. Learn. Res.* 5.Feb (2004), pp. 153–188. URL: https://www.jmlr.org/papers/v5/lavrac04a.html.

[123] Alexander Lawrinenko. “Identical Parallel Machine Scheduling Problems: Structural patterns, bounding techniques and solution procedures”. PhD thesis. Friedrich-Schiller-Universität Jena, 2017. URL: https://nbn-resolving.org/urn:nbn:de:gbv:27-dbt-20170427-0956483.

[124] Alexander Lawrinenko, Stefan Schwerdfeger, and Rico Walter. “Reduction criteria, upper bounds, and a dynamic programming based heuristic for the max–min k_i-partitioning problem”. In: *J. Heuristics* 24.2 (2018), pp. 173–203. DOI: 10.1007/s10732-017-9362-9.

[125] Jaesung Lee, Wangduk Seo, and Dae-Won Kim. “Effective Evolutionary Multilabel Feature Selection under a Budget Constraint”. In: *Complexity* 2018 (2018). DOI: 10.1155/2018/3241489.

[126] Matthijs van Leeuwen and Arno Knobbe. “Diverse subgroup set discovery”. In: *Data Min. Knowl. Disc.* 25.2 (2012), pp. 208–242. DOI: 10.1007/s10618-012-0273-y.

[127] Matthijs van Leeuwen and Antti Ukkonen. “Discovering Skylines of Subgroup Sets”. In: *Proc. ECML PKDD*. Prague, Czech Republic, 2013, pp. 272–287. DOI: 10.1007/978-3-642-40994-3_18.

[128] Florian Lemmerich, Martin Atzmueller, and Frank Puppe. “Fast exhaustive subgroup discovery with numerical target concepts”. In: *Data Min. Knowl. Disc.* 30.3 (2016), pp. 711–762. DOI: 10.1007/s10618-015-0436-8.

[129] Florian Lemmerich and Martin Becker. “pysubgroup: Easy-to-Use Subgroup Discovery in Python”. In: *Proc. ECML PKDD*. Dublin, Ireland, 2019, pp. 658–662. DOI: 10.1007/978-3-030-10997-4_46.

[130] Florian Lemmerich, Mathias Rohlfs, and Martin Atzmueller. “Fast Discovery of Relevant Subgroup Patterns”. In: *Proc. FLAIRS*. Daytona Beach, FL, USA, 2010, pp. 428–433. URL: https://aaai.org/papers/flairs-2010-1262/.

[131] Chu Min Li and Felip Manyà. “MaxSAT, Hard and Soft Constraints”. In: *Handbook of Satisfiability*. 2nd ed. IOS Press, 2021. Chap. 23, pp. 903–927. DOI: 10.3233/FAIA201007.

[132] Jundong Li et al. “Feature Selection: A Data Perspective”. In: *ACM Comput. Surv.* 50.6 (2017). DOI: 10.1145/3136625.

[133] Rui Li et al. “Efficient redundancy reduced subgroup discovery via quadratic programming”. In: *J. Intell. Inf. Syst.* 44.2 (2015), pp. 271–288. DOI: 10.1007/s10844-013-0284-1.

[134] Kai Liu and Jin Tian. “Subspace Learning with an Archive-Based Genetic Algorithm”. In: *Proc. IEEM*. Changsha, China, 2018, pp. 181–188. DOI: 10.1007/978-981-13-3402-3_20.

[135] Yue Liu et al. “Multi-Layer Feature Selection Incorporating Weighted Score-Based Expert Knowledge toward Modeling Materials with Targeted Properties”. In: *Adv. Theor. Simul.* 3.2 (2020). DOI: 10.1002/adts.201900215.

[136] Antonio Lopez-Martinez-Carrasco et al. “Discovering Diverse Top-K Characteristic Lists”. In: *Proc. IDA*. Louvain-la-Neuve, Belgium, 2023, pp. 262–273. DOI: 10.1007/978-3-031-30047-9_21.

[137] Quentin Louveaux and Sébastien Mathieu. “A combinatorial branch-and-bound algorithm for box search”. In: *Discrete Optim.* 13 (2014), pp. 36–48. DOI: 10.1016/j.disopt.2014.05.001.

[138] Guangquan Lu et al. “Unsupervised feature selection with graph learning via low-rank constraint”. In: *Multimed. Tools Appl.* 77.22 (2018), pp. 29531–29549. DOI: 10.1007/s11042-017-5207-7.

[139] Tarcísio Lucas, Renato Vimieiro, and Teresa Ludermir. “SSDP+: A Diverse and More Informative Subgroup Discovery Approach for High Dimensional Data”. In: *Proc. CEC*. Rio de Janeiro, Brazil, 2018. DOI: 10.1109/CEC.2018.8477855.

[140] Scott M. Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Proc. NIPS*. Long Beach, CA, USA, 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.

[141] Michael Mampaey et al. “Efficient Algorithms for Finding Richer Subgroup Descriptions in Numeric and Nominal Data”. In: *Proc. ICDM*. Brussels, Belgium, 2012, pp. 499–508. DOI: 10.1109/ICDM.2012.117.

[142] Ankita Mangal and Elizabeth A. Holm. “A Comparative Study of Feature Selection Methods for Stress Hotspot Classification in Materials”. In: *Integr. Mater. Manuf. Innovation* 7.3 (2018), pp. 87–95. DOI: 10.1007/s40192-018-0109-8.

[143] Romain Mathonat et al. “Anytime Subgroup Discovery in High Dimensional Numerical Data”. In: *Proc. DSAA*. Porto, Portugal, 2021. DOI: 10.1109/DSAA53316.2021.9564223.

[144] Federico Matteucci, Vadim Arzamasov, and Klemens Böhm. “A benchmark of categorical encoders for binary classification”. In: *Proc. NeurIPS*. New Orleans, LA, USA, 2023, pp. 54855–54875. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/ac01e21bb14609416760f790dd8966ae-Abstract-Datasets_and_Benchmarks.html.

[145] Brian W. Matthews. “Comparison of the predicted and observed secondary structure of T4 phage lysozyme”. In: *Biochim. Biophys. Acta - Protein Struct.* 405.2 (1975), pp. 442–451. DOI: 10.1016/0005-2795(75)90109-9.

[146] Marvin Meeng, Wouter Duivesteijn, and Arno Knobbe. “ROCsearch – An ROC-guided Search Strategy for Subgroup Discovery”. In: *Proc. SDM*. Philadelphia, PA, USA, 2014, pp. 704–712. DOI: 10.1137/1.9781611973440.81.

[147] Marvin Meeng and Arno Knobbe. “For real: a thorough look at numeric attributes in subgroup discovery”. In: *Data Min. Knowl. Disc.* 35.1 (2021), pp. 158–212. DOI: 10.1007/s10618-020-00703-x.

[148] Marcilio Mendonca, Moises Branco, and Donald Cowan. "S.P.L.O.T.: Software Product Lines Online Tools". In: *Proc. OOPSLA*. Orlando, FL, USA, 2009, pp. 761–762. DOI: 10.1145/1639950.1640002.

[149] Wil Michiels et al. "Computer-assisted proof of performance ratios for the Differencing Method". In: *Discrete Optim.* 9.1 (2012), pp. 1–16. DOI: 10.1016/j.disopt.2011.10.001.

[150] Matej Mihelčić and Adrian Satja Kurdija. "On the complexity of redescription mining". In: *Theor. Comput. Sci.* 944 (2023). DOI: 10.1016/j.tcs.2022.12.023.

