Mining Exceptional Linear Regression Models with Tree-Constrained Gradient Ascent

Author:
T.E. KRAK
ICA: 3788644

Supervisors:
DR. A.J. FEELDERS
PROF. DR. A.P.J.M. SIEBES

Master's Thesis
submitted in fulfillment of the requirements
for the degree of
Master of Science
in the programme
Technical Artificial Intelligence
at the
Faculty of Science
Department of Information and Computing Sciences
Utrecht University

Utrecht, August 2013

ABSTRACT

The Exceptional Model Mining (EMM) framework is a problem encountered in exploratory data mining, in which the task is to find subgroups in a dataset, for which a model induced within that group deviates substantially from the same kind of model induced on the entire dataset.

In this work, we consider the EMM problem with linear regression models, where we seek to find subgroups within which the regression model's coefficients deviate substantially from those of the global model. In particular, we consider the Beam Search (BS) strategy that is often used for solving this problem heuristically, and seek to develop a search strategy that can improve on the performance of that algorithm within this context.

Specifically, then, we propose a search strategy that uses individual records' influence on the parameter-estimation of the model within, and hence on the quality of, a subgroup being optimized during the search. The key idea is to generalize, during this search, the notion of a subgroup to that of a *soft* subgroup. We use this conceptualization to rewrite the quality measure as a differentiable function of the individual records' degree of inclusion in this soft subgroup.

Optimization is then performed using what may be interpreted as a form of constrained gradient ascent. Specifically, we introduce a generalization of classification trees, which may be grown on any differentiable objective function. We show that the resulting model may be interpreted, colloquially, as trying to find the *weakest* possible constraint on the objective function, while still conforming to the tree structure. Optimization is then performed by assigning all records to the tree's leafs, and the records' degree of inclusion in the subgroup is optimized *en bloc*, for all records assigned to the same leaf, using gradient ascent. The tree structure then enables us, once converged, to readily express the resulting optimum in a rule-based format that is meaningful in the EMM context. We refer to this tree-constrained gradient ascent strategy as GRADCAT, for Gradient Ascent with Differentiable ClAssification Trees.

We furthermore propose a post-processing procedure that is used to reduce overfitting on the training data, and to remove superfluous conditions in the resulting subgroup description. We refer in this work to the composite algorithm, which constitutes the novel EMM-specific search strategy proper, as GRADCAT-EMM.

By showing that EMM with linear regression models is a strict generalization of the Subgroup Discovery paradigm, i.e., that the latter is

strictly a special case of the former, we find that the novel algorithm can also be immediately applied in that context.

Experiments are performed in which GRADCAT-EMM's performance is compared with that of Beam Search, as well as with that of a composite algorithm that uses the proposed post-processing together with the BS algorithm's method of finding subgroup descriptions. We perform these experiments on both synthetic and real world datasets. On the synthetic data, we find that GRADCAT-EMM consistently statistically significantly outperforms both of the BS-based strategies. On the real world datasets, we initially find that BS performs significantly better than both other algorithms. However, comparing characteristics of the results obtained, we argue that this does not properly reflect our preferences of these characteristics. Arguing that this is due to an oversight in the evaluation measure used, we re-run the experiments on the real world data, using an adapted measure that is meant to compensate for this observed effect. From these experiments, we find that we no longer observe a significant difference in the characteristics of the results obtained. We also find that the BS algorithm with post-processing now performs marginally better than the other strategies, but we largely fail to detect any significant difference between the algorithms.

We end up arguing that the observed relative difference in the algorithms' performance, between the synthetic and real world datasets, can largely be attributed to a difference in the characteristics of these different datasets. In attempting to characterize the difference between these datasets, we do not find a real preference for any of the algorithms if the dataset represents a relatively small and well-structured search space. However, we conclude by hypothesizing that GRADCAT-EMM may be expected to perform particularly well on datasets that represent a large search space with a very low signal-to-noise ratio.

CONTENTS

"And now for something completely different..."

— John Cleese

1 INTRODUCTION

"We are drowning in information and starving for knowledge"[1] is an oft-quoted phrase succinctly describing a particular issue arising from society's ever increasing amount of data collection and storage. To put the point somewhat less eloquently, given the enormous amounts of data that are collected nowadays through automated means, how are institutions and analysts to make sense out of this?

The field of *data mining* aims to address this question, by developing algorithms and techniques that aim to automatically extract useful bits of information from (large) collections of data. Such algorithms may aim, for example, to automatically find frequently occurring patterns of, e.g., customer behavior, which may subsequently be used as a basis for making business decisions.