[151] Matthias Mnich and René van Bevern. "Parameterized complexity of machine scheduling: 15 open problems". In: *Comput. Oper. Res.* 100 (2018), pp. 254–261. DOI: 10.1016/j.cor.2018.07.020.

[152] Kiarash Mohammadi et al. "Scaling Guarantees for Nearest Counterfactual Explanations". In: *Proc. AIES*. Virtual Event, USA, 2021, pp. 177–187. DOI: 10.1145/3461702.3462514.

[153] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. "Interpretable Machine Learning – A Brief History, State-of-the-Art and Challenges". In: *Proc. ECML PKDD Workshops*. Ghent, Belgium, 2020, pp. 417–431. DOI: 10.1007/978-3-030-65965-3_28.

[154] Niloofar Momeni et al. "CAFS: Cost-Aware Features Selection Method for Multimodal Stress Monitoring on Wearable Devices". In: *IEEE Trans. Biomed. Eng.* 69.3 (2021), pp. 1072–1084. DOI: 10.1109/TBME.2021.3113593.

[155] MOSEK ApS. *MOSEK Modeling Cookbook : Mixed integer optimzation*. Accessed: 2024-06-27. 2024. URL: https://docs.mosek.com/modeling-cookbook/mio.html.

[156] Ramaravind K. Mothilal, Amit Sharma, and Chenhao Tan. "Explaining Machine Learning Classifiers through Diverse Counterfactual Explanations". In: *Proc. FAT**. Barcelona, Spain, 2020, pp. 607–617. DOI: 10.1145/3351095.3372850.

[157] Emmanuel Müller et al. "Relevant Subspace Clustering: Mining the Most Interesting Non-Redundant Concepts in High Dimensional Data". In: *Proc. ICDM*. Miami Beach, FL, USA, 2009, pp. 377–386. DOI: 10.1109/ICDM.2009.10.

[158] Inga M. Müller. "Feature selection for energy system modeling: Identification of relevant time series information". In: *Energy AI* 4 (2021). DOI: 10.1016/j.egyai.2021.100057.

[159] Roger B. Myerson. "Utilitarianism, Egalitarianism, and the Timing Effect in Social Choice Problems". In: *Econometrica* 49.4 (1981), pp. 883–897. DOI: 10.2307/1912508.

[160] Nina Narodytska et al. "Learning Optimal Decision Trees with SAT". In: *Proc. IJCAI*. Stockholm, Sweden, 2018, pp. 1362–1368. DOI: 10.24963/ijcai.2018/189.

[161] Felix Neutatz, Felix Biessmann, and Ziawasch Abedjan. "Enforcing Constraints for Machine Learning Systems via Declarative Feature Selection: An Experimental Study". In: *Proc. SIGMOD*. Virtual Event, China, 2021, pp. 1345–1358. DOI: 10.1145/3448016.3457295.

[162] Felix Neutatz, Marius Lindauer, and Ziawasch Abedjan. "AutoML in heavily constrained applications". In: *VLDB J.* (2023). DOI: 10.1007/s00778-023-00820-1.

[163] Raymond T. Ng et al. "Exploratory Mining and Pruning Optimizations of Constrained Associations Rules". In: *ACM SIGMOD Rec.* 27.2 (1998), pp. 13–24. DOI: 10.1145/276305.276307.

[164] Hai Nguyen, Katrin Franke, and Slobodan Petrović. "Optimizing a Class of Feature Selection Measures". In: *Proc. DISCML*. Vancouver, BC, Canada, 2009. URL: https://www.researchgate.net/publication/231175763.

[165] Hai Thanh Nguyen, Katrin Franke, and Slobodan Petrović. "Towards a Generic Feature-Selection Measure for Intrusion Detection". In: *Proc. ICPR*. Istanbul, Turkey, 2010, pp. 1529–1532. DOI: 10.1109/ICPR.2010.378.

[166] Hoang Vu Nguyen, Emmanuel Müller, and Klemens Böhm. "4S: Scalable Subspace Search Scheme Overcoming Traditional Apriori Processing". In: *Proc. Big Data*. Santa Clara, CA, USA, 2013, pp. 359–367. DOI: 10.1109/BigData.2013.6691596.

[167] Xuan Vinh Nguyen et al. "Effective Global Approaches for Mutual Information Based Feature Selection". In: *Proc. KDD*. New York, NY, USA, 2014, pp. 512–521. DOI: 10.1145/2623330.2623611.

[168] Robert Nieuwenhuis and Albert Oliveras. "On SAT Modulo Theories and Optimization Problems". In: *Proc. SAT*. Seattle, WA, USA, 2006, pp. 156–169. DOI: 10.1007/11814948_18.

[169] Uchechukwu F. Njoku et al. "Wrapper Methods for Multi-Objective Feature Selection". In: *Proc. EDBT*. Ioannina, Greece, 2023, pp. 697–709. DOI: 10.48786/edbt.2023.58.

[170] Eugene Nudelman et al. "Understanding Random SAT: Beyond the Clauses-to-Variables Ratio". In: *Proc. CP*. Toronto, ON, Canada, 2004, pp. 438–452. DOI: 10.1007/978-3-540-30201-8_33.

[171] Lina Ochoa et al. "Constraint programming heuristics for configuring optimal products in multi product lines". In: *Inf. Sci.* 474 (2019), pp. 33–47. DOI: 10.1016/j.ins.2018.09.042.

[172] Jeho Oh et al. "Finding Near-Optimal Configurations in Product Lines by Random Sampling". In: *Proc. ESEC/FSE*. Paderborn, Germany, 2017, pp. 61–71. DOI: 10.1145/3106237.3106273.

[173] Randal S. Olson et al. "PMLB: a large benchmark suite for machine learning evaluation and comparison". In: *BioData Min.* 10 (2017). DOI: 10.1186/s13040-017-0154-4.

[174] Pavel Paclík et al. "On Feature Selection with Measurement Cost and Grouped Features". In: *Proc. SSPR /SPR*. Windsor, ON, Canada, 2002, pp. 461–469. DOI: 10.1007/3-540-70659-3_48.

[175] Laxmi Parida and Naren Ramakrishnan. "Redescription Mining: Structure Theory and Algorithms". In: *Proc. AAAI*. Pittsburgh, PA, USA, 2005, pp. 837–844. URL: https://aaai.org/papers/00837-aaai05-132-redescription-mining-structure-theory-and-algorithms/.

[176] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *J. Mach. Learn. Res.* 12.85 (2011), pp. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[177] Hanchuan Peng, Fuhui Long, and Chris Ding. "Feature Selection Based on Mutual Information Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 27.8 (2005), pp. 1226–1238. DOI: 10.1109/TPAMI.2005.159.

[178] Laurent Perron and Vincent Furnon. *OR-Tools*. Accessed: 2024-06-27. Google, 2024. URL: https://developers.google.com/optimization/.

[179] Jan H. Plasberg and W. Bastiaan Kleijn. "Feature Selection Under a Complexity Constraint". In: *IEEE Trans. Multimedia* 11.3 (2009), pp. 565–571. DOI: 10.1109/TMM.2009.2012944.

[180] Andrei Popescu et al. "An overview of machine learning techniques in constraint solving". In: *J. Intell. Inf. Syst.* 58.1 (2022), pp. 91–118. DOI: 10.1007/s10844-021-00666-5.

[181] Hugo M. Proença et al. "Robust subgroup discovery: Discovering subgroup lists using *MDL*". In: *Data Min. Knowl. Disc.* 36.5 (2022), pp. 1885–1970. DOI: 10.1007/s10618-022-00856-x.

[182] Naren Ramakrishnan et al. "Turning CARTwheels: An Alternating Algorithm for Mining Redescriptions". In: *Proc. KDD*. Seattle, WA, USA, 2004, pp. 266–275. DOI: 10.1145/1014052.1014083.

[183] Rampi Ramprasad et al. "Machine learning in materials informatics: recent applications and prospects". In: *npj Comput. Mater.* 3 (2017). DOI: 10.1038/s41524-017-0056-5.

[184] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?" Explaining the Predictions of Any Classifier". In: *Proc. KDD*. San Francisco, CA, USA, 2016, pp. 1135–1144. DOI: 10.1145/2939672.2939778.

[185] Irene Rodriguez-Lujan et al. "Quadratic Programming Feature Selection". In: *J. Mach. Learn. Res.* 11.49 (2010), pp. 1491–1516. URL: http://jmlr.org/papers/v11/rodriguez-lujan10a.html.

[186] Joseph D. Romano et al. *PMLB v1.0: An open source dataset collection for benchmarking machine learning methods*. arXiv:2012.00058v3 [cs.LG]. 2021. DOI: 10.48550/arXiv.2012.00058.

[187] Mehrdad Rostami, Kamal Berahmand, and Saman Forouzandeh. "A novel method of constrained feature selection by the measurement of pairwise constraints uncertainty". In: *J. Big Data* 7 (2020). DOI: 10.1186/s40537-020-00352-3.

[188] Chris Russell. "Efficient Search for Diverse Coherent Explanations". In: *Proc. FAT**. Atlanta, GA, USA, 2019, pp. 20–28. DOI: 10.1145/3287560.3287569.

[189] Yvan Saeys, Thomas Abeel, and Yves Van de Peer. "Robust Feature Selection Using Ensemble Feature Selection Techniques". In: *Proc. ECML PKDD*. Antwerp, Belgium, 2008, pp. 313–325. DOI: 10.1007/978-3-540-87481-2_21.

[190] Sartaj K. Sahni. "Algorithms for Scheduling Independent Tasks". In: *J. ACM* 23.1 (1976), pp. 116–127. DOI: 10.1145/321921.321934.

[191] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. "On the Value of User Preferences in Search-Based Software Engineering: A Case Study in Software Product Lines". In: *Proc. ICSE*. San Francisco, CA, USA, 2013, pp. 492–501. DOI: 10.1109/ICSE.2013.6606595.

[192] Thomas J. Schaefer. "The Complexity of Satisfiability Problems". In: *Proc. STOC*. San Diego, CA, USA, 1978, pp. 216–226. DOI: 10.1145/800133.804350.

[193] André Schidler and Stefan Szeider. "SAT-based Decision Tree Learning for Large Data Sets". In: *Proc. AAAI*. Virtual Conference, 2021, pp. 3904–3912. DOI: 10.1609/aaai.v35i5.16509.

[194] Martin Scholz. "Sampling-Based Sequential Subgroup Mining". In: *Proc. KDD*. Chicago, IL, USA, 2005, pp. 265–274. DOI: 10.1145/1081870.1081902.

[195] Ethan L. Schreiber, Richard E. Korf, and Michael D. Moffitt. "Optimal Multi-Way Number Partitioning". In: *J. ACM* 65.4 (2018), pp. 1–61. DOI: 10.1145/3184400.

[196] Borja Seijo-Pardo et al. "Ensemble feature selection: Homogeneous and heterogeneous approaches". In: *Knowl.-Based Syst.* 118 (2017), pp. 124–139. DOI: 10.1016/j.knosys.2016.11.017.

[197] Sebastiano B. Serpico and Lorenzo Bruzzone. "A New Search Algorithm for Feature Selection in Hyperspectral Remote Sensing Images". In: *IEEE Trans. Geosci. Remote Sens.* 39.7 (2001), pp. 1360–1367. DOI: 10.1109/36.934069.

[198] Pouya Shati, Eldan Cohen, and Sheila McIlraith. "SAT-Based Approach for Learning Optimal Decision Trees with Non-Binary Features". In: *Proc. CP*. Montpellier, France (Virtual Conference), 2021. DOI: 10.4230/LIPIcs.CP.2021.50.

[199] Stephen She. *Linux Variability Analysis Tools*. Accessed: 2024-06-27. 2024. URL: https://code.google.com/archive/p/linux-variability-analysis-tools/.

[200] Razieh Sheikhpour et al. "A survey on semi-supervised feature selection methods". In: *Pattern Recognit.* 64 (2017), pp. 141–158. DOI: 10.1016/j.patcog.2016.11.003.

[201] Arvind Kumar Shekar, Patricia Iglesias Sánchez, and Emmanuel Müller. "Diverse Selection of Feature Subsets for Ensemble Regression". In: *Proc. DaWaK*. Lyon, France, 2017, pp. 259–273. DOI: 10.1007/978-3-319-64283-3_19.

[202] Aditya A. Shrotri et al. "Constraint-Driven Explanations for Black-Box ML Models". In: *Proc. AAAI*. Virtual Conference, 2022, pp. 8304–8314. DOI: 10.1609/aaai.v36i8.20805.

[203] Umair F. Siddiqi, Sadiq M. Sait, and Okyay Kaynak. "Genetic Algorithm for the Mutual Information-Based Feature Selection in Univariate Time Series Data". In: *IEEE Access* 8 (2020), pp. 9597–9609. DOI: 10.1109/ACCESS.2020.2964803.

[204] Andreia Silva and Cláudia Antunes. "Constrained pattern mining in the new era". In: *Knowl. Inf. Syst.* 47.3 (2016), pp. 489–516. DOI: 10.1007/s10115-015-0860-5.

[205] Wilson Silva, Kelwin Fernandes, and Jaime S. Cardoso. "How to produce complementary explanations using an Ensemble model". In: *Proc. IJCNN*. Budapest, Hungary, 2019. DOI: 10.1109/IJCNN.2019.8852409.

[206] Noah Simon et al. "A Sparse-Group Lasso". In: *J. Comput. Graphical Stat.* 22.2 (2013), pp. 231–245. DOI: 10.1080/10618600.2012.681250.

[207] Carsten Sinz. "Towards an Optimal CNF Encoding of Boolean Cardinality Constraints". In: *Proc. CP*. Sitges, Spain, 2005, pp. 827–831. DOI: 10.1007/11564751_73.

[208] Gustavo Sosa-Cabrera et al. "Feature selection: a perspective on inter-attribute cooperation". In: *Int. J. Data Sci. Anal.* 17.2 (2024), pp. 139–151. DOI: 10.1007/s41060-023-00439-z.

[209] Ilia Stepin et al. "A Survey of Contrastive and Counterfactual Explanation Generation Methods for Explainable Artificial Intelligence". In: *IEEE Access* 9 (2021), pp. 11974–12001. DOI: 10.1109/ACCESS.2021.3051315.

[210] Markus Sudmanns et al. "Data-driven exploration and continuum modeling of dislocation networks". In: *Modell. Simul. Mater. Sci. Eng.* 28.6 (2020). DOI: 10.1088/1361-651x/ab97ef.