A particular task that arises in this context is that of *Subgroup Discovery* [18] (SD), in which the problem is to find descriptions of subgroups within a population, for which a particular variable deviates substantially from that of the global population. For example, a researcher in the medical domain may have a database containing information on persons (e.g., age, sex, dietary information, etc.), along with for each person a binary variable indicating occurrence of a particular disease. This researcher may be interested in whether there are groups within this population for which the probability of occurrence of this disease is substantially higher or lower than the probability within the entire population (viz., the entire database). The field of Subgroup Discovery aims to provide algorithms that may automatically find answers to such a question.

A generalization of Subgroup Discovery is the *Exceptional Model Mining* [24] (EMM) problem, which aims to increase the expressiveness of the questions that may be asked regarding the characteristics of subgroups that one wishes to find. Here, instead of a single variable that may deviate from the norm within a particular subgroup, it is now assumed that there are multiple such variables. It is assumed that a particular, typically statistical, model is induced on these variables, and the task is now to find a subgroup for which the model induced on these variables within that subgroup deviates substantially from the model induced on these variables within the entire population. One example of the kind of model that may be used here is the linear regression model [2, 15]. To continue with a (perhaps somewhat artificial) medi-

1 Rutherford D. Roger [15].

cal example, assume that we have a database containing information on patients as before, but that here all patients are known to suffer from the same disease. Furthermore, say that we have for each patient a variable describing the dosage of a particular used medicine, along with a variable quantifying, say, the severity of symptoms of the disease. One could then formulate the linear regression model

$$\text{SeverityOfSymptoms} = \beta_0 + \beta_1 \times \text{Dosage},$$

and ask the question of whether the variable β_1 differs substantially within a certain group of patients, compared to the value of β_1 over the entire population. Such information would describe a different relation within that group between the dosage used and the severity of the symptoms experienced by patients, which may for example be of value when one has to prescribe this medicine to future patients. As an example of a rather extreme case in which such information may be of particular interest, assume that β_1 is negative within the entire population, but positive within a certain subgroup. This would imply that the symptoms' severity is expected to decrease with dosage when one looks at the entire population, but actually *increases* with dosage within that subgroup!

1.1 RESEARCH GOALS AND MOTIVATION

Tasks within the field of data mining in general, and the fields of SD and EMM in particular, may typically be characterized by several important factors that influence how they may be solved. Firstly, one is generally dealing with datasets containing both a large number of records, and a large number of attributes for each record. Secondly, if in particular the EMM task is formulated as an optimization problem, the corresponding objective function will in general not be convex. These factors combine to yield a search space that is typically both very large, and which contains many local optima. As such, exhaustive search to solve the general problem exactly is typically considered computationally intractable. Hence, heuristics are often used to traverse the search space instead.

In this work, we will focus on addressing the EMM task, and in particular the instantiation of EMM that seeks to find exceptional linear regression models. Previous work by Duivesteijn *et al.* [12] has shown that when applied to real world datasets, searching for subgroups with exceptional linear regression models may indeed find interesting groups within such datasets.

As a heuristic for traversing the search space of the EMM problem, a beam search strategy [34] is often used [24, 11, 12]. However, recent work [22, 23] has shown that using more sophisticated approaches may lead to improved results for the EMM task.

In particular, van Leeuwen [22] notes that an inherent limitation of beam search is that it only searches for high quality *descriptions* of subgroups; that is, it ignores how the records in the subgroup influence the model, and how it is this *model* that ultimately determines the subgroup's quality. Van Leeuwen therefore introduces an algorithm which exploits this observation for a particular instantiation of EMM, but it is not obvious how this algorithm may be extended to other model classes. Further, in 2012, van Leeuwen *et al.* [23] published work which addresses a different issue inherent in beam search; the algorithm's tendency to return a set of subgroups which, although all being unique, differ only marginally. That is, the returned subgroups' descriptions tend to have clauses appended to them which may be considered 'noise'. Obviously, such non-informational clauses may interfere with an end-user's ability to interpret these results.

In this work, then, we propose a new heuristic strategy for solving the EMM problem, which explicitly uses information about the individual records' influence on the model, and hence on the subgroup's quality, to find high quality subgroups. Further, we demonstrate that this algorithm is at least immediately applicable to the general problem of Subgroup Discovery. A post-processing procedure is incorporated in the algorithm, which aims, among other things, to reduce unnecessary complexity, i.e., noise, in the resulting subgroups' descriptions.