[211] Thomas Thum, Don Batory, and Christian Kastner. "Reasoning about Edits to Feature Models". In: *Proc. ICSE*. Vancouver, BC, Canada, 2009, pp. 254–264. DOI: 10.1109/ICSE.2009.5070526.

[212] Holger Trittenbach and Klemens Böhm. "Dimension-based subspace search for outlier detection". In: *Int. J. Data Sci. Anal.* 7.2 (2019), pp. 87–101. DOI: 10.1007/s41060-018-0137-7.

[213] Felix Ulrich-Oltean, Peter Nightingale, and James Alfred Walker. "Selecting SAT Encodings for Pseudo-Boolean and Linear Integer Constraints". In: *Proc. CP*. Haifa, Israel, 2022. DOI: 10.4230/LIPIcs.CP.2022.38.

[214] Joaquin Vanschoren et al. "OpenML: networked science in machine learning". In: *ACM SIGKDD Explor. Newsl.* 15.2 (2014), pp. 49–60. DOI: 10.1145/2641190.2641198.

[215] Sebastián Ventura and José María Luna. "Subgroup Discovery". In: *Supervised Descriptive Pattern Mining*. 1st ed. Springer, 2018. Chap. 4, pp. 71–98. DOI: 10.1007/978-3-319-98140-6_4.

[216] Sahil Verma et al. *Counterfactual Explanations for Machine Learning: A Review*. arXiv:2010.10596v3 [cs.LG]. 2022. DOI: 10.48550/arXiv.2010.10596.

[217] Sandra Wachter, Brent Mittelstadt, and Chris Russell. "Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR". In: *Harv. J. Law Technol.* 31.2 (2017), pp. 841–887. URL: https://jolt.law.harvard.edu/assets/articlePDFs/v31/Counterfactual-Explanations-without-Opening-the-Black-Box-Sandra-Wachter-et-al.pdf.

[218] Nicholas Wagner and James M. Rondinelli. "Theory-Guided Machine Learning in Materials Science". In: *Front. Mater. Sci.* 3 (2016). DOI: 10.3389/fmats.2016.00028.

[219] Rico Walter, Martin Wirth, and Alexander Lawrinenko. "Improved approaches to the exact solution of the machine covering problem". In: *J. Sched.* 20.2 (2017), pp. 147–164. DOI: 10.1007/s10951-016-0477-x.

[220] Danding Wang et al. "Designing Theory-Driven User-Centric Explainable AI". In: *Proc. CHI*. Glasgow, United Kingdom, 2019. DOI: 10.1145/3290605.3300831.

[221] Daniel Weygand et al. "Discrete dislocation modeling in three-dimensional confined volumes". In: *Mater. Sci. Eng., A* 309–310 (2001), pp. 420–424. DOI: 10.1016/S0921-5093(00)01632-4.

[222] Jules White et al. "Automated diagnosis of feature model configurations". In: *J. Syst. Software* 83.7 (2010), pp. 1094–1107. DOI: 10.1016/j.jss.2010.02.017.

[223] Wallace Alvin Wilson. "On Semi-Metric Spaces". In: *Am. J. Math.* 53.2 (1931), pp. 361–373. DOI: 10.2307/2370790.

[224] Gerhard J. Woeginger. "A polynomial-time approximation scheme for maximizing the minimum machine completion time". In: *Oper. Res. Lett.* 20.4 (1997), pp. 149–154. DOI: 10.1016/S0167-6377(96)00055-7.

[225] Adam Woznica, Phong Nguyen, and Alexandros Kalousis. "Model Mining for Robust Feature Selection". In: *Proc. KDD*. Beijing, China, 2012, pp. 913–921. DOI: 10.1145/2339530.2339674.

[226] Haiqin Yang et al. "Budget constrained non-monotonic feature selection". In: *Neural Networks* 71 (2015), pp. 214–224. DOI: 10.1016/j.neunet.2015.08.004.

[227] Jinqiang Yu et al. "Learning Optimal Decision Sets and Lists with SAT". In: *J. Artif. Intell. Res.* 72 (2021), pp. 1251–1279. DOI: 10.1613/jair.1.12719.

[228] Lei Yu and Huan Liu. "Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution". In: *Proc. ICML*. Washington DC, USA, 2003, pp. 856–863. URL: https://aaai.org/papers/icml03-111-feature-selection-for-high-dimensional-data-a-fast-correlation-based-filter-solution/.

[229] Ming Yuan and Yi Lin. "Model selection and estimation in regression with grouped variables". In: *J. R. Stat. Soc. B* 68.1 (2006), pp. 49–67. DOI: 10.1111/j.1467-9868.2005.00532.x.

[230] Daoqiang Zhang, Songcan Chen, and Zhi-Hua Zhou. "Constraint Score: A new filter method for feature selection with pairwise constraints". In: *Pattern Recognit.* 41.5 (2008), pp. 1440–1451. DOI: 10.1016/j.patcog.2007.10.009.

[231] Jilian Zhang, Kyriakos Mouratidis, and HweeHwa Pang. "Heuristic Algorithms for Balanced Multi-Way Number Partitioning". In: *Proc. IJCAI*. Barcelona, Spain, 2011, pp. 693–698. DOI: 10.5591/978-1-57735-516-8/IJCAI11-122.

[232] Leyuan Zhang et al. "Nonlinear sparse feature selection algorithm via low matrix rank constraint". In: *Multimed. Tools Appl.* 78.23 (2019), pp. 33319–33337. DOI: 10.1007/s11042-018-6909-1.

[233] Rui Zhang, Yunxing Zhang, and Xuelong Li. "Unsupervised Feature Selection via Adaptive Graph Learning and Constraint". In: *IEEE Trans. Neural Networks Learn. Syst.* 33.3 (2020), pp. 1355–1362. DOI: 10.1109/TNNLS.2020.3042330.

[234] Zhiwei Zhang et al. "Learning for Efficient Supervised Query Expansion via Two-stage Feature Selection". In: *Proc. SIGIR*. Pisa, Italy, 2016, pp. 265–274. DOI: 10.1145/2911451.2911539.

[235] Peng Zhao, Guilherme Rocha, and Bin Yu. *Grouped and Hierarchical Model Selection through Composite Absolute Penalties*. Tech. rep. Department of Statistics, UC Berkeley, 2006. URL: https://statistics.berkeley.edu/sites/default/files/tech-reports/703.pdf.

A. Appendix

In this chapter, we provide supplementary materials for Chapter 6 (cf. Section A.1) and Chapter 7 (cf. Section A.2).

A.1. Finding Alternative Feature Sets

In this section, we provide supplementary materials for Chapter 6. Section A.1.1 provides complete definitions of the alternative-feature-selection problem (cf. Section 6.2.2) for the univariate objective (cf. Equation 2.1). Section A.1.2 contains proofs for the complexity analysis with the univariate objective (cf. Section 6.2.4.2).

A.1.1. Complete Optimization Problems for the Univariate Objective

In this section, we provide complete specifications of the alternative-feature-selection problem for sequential and simultaneous search as a 0-1 integer linear problem. In particular, we combine all relevant definitions and equations from Section 6.2. We use the objective of univariate filter feature selection (cf. Equation 2.1). The corresponding feature qualities $q(\cdot)$ are constants in the optimization problem. Further, we use the Dice dissimilarity (cf. Equations 6.3 and 6.8) to measure feature-set dissimilarity for alternatives. The dissimilarity threshold $\tau \in [0, 1]$ is a user-defined constant. Finally, we assume fixed, user-defined feature-set sizes $k \in \mathbb{N}$.

Sequential-search problem In the sequential case (cf. Definition 7 and Equation 6.9), only one feature set F_s is variable in the optimization problem, while the existing feature sets $F_{\bar{s}} \in \mathbb{F}$ with their selection vectors $\bar{s}$ are constants.

$$
\begin{aligned}
\max_{s} \quad & Q_{\text{uni}}(s, X, y) = \sum_{j=1}^{n} q(X_{\cdot j}, y) \cdot s_j \\
\text{subject to:} \quad \forall F_{\bar{s}} \in \mathbb{F}: \quad & \sum_{j=1}^{n} s_j \cdot \bar{s}_j \leq (1 - \tau) \cdot k \\
& \sum_{j=1}^{n} s_j = k \\
& s \in \{0, 1\}^n
\end{aligned}
\tag{A.1}
$$

Simultaneous-search problem In the simultaneous case (cf. Definition 9 and Equation 6.10), all feature sets are variable. $a \in \mathbb{N}_0$ denotes the number of alternatives, which corresponds to the number of feature sets minus one. Next, we introduce auxiliary variables to linearize products between decision variables (cf. Equation 6.6). Finally, we use sum-aggregation (cf. Equation 6.11) over alternatives in the objective here.

$$
\begin{aligned}
\max_{s^{(0)},\ldots,s^{(a)}} \quad & \sum_{l} Q_{\text{uni}}(s^{(l)}, X, y) = \sum_{l} \sum_{j} q(X_{\cdot j}, y) \cdot s_j^{(l)} \\
\text{subject to:} \quad \forall l_1\ \forall l_2 : \quad & \sum_{j} t_j^{(l_1,l_2)} \leq (1-\tau) \cdot k \\
\forall l_1\ \forall l_2\ \forall j : \quad & t_j^{(l_1,l_2)} \leq s_j^{(l_1)} \\
\forall l_1\ \forall l_2\ \forall j : \quad & t_j^{(l_1,l_2)} \leq s_j^{(l_2)} \\
\forall l_1\ \forall l_2\ \forall j : \quad & 1 + t_j^{(l_1,l_2)} \geq s_j^{(l_1)} + s_j^{(l_2)} \\
\forall l : \quad & \sum_{j} s_j^{(l)} = k \\
\forall l : \quad & s^{(l)} \in \{0,1\}^n \\
\forall l_1\ \forall l_2 : \quad & t^{(l_1,l_2)} \in \{0,1\}^n \\
\text{with indices:} \quad & l \in \{0,\ldots,a\} \\
& l_1 \in \{1,\ldots,a\} \\
& l_2 \in \{0,\ldots,l_1-1\} \\
& j \in \{1,\ldots,n\}
\end{aligned}
\tag{A.2}
$$

A.1.2. Proofs

In this section, we provide proofs for propositions from Section 6.2.4.2, i.e., complexity results for alternative feature selection with univariate feature qualities.

A.1.2.1. Proof of Proposition 14

Proof. Let I be an arbitrary problem instance of the simultaneous-search problem with min-aggregation, univariate feature qualities, a complete-partitioning scenario, and a fixed feature-set size k (cf. Proposition 13). We add a new feature f' to I and keep the parameters a, k, and τ as before, obtaining an instance I' of the incomplete-partitioning scenario since one feature will not be selected. We set the quality q' of f' to be lower than all other feature qualities in I. Since the univariate objective monotonically increases in the selected feature qualities, selecting feature f' in a solution of I' does not have any benefit since f' would replace a feature with higher quality. If f' is not selected, this solution of I' also solves I. However, if the qualities of the alternatives are not equal, f' might still be chosen in a set that does not have the minimum quality of all sets since only the latter determines the objective value (cf. Example 2). In that case, we replace f' with the remaining unselected

feature; the objective value remains the same, and the solution becomes valid for I. Thus, in any case, we can easily transform a solution for I' to a solution for I.

Overall, an algorithm for incomplete partitioning can solve arbitrary complete-partitioning instances with negligible computational overhead. Thus, a polynomial-time algorithm for incomplete partitioning could also solve complete partitioning polynomially. However, the latter problem is $\mathcal{NP}$-complete (cf. Proposition 13), so incomplete partitioning has to be $\mathcal{NP}$-hard. Since checking a solution for incomplete partitioning needs only polynomial time, we obtain membership in $\mathcal{NP}$ and thereby $\mathcal{NP}$-completeness. □

A.1.2.2. Proof of Proposition 15

Proof. Let I be an arbitrary problem instance of the simultaneous-search problem with min-aggregation, univariate feature qualities, a complete-partitioning scenario, the Dice dissimilarity (cf. Equation 6.3) as $d(\cdot)$, and a fixed feature-set size k (cf. Proposition 13). We create a new problem instance I' by adding a new feature f' and increasing the feature-set size to $k' = k + 1$. Further, we set $\tau' = (k' - 1)/k'$, thereby allowing an overlap of at most one feature between feature sets. Also, we choose f' to have a considerably higher quality q' than all other features. Our goal is to force the selection of f' in all feature sets, no matter which other features are selected. One possible choice is $q' = \sum_{j=1}^{n} q_j + \varepsilon$ for a small $\varepsilon \in \mathbb{R}_{>0}$ This quality q' of f' is higher than of any feature set not containing it. Thus, a solution for I' contains f' in each feature set, while the remaining features are part of exactly one feature set. Hence, we can remove f' to get feature sets of size $k = k' - 1$ that constitute an optimal solution for the original problem instance I.

This transformation shows how an algorithm for problem instances with $\tau < 1$ can help solve arbitrary problem instances with $\tau = 1$. Given the $\mathcal{NP}$-completeness of the latter problem (cf. Proposition 13), we obtain $\mathcal{NP}$-hardness of the former. □

One can transfer this reduction from $\tau' = (k' - 1)/k'$ to all other $\tau > 0$. In particular, for a given k, there is only a finite number of τ values leading to different set overlaps, e.g., $\tau = \{0, 1/k, \ldots, (k-1)/k, 1\}$ for the Dice dissimilarity. The proof for the highest overlap except $\tau = 0$ requires creating an instance I' with $\tau' = 1/k$ from an instance with $\tau = 1$. For this purpose, $k^2 - k$ features need to be added since $\tau' = k/k' = k/(k + k^2 - k) = 1/k$. I.e., k out of $k' = k^2$ features need to form a complete partitioning, while the remaining $k^2 - k$ features occur in each feature set and will be removed after solving I'. The number of features to be added is polynomial in k and thereby also polynomial in n.

A.1.2.3. Proof of Proposition 16

Proof. We discuss the simultaneous-search problem (cf. Definition 9) with sum-aggregation (cf. Equation 6.11) first. We leverage the monotonicity of the univariate objective with sum-aggregation. In particular, this objective cannot decrease when selecting features of higher quality. Thus, we order all features decreasingly by their quality, which yields

the complexity of $O(n \cdot \log n)$. Next, we pick features in this order without replacement and assign them to sets until we have the user-defined number of alternatives with the user-defined feature-set sizes. Apart from observing cardinality constraints, the actual assignment of the selected features to sets does not matter quality-wise since swapping features between sets does not change the summed objective. Thus, one can fill the feature sets in an arbitrary order. Each assignment runs in $O(1)$, e.g., using arrays to store feature-set membership, yielding $O(n)$ for all features. Without cardinality constraints, only the number of alternatives needs to be satisfied. Further, if all features need to be selected, i.e., for a complete partitioning, one need not sort the features. Finally, if only a small fraction of features needs to be selected, one might slightly improve complexity to $O(k \cdot n)$ by iteratively picking the maximum instead of sorting all qualities.