Briefly, the algorithm may be explained as follows. The key idea is to generalize, during the search process, the records' inclusion in the subgroup to a *soft* measure of inclusion. This allows us to formulate the quality function, that is, the exceptionality of the subgroup's model, as a *differentiable* function of the records' inclusion in the subgroup. This enables us to optimize the subgroup's quality through a specific gradient ascent procedure.

In particular, optimization is performed using what may be interpreted as a form of constrained gradient ascent. For this, we introduce an adaptation of classification trees which, informally, grows a 'scaffolding' of subgroup descriptions within which the gradient ascent's optimum must be found. Somewhat more formally, the resulting structure assigns all records to the tree's leafs, and all records assigned to the same leaf must have the same degree of subgroup inclusion. Optimization is then performed under the constraint that inclusion must be optimized *en bloc* for all records assigned to the same leaf. Once converged, we have a representation of a ('soft') subgroup that is readily expressed in a rule-based format, by virtue of the tree-structure within which it was obtained.

As mentioned, we use a post-processing procedure to arrive at the final subgroup. This is a somewhat involved process with several distinct purposes. First, it is used to convert the resulting soft subgroup into a 'hard' description that is meaningful in the EMM context. Furthermore,

it seeks to both reduce overfitting, and allow for a way to move from local optima that the soft subgroup may have ended up in to even higher quality subgroups. Finally, it aims to reduce the complexity of the subgroup, in terms of the number of clauses in the subgroup's description, both as an application of Occam's razor, and to allow for an easier interpretation of the resulting subgroup description.

In summary, then, we will in this work aim to address the question of whether we can improve on the beam search strategy's performance in finding subgroups with exceptional linear regression models. In particular, we will attempt to develop a search strategy that can successfully use information about individual records' influence on the subgroup's model, in order to obtain such improved performance. We will furthermore investigate whether we can successfully incorporate a mechanism to reduce the number of 'noise' clauses in these subgroups' descriptions.

1.2 STRUCTURE OF THE THESIS

In the remainder of this work, Chapter 2 introduces notation, formalizes the setting of the EMM problem, and describes in basic detail the standard classification tree model which we will later adapt when describing our algorithm. Chapter 3 discusses related work, and describes formally the beam search strategy that is often used for solving the EMM problem. In Chapter 4, the proposed algorithm is described in detail, and Chapter 5 makes some qualitative comparisons between the proposed algorithm and some selected related work from the literature. Chapter 6 describes the experiments that we have performed for evaluating our algorithm, and quantitative comparisons with beam search are made. Chapter 7 finally closes with some conclusions.

2

PRELIMINARIES

In this chapter, we lay the necessary groundwork for the remainder of this thesis. We introduce the notation used throughout this work, formalize the setting of the EMM problem, and introduce the concepts that will form the foundations from which we will develop our algorithm in Chapter 4. Specifically, Section 2.1 briefly discusses the basics of the notation that we will use. In Section 2.2, we formalize the EMM problem, and Section 2.3 briefly discusses the use of linear regression models. Section 2.4, finally, briefly formalizes the notion of classification problems, and subsequently introduces the basic Classification Tree model that we will later adapt in the development of our algorithm.

2.1 NOTATION

We start by briefly introducing the notation used throughout this work. We use lowercase characters for most variables, e.g., x. Bold lowercase characters, e.g., $\mathbf{x}$, denote column vectors, and we write the i-th element of $\mathbf{x}$ as x_i. Bold uppercase characters, e.g., $\mathbf{X}$, denote matrices. We write the element at the i-th row and j-th column of $\mathbf{X}$ as x_{ij}. The i-th row of $\mathbf{X}$ is denoted $\mathbf{X}_{i.}$, and we write its j-th column as $\mathbf{X}_{.j}$. The transpose of a vector $\mathbf{x}$ is written $\mathbf{x}^\top$, and similarly for matrices. We also write the dot product $\mathbf{x}^\top\mathbf{x}$ of a vector $\mathbf{x}$ with itself as its squared L_2-norm, which we will denote $\|\mathbf{x}\|^2 = \mathbf{x}^\top\mathbf{x}$. For matrices and vectors with fixed elements, we write a subscript indicating their dimensions. For example, $\mathbf{I}_{n\times n}$ denotes the $n \times n$ identity matrix, and we write $\mathbf{0}_{n\times 1}$ and $\mathbf{1}_{n\times 1}$ for $n \times 1$ column vectors whose elements are all 0 or 1, respectively. If $\mathbf{X}$ denotes an $n \times n$ matrix, we may write $\mathrm{Tr}(\mathbf{X})$ for the trace of $\mathbf{X}$, that is, $\mathrm{Tr}(\mathbf{X}) = \sum_{i=1}^{n} x_{ii}$. Furthermore, we write $\mathrm{diag}(\mathbf{x})$ for the $n \times n$ diagonal matrix constructed from the $n \times 1$ vector $\mathbf{x}$, that is, $\mathrm{diag}(\mathbf{x})_{ii} = x_i$ for all $1 \leq i \leq n$, and $\mathrm{diag}(\mathbf{x})_{ij} = 0$ for all $i \neq j$ and $1 \leq i, j \leq n$.