For the sequential-search problem (cf. Definition 7), we conduct the same quality-sorting procedure. In contrast to the simultaneous-search problem, the actual assignment of features to sets matters since the sets have an explicit order. In particular, each alternative should get the remaining highest-quality features until its user-defined size is reached. The complexity is still dominated by sorting and therefore $O(n \cdot \log n)$. □

A.2. Discovering Sparse and Alternative Subgroup Descriptions

In this section, we supplement Chapter 7. Section A.2.1 describes how to encode constrained subgroup discovery as a mixed integer linear program, complementing Section 7.3.1. Section A.2.2 contains proofs for complexity propositions from Section 7.3.

A.2.1. Encoding via Mixed Integer Linear Programming (MILP)

We start from the SMT formulation and introduce additional variables and constraints to linearize certain logical expressions.

Unconstrained subgroup discovery We keep all decision variables from the corresponding SMT formulation (cf. Equation 7.5): the binary variables b_i for subgroup membership and the real-valued bound variables lb_j and ub_j. The bound constraints (cf. Equation 7.3) remain unchanged as well. Further, we retain the optimization objective, which already is linear in b_i (cf. Equations 7.1 and 7.2). However, we need to linearize the logical AND ($\wedge$) operators in the definition of subgroup membership b_i (cf. Equation 7.4) by introducing auxiliary variables and further constraints. In particular, we supplement the variables $b \in \{0,1\}^m$ by $b^{\text{lb}} \in \{0,1\}^{m \times n}$ and $b^{\text{ub}} \in \{0,1\}^{m \times n}$. These new binary variables indicate whether a particular data object satisfies the lower or upper bound for a particular feature. Using linearization techniques for constraint satisfaction and AND operators from [155], we obtain the following set of constraints to replace Equation 7.4:

$$
\begin{aligned}
\forall i\, \forall j: \quad & X_{ij} + m_j \cdot b_{ij}^{\mathrm{lb}} \leq lb_j - \varepsilon_j \\
\forall i\, \forall j: \quad & lb_j \leq X_{ij} + M_j \cdot \left(1 - b_{ij}^{\mathrm{lb}}\right) \\
\forall i\, \forall j: \quad & ub_j + m_j \cdot b_{ij}^{\mathrm{ub}} \leq X_{ij} - \varepsilon_j \\
\forall i\, \forall j: \quad & X_{ij} \leq ub_j + M_j \cdot \left(1 - b_{ij}^{\mathrm{ub}}\right) \\
\forall i\, \forall j: \quad & b_i \leq b_{ij}^{\mathrm{lb}} \\
\forall i\, \forall j: \quad & b_i \leq b_{ij}^{\mathrm{ub}} \\
\forall i: \quad & \sum_{j=1}^{n} \left(b_{ij}^{\mathrm{lb}} + b_{ij}^{\mathrm{ub}}\right) \leq b_i + 2n - 1 \\
\text{with indices:} \quad & i \in \{1, \ldots, m\} \\
& j \in \{1, \ldots, n\}
\end{aligned}
\tag{A.3}
$$

The first two inequalities ensure that $b_{ij}^{\mathrm{lb}} = 1$ if and only if $lb_j \leq X_{ij}$. The following two inequalities relate b_{ij}^{ub} to $X_{ij} \leq ub_j$. The values $\varepsilon_j \in \mathbb{R}_{>0}$ are small constants that turn strict inequalities into non-strict ones since a MILP solver may only support the latter. One possible choice is to sort all unique feature values and take the minimum difference between two consecutive values in that order. The values $M_j \in \mathbb{R}_{>0}$ and $m_j \in \mathbb{R}_{<0}$ are large positive and negative constants, respectively. They help to connect real-valued and binary-valued expressions, compensating for the latter's smaller range. One choice for M_j is a value larger than the difference between the feature's minimum and maximum:

$$
\begin{aligned}
\forall j \in \{1, \ldots, n\} \quad & M_j := 2 \cdot \left(\max_{i \in \{1,\ldots,m\}} X_{ij} - \min_{i \in \{1,\ldots,m\}} X_{ij} \right) \\
\forall j \in \{1, \ldots, n\} \quad & m_j := 2 \cdot \left(\min_{i \in \{1,\ldots,m\}} X_{ij} - \max_{i \in \{1,\ldots,m\}} X_{ij} \right)
\end{aligned}
\tag{A.4}
$$

In particular, the difference between the subgroup's bounds and arbitrary feature values must be smaller than M_j and larger than m_j, unless the bounds are placed outside the feature's value range. Since the latter does not improve the subgroup's quality in any case, we prevent it with additional constraints on the bound variables lb_j and ub_j:

$$
\begin{aligned}
\forall j \in \{1, \ldots, n\} \quad & \min_{i \in \{1,\ldots,m\}} X_{ij} \leq lb_j \leq \max_{i \in \{1,\ldots,m\}} X_{ij} \\
\forall j \in \{1, \ldots, n\} \quad & \min_{i \in \{1,\ldots,m\}} X_{ij} \leq ub_j \leq \max_{i \in \{1,\ldots,m\}} X_{ij}
\end{aligned}
\tag{A.5}
$$

Finally, the last three inequalities in Equation A.3 tie b_{ij}^{lb} and b_{ij}^{ub} to b_i and linearize the logical AND ($\wedge$) operators from Equation 7.4. In particular, these constraints ensure that a data object is a member of the subgroup, i.e., $b_i = 1$, if and only if all feature values of the data object observe the bounds, i.e., all corresponding $b_{ij}^{\mathrm{lb}} = 1$ and $b_{ij}^{\mathrm{ub}} = 1$.

Feature-cardinality constraints The feature-cardinality constraint of the SMT formulation (cf. Equation 7.7) already is a linear expression in the feature-selection variables s_j.

However, the constraints defining s_j (cf. Equation 7.6) contain a logical OR ($\vee$) operator and comparison ($<$) expressions. We linearize these constraints as follows:

$$\begin{aligned}
\forall i\, \forall j: \quad & 1 - b_{ij}^{\text{lb}} \leq s_j^{\text{lb}} \\
\forall i\, \forall j: \quad & 1 - b_{ij}^{\text{ub}} \leq s_j^{\text{ub}} \\
\forall j: \quad & s_j^{\text{lb}} \leq s_j \\
\forall j: \quad & s_j^{\text{ub}} \leq s_j \\
\forall j: \quad & s_j \leq 2m - \sum_{i=1}^{m} \left(b_{ij}^{\text{lb}} + b_{ij}^{\text{ub}} \right) \\
\text{with indices:} \quad & i \in \{1, \ldots, m\} \\
& j \in \{1, \ldots, n\}
\end{aligned} \tag{A.6}$$

The first four inequalities ensure that a feature is selected, i.e., $s_j = 1$, if any data object's feature value lies outside the subgroup's bounds, i.e., any $b_{ij}^{\text{lb}} = 0$ or $b_{ij}^{\text{ub}} = 0$. The last inequality covers the other direction of the logical equivalence: If a feature is selected, then at least one data object's feature value lies outside the subgroup's bounds.

Alternative subgroup descriptions The objective function for alternative subgroup descriptions in the SMT formulation (cf. Equation 7.12) is already linear. We only need to replace the logical negation ($\neg$) operator:

$$\text{sim}_{\text{nHamm}}(b^{(a)}, b^{(0)}) = \frac{1}{m} \cdot \Bigg(\sum_{\substack{i \in \{1,\ldots,m\} \\ b_i^{(0)}=1}} b_i^{(a)} + \sum_{\substack{i \in \{1,\ldots,m\} \\ b_i^{(0)}=0}} \left(1 - b_i^{(a)}\right) \Bigg) \tag{A.7}$$

The same replacement also applies to the dissimilarity constraints (cf. Equation 7.13):

$$\forall l \in \{0, \ldots, a-1\}: \; \text{dis}_{\text{des}}(s^{(a)}, s^{(l)}) = \sum_{\substack{j \in \{1,\ldots,n\} \\ s_j^{(l)}=1}} \left(1 - s_j^{(a)}\right) \geq \min\left(\tau_{\text{abs}},\; k^{(l)}\right) \tag{A.8}$$

Otherwise, this expression already is linear as well.

Implementation Our published code (cf. Section 1.4) contains a MILP implementation for unconstrained and feature-cardinality-constrained subgroup discovery. We use the package *OR-Tools* [178] with *SCIP* [27] as the optimizer. However, in preliminary experiments, this implementation was (on average) slower than the SMT implementation or yielded worse subgroup quality in the same runtime. Further, it sometimes finished considerably after the prescribed timeout or ran out of memory after consuming dozens of gigabytes. Thus, we stuck to the SMT implementation for our main experiments (cf. Section 7.4.2).

A.2.2. Proofs

In this section, we provide proofs for propositions from Section 7.3, particularly for the complexity results for subgroup discovery with a feature-cardinality constraint and for searching alternative subgroup descriptions.

A.2.2.1. Proof of Proposition 26

Proof. Let I be an arbitrary instance of the decision problem Set Covering [103]. I consists of a set of elements $E = \{e_1, \ldots, e_m\}$, a set of sets $\mathbb{S} = \{S_1, \ldots, S_n\}$ with $E = \bigcup_{S \in \mathbb{S}} S$, and a cardinality threshold $k \in \mathbb{N}$. Set Covering asks whether a subset $\mathbb{C} \subseteq \mathbb{S}$ exists with $|\mathbb{C}| \leq k$ and $E = \bigcup_{S \in \mathbb{C}} S$, i.e., a subset of $\mathbb{S}$ which contains (= covers) each element from E in at least one set and consist of at most k sets.

We transform I into an instance I' of the perfect-subgroup-discovery problem (cf. Definition 13) with a feature-cardinality constraint (cf. Definition 15). To this end, we define a binary dataset $X \in \{0,1\}^{(m+1)\times n}$, prediction target $y \in \{0,1\}^{m+1}$, and retain the number of sets $k \in \mathbb{N}$ as feature cardinality k. In particular, data objects represent elements from E, and features represent sets from $\mathbb{S}$. X_{ij} denotes $e_i \in S_j$, i.e., membership of Element i in Set j. The additional index $i = m+1$ represents a *dummy element* that is not part of any set, so all its feature values are set to 0. Further, we define the prediction target $y \in \{0,1\}^{m+1}$ as $y_{m+1} = 1$ and $y_i = 0$ for all other indices $i \in \{1, \ldots, m\}$. This prediction target represents whether an element should *not* be covered by the set of sets $\mathbb{C} \subseteq \mathbb{S}$. In particular, all elements from E should be covered but not the new dummy element. This 'inverted' definition of the prediction target stems from the different nature of set covers and subgroup descriptions: Set covers include elements from selected sets, with the empty cover $\mathbb{C} = \emptyset$ containing no elements. There is a logical OR ($\vee$) respectively set union over the selected sets. In contrast, subgroup descriptions exclude data objects based on bounds for their selected features, with the unrestricted subgroup containing all data objects. There is a logical AND ($\wedge$) over the features' bounds.

A perfect subgroup (cf. Definition 12) exactly replicates the prediction target y as subgroup membership. Here, it only contains the data object representing the dummy element but no data objects representing elements from E. As all feature values of this dummy data object are 0, the subgroup description only consists of the bounds $lb_j = ub_j = 0$ for selected features and $lb_j = 0 < 1 = ub_j$ for unselected features. Therefore, the data objects described by the selected features represent elements not contained in any selected set, which only applies to the dummy element. Vice versa, all remaining data objects represent elements that are part of at least one selected set, which applies to all elements from E. Further, the feature-cardinality constraint (cf. Definition 15) ensures that at most k features are selected, which means that at most k sets are selected. Thus, the selected features of a perfect subgroup represent sets forming a valid set cover $\mathbb{C}$. In contrast, if no feature set of the desired size k can describe a perfect subgroup, then at least one data object with prediction target $y_i = 0$ has to be a subgroup member. Thus, at least one element is not contained in any set forming the set cover, so no valid set cover of size k exists.

Overall, a solution to the instance I' of the perfect-subgroup discovery problem (cf. Definition 13) with a feature-cardinality constraint (cf. Definition 15) also solves the instance I of Set Covering [103] with negligible computational overhead. In particular, an efficient algorithm for the former would also efficiently solve the latter. However, since the latter problem is $\mathcal{NP}$-hard [103], the former is as well. To be more precise, perfect-subgroup discovery with a feature-cardinality constraint resides in $\mathcal{NP}$ and therefore is $\mathcal{NP}$-complete. In particular, checking a solution induces a polynomial cost of $O(m \cdot n)$, requiring one pass over the dataset to determine subgroup membership and feature selection. □

This proof adapts the proof of [29] for minimizing the feature cardinality of a given subgroup description. The latter reduces from the optimization problem Minimum Set Cover, while we use the decision problem Set Covering. Further, we replace the notion of a given subgroup description [29] with the notion of a perfect subgroup and employ lower and upper bounds instead of 'feature=value' conditions. The latter difference is irrelevant for binary datasets, where selected features have $lb_j = ub_j$ and thereby implicitly select a feature value. The hardness result naturally extends to real-valued datasets.

Note that the hardness reduction does not work for the special case $k = n$. For Set Covering, this case allows all sets to be selected, which trivially solves the problem. Vice versa, the unconstrained problem of perfect-subgroup discovery (cf. Definition 13) admits a polynomial-time solution (cf. Proposition 21).

A.2.2.2. Proof of Proposition 27

Proof. Let I be an arbitrary instance of the perfect-subgroup-discovery problem (cf. Definition 13) with a feature-cardinality constraint (cf. Definition 15). We transform I into an instance I' of the subgroup-discovery problem (cf. Definition 3) with the same constraint. In particular, we define the objective as optimizing subgroup quality $Q(lb, ub, X, y)$ rather than searching for a perfect subgroup (cf. Definition 12) that may or may not exist. The other inputs of the problem instance (X, y, and k) remain the same.