We use the $\equiv$ operator for definitions, and the $=$ operator for implied, derived, or well-known equalities, as well as for assignments.

Functions and sets are both denoted with either lowercase or uppercase characters, although we generally write, e.g., $f(\cdot)$ or $F(\cdot)$ when discussing functions. Otherwise, we rely on context to make clear the intended meaning. More abstract sets, or classes, are denoted with a calligraphic font, e.g., $\mathcal{A}$. Sets are defined in the standard fashion using curly brackets, e.g., a set S with elements $s_1, \ldots, s_n$ would be defined as $S \equiv \{s_1, \ldots, s_n\}$. Contrariwise, we write tuples using angle brackets,

e.g., a tuple T with elements a and b would be defined as $T \equiv \langle a, b \rangle$. Whenever something like a set or tuple is itself indexed with a subscript, we will refer to its elements as having that index as a superscript, e.g., $S_i \equiv \{s_1^i, \ldots, s_n^i\}$. The $\emptyset$ character is used both to define the empty set, and as the 'null', i.e., undefined or nonexistent, element. For example, $S = \emptyset$ would specify that S is the empty set, whereas $f(x) = \emptyset$ would denote that $f(\cdot)$ is undefined at x.

2.2 EXCEPTIONAL MODEL MINING

In the Exceptional Model Mining (EMM) framework [24], a dataset $\mathcal{D}$ is typically represented as a multiset of *n records* $r \in \mathcal{D}$ of the form

$$r \equiv \langle a_1, \ldots, a_k, x_1, \ldots, x_p \rangle.$$

The elements $a_1, \ldots, a_k$ are called the *attributes* of r, and $x_1, \ldots, x_p$ the *targets* of r. The attributes are assumed to be taken from some unrestricted domain $\mathcal{A}$, whereas the targets are taken from a domain $\mathcal{T}$ which depends more strongly on the specific problem that one is working on. We refer to the i-th element in $\mathcal{D}$ as r_i, to the set of r_i's attributes as a^i, and to the set of its targets as x^i.

A *pattern* is defined as a function $P : \mathcal{A} \to \{0, 1\}$, and the set of all patterns is denoted with $\mathcal{P}$ and is called the *pattern language*. A pattern is said to *cover* a record r_i if and only if $P(a^i) = 1$. Given a pattern $P \in \mathcal{P}$, a *subgroup* G_P is a multiset $G_P \subseteq \mathcal{D}$ such that

$$G_P \equiv \left\{ r_i \in \mathcal{D} \quad | \quad P(a^i) = 1 \right\}.$$

In the sequel, we will largely drop the subscript P from our notation in cases where no confusion should arise.

Finally, a *quality measure* is defined over the set of all patterns, such that given a dataset $\mathcal{D}$, a quality measure is a function $\varphi_{\mathcal{D}} : \mathcal{P} \to \mathbb{R}$.

In the EMM framework, a model M_D is fitted on the targets of the records in the entire dataset, and a separate model M_G is fitted on the targets of the records in a given subgroup G. Both models are assumed to be from the same model class $\mathcal{M}$. The quality measure then attempts to quantify the quality of the subgroup by the exceptionality of the model fitted on that subgroup, i.e., by computing some distance function between M_D and M_G.

The goal of EMM may now be described as finding a set of 'interesting patterns', where 'interesting patterns' are those that are given a high value by the employed quality measure. For example, given some threshold value $\mu \in \mathbb{R}$, the goal of EMM may be described as finding the set

$$\{ P \in \mathcal{P} \quad | \quad \varphi_{\mathcal{D}}(P) > \mu \},$$

which is to say, the set of all patterns with a quality greater than μ. We want to note that other formulations of the goal of EMM do exist, such as finding the set of w patterns with highest quality. Regardless of the exact formulation of the problem, the size of the set of all possible patterns strongly depends on the expressiveness of the pattern language employed. For most reasonably expressive pattern languages, then, this means that the problem in its general formulation is computationally intractable.