Based on the assumption we made on $Q(lb, ub, X, y)$ in Proposition 27, the optimal solution for I' is a perfect subgroup if the latter exists. Thus, if the optimal subgroup for I' is not perfect, then a perfect subgroup does not exist at all. Checking whether a subgroup is perfect entails a cost of $O(m \cdot n)$, i.e., computing subgroup membership and checking for false positives and false negatives. Overall, an algorithm for subgroup discovery (cf. Definition 3) with a feature-cardinality constraint (cf. Definition 15) solves perfect-subgroup discovery (cf. Definition 13) with the same constraint with negligible overhead. Since the latter problem is $\mathcal{NP}$-complete (cf. Proposition 26) and the former resides in the complexity class $\mathcal{NP}$, the former is $\mathcal{NP}$-complete as well. □

Alternatively, one could reduce from the optimization problem Maximum Coverage [38], similar to the proof for Proposition 26 (cf. Section A.2.2.1), which reduces from the decision problem Set Covering [103]. In contrast to Set Covering, the $k \in \mathbb{N}$ selected subsets in Maximum Coverage need not cover all elements but should cover as many elements as

possible. In subgroup discovery, the latter objective corresponds to a particular notion of subgroup quality: maximizing the number of true negatives or minimizing the number of false positives, i.e., excluding as many negative data objects from the subgroup as possible. This problem is minimal-optimal-recall-subgroup discovery (cf. Definition 11). However, the latter's objective is simpler than WRAcc (cf. Equation 2.3), which we focus on in this dissertation. Our proof above is more general regarding the notion of subgroup quality but more narrow in the sense that it reduces from a search problem, assuming a particular value of the objective function, instead of an optimization problem.

A.2.2.3. Proof of Proposition 30

Proof. Let I be an arbitrary instance of the perfect-subgroup-discovery problem (cf. Definition 13) with a feature-cardinality constraint (cf. Definition 15). We transform I into an instance I' of the perfect-alternative-subgroup-description-discovery problem (cf. Definition 18) with the same constraint. We retain the feature-cardinality threshold $k \in \mathbb{N}$ and slightly modify the dataset $X \in \mathbb{R}^{m \times n}$, as explained later.

Based on the assumptions from Proposition 30, we define the original subgroup for I' to be perfect (cf. Definition 12), i.e., having subgroup membership $b^{(0)} = y$. Also, we choose the dissimilarity threshold $\tau \in \mathbb{R}_{\geq 0}$ high enough that the alternative subgroup description may not select any features from the original subgroup description. E.g., we can choose the deselection dissimilarity (cf. Equation 7.11) and $\tau_{\text{abs}} = k$. We need not even explicitly define the original feature selection since we must not select these features in the alternative anyway. For the sake of completeness, we can define dataset $X' \in \mathbb{R}^{m \times (n+k)}$ of problem instance I' as dataset $X \in \mathbb{R}^{m \times n}$ of problem instance I with k features added that equal the prediction target y. Choosing the bounds $lb_j = ub_j = 1$ on any of these extra features produces the desired original subgroup membership $b^{(0)} = y$. We further assume that all extra features were selected in the original subgroup description but no actual features from X were, i.e., $\forall j \in \{1, \ldots, n\} : s_j^{(0)} = 0$ and $\forall j \in \{n+1, \ldots, n+k\} : s_j^{(0)} = 1$.

A solution for problem instance I' also solves I. In particular, the perfect alternative subgroup description (cf. Definition 17) is a perfect subgroup since it perfectly replicates the original subgroup membership, which is perfect, i.e., $b^{(a)} = b^{(0)} = y$. Due to the dissimilarity constraint, the alternative subgroup description only selects features from dataset X, not those newly added to create X'. Finally, both I and I' use a feature-cardinality constraint with threshold k. Thus, if a perfect alternative subgroup description for I' exists, it also solves I. If it does not exist, then there is also no other perfect subgroup for I.

Thus, an efficient algorithm for the perfect-alternative-subgroup-description-discovery problem (cf. Definition 18) with a feature-cardinality constraint (cf. Definition 15) would also efficiently solve perfect-subgroup discovery (cf. Definition 13) with the same constraint. However, we already established that the latter problem is $\mathcal{NP}$-complete (cf. Proposition 26). Further, evaluating a solution for the former problem entails a polynomial cost of $O(m \cdot n)$ for checking constraints regarding bounds, feature cardinality, and dissimilarity, placing the problem in complexity class $\mathcal{NP}$ and thereby making it $\mathcal{NP}$-complete. □

A.2.2.4. Proof of Proposition 31

Proof. Let I be an arbitrary instance of the perfect-alternative-subgroup-description-discovery problem (cf. Definition 18) with a feature-cardinality constraint (cf. Definition 15) and a perfect original subgroup (cf. Definition 12). We transform I into an instance I' of the same problem but with an imperfect original subgroup. In particular, we retain all problem inputs as-is except defining dataset $X' \in \mathbb{R}^{(m+1)\times n}$ of I' as dataset $X \in \mathbb{R}^{m\times n}$ of I plus an additional data object with label $y_{m+1} = 0$ but exactly the same feature values as an arbitrary existing data object $X_{i\cdot}$ with $y_i = 1$. In particular, this data object prevents the existence of a perfect subgroup. However, we assume this data object to be a member of the original subgroup, i.e., $b^{(0)}_{m+1} = 1$, while subgroup membership of all other data objects corresponds to their prediction target, i.e., $\forall i \in \{1, \ldots, m\} : \; b^{(0)}_i = y_i$.

If there is a solution for problem instance I', we can easily transform it to a solution for I. In particular, since the solution is a perfect alternative subgroup description (cf. Definition 17), it replicates $b^{(0)}$, i.e., assigns all positive data objects of I to the alternative subgroup and places all negative data objects of I outside. The additional data object is also a member of the alternative subgroup in I' but does not exist in I. Thus, the solution is a perfect subgroup for I. In contrast, if no solution for I' exists, then there is also no solution for I.

Overall, an efficient algorithm for perfect-alternative-subgroup-description discovery (cf. Definition 18) with a feature-cardinality constraint (cf. Definition 15) and an imperfect original subgroup (cf. Definition 12) could be adapted to efficiently solve this problem for a perfect original subgroup. However, since the latter problem is $\mathcal{NP}$-complete (cf. Proposition 30), the former, which resides in $\mathcal{NP}$ as well, is also $\mathcal{NP}$-complete. □

A.2.2.5. Proof of Proposition 32

Proof. Let I be an arbitrary instance of the perfect-alternative-subgroup-description-discovery problem (cf. Definition 18) with a feature-cardinality constraint (cf. Definition 15). We transform I into an instance I' of the alternative-subgroup-description-discovery problem (cf. Definition 16) with the same constraint. In particular, we define the objective as optimizing the subgroup-membership similarity $\mathrm{sim}(\cdot)$ rather than asking for a perfect alternative subgroup description (cf. Definition 17) that may or may not exist. The other inputs of the problem instance remain the same.

Based on the assumption on $\mathrm{sim}(\cdot)$ from Proposition 32, the optimal solution for I' is a perfect alternative subgroup description if the latter exists. Thus, if the optimal alternative subgroup description for I' is not a perfect alternative, then a perfect alternative subgroup description does not exist. Overall, an algorithm for alternative-subgroup-description discovery (cf. Definition 16) with a feature-cardinality constraint (cf. Definition 15) solves perfect-alternative-subgroup-description discovery (cf. Definition 18) with the same constraint with negligible overhead. Since the latter problem is $\mathcal{NP}$-complete (cf. Propositions 30 and 31) and the former resides in $\mathcal{NP}$, the former is also $\mathcal{NP}$-complete. □