To circumvent this problem, heuristics are often employed. For example, a beam search strategy may be used [24, 12] to limit the number of patterns that have to be considered.

Throughout this work, we consider a specific instantiation of the EMM framework. The model class $\mathcal{M}$ that we consider is the class of linear regression models. It would thus appear to make sense to require the target domain $\mathcal{T}$ to be the real numbers, and hence we let $\mathcal{T} \equiv \mathbb{R}^p$. Furthermore, we assume the domain of attributes $\mathcal{A}$ to be some combination of real and categorical attributes. Thus, we let $\mathcal{A} \equiv \mathbb{A}_1 \times \ldots \times \mathbb{A}_k$, where $\mathbb{A}_j \in \{\mathbb{R}, \mathbb{N}_j\}$ for all $1 \leq j \leq k$, and $\mathbb{N}_j \subset \mathbb{N}$ is a non-empty finite set of the possible values of the j-th attribute when this attribute is categorical.

Finally, we specify the pattern language $\mathcal{P}$ to be the set of all disjunctions of conjunctions of logical conditions on $\mathcal{A}$. In particular, if the j-th attribute is real, the valid conditions on this attribute are of the form $(a_j \leq \tau)$ and $(a_j > \tau)$, for some choice of threshold value $\tau \in \mathbb{R}$. Alternatively, if the j-th attribute is categorical, the valid conditions on this attribute are of the form $(a_j \in v)$, for some choice $v \subset \mathbb{N}_j$ that is a non-empty strict subset of the possible values of this attribute.

2.3 LINEAR REGRESSION

For the sake of notational convenience, we now rewrite the targets x^i of each record r_i in the dataset $\mathcal{D}$ in matrix notation. Let $\mathbf{X}$ be an $n \times p$ matrix such that

$$\mathbf{X} \equiv \begin{bmatrix} 1 & x_1^1 & \ldots & x_{p-1}^1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^n & \ldots & x_{p-1}^n \end{bmatrix},$$

where x_j^i is the j-th target attribute of the record r_i. We will in the following assume that $\mathbf{X}$ is a full column-rank matrix. Furthermore, let $\mathbf{y}$ be an $n \times 1$ vector such that

$$\mathbf{y} \equiv \left[x_p^1, \ldots, x_p^n\right]^\top.$$

In the sequel, when we refer to a linear regression model [2, 15], we intend it to mean the model

$$\mathbf{y} = \mathbf{X}\beta + \varepsilon.$$

Here, β denotes the unknown $p \times 1$ vector of coefficients that we wish to estimate. Furthermore, ε is an $n \times 1$ vector of normally distributed random errors such that $\mathrm{Var}[\varepsilon] = \sigma^2 \mathbf{I}_{n\times n}$ and $\mathbb{E}[\varepsilon] = \mathbf{0}_{n\times 1}$. The least-squares estimate $\hat{\beta}$ for the coefficients β of this model is given by

$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$$

The estimate for the (regression) target vector $\mathbf{y}$ is denoted $\hat{\mathbf{y}}$ and given by

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta}.$$

Finally, the residual vector $\mathbf{e}$ is given by

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}.$$

These estimates will be used when we consider the model fitted on the entire dataset, i.e., the model that is referred to as M_D in the EMM context. To easily represent the model M_G that is fitted to a given subgroup G, we introduce the $n \times n$ diagonal zero-one matrix $\mathbf{W}$. This matrix is defined such that for all $1 \leq i \leq n$,

$$w_{ii} \equiv \begin{cases} 1 & \text{if } r_i \in G, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Using this matrix, we may obtain the matrix $\mathbf{X}_G = \mathbf{W}\mathbf{X}$ that contains the targets of all records in the subgroup, along with zero-rows at positions corresponding to records not included in G. Assuming that $\mathbf{X}_G$ has full column-rank, we may now compute the coefficients of the linear regression model fitted to the subgroup G as

$$\hat{\beta}_G = \left(\mathbf{X}^\top \mathbf{W}\mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{W}\mathbf{y},$$

where $\hat{\beta}_G$ denotes the least squares estimate of the coefficient vector β when we consider only the records in the subgroup G. We can still use these coefficients to make predictions for the entire set of records in the dataset, which are given by the equality $\hat{\mathbf{y}}_G = \mathbf{X}\hat{\beta}_G$. Likewise, the vector of residuals is now defined as $\mathbf{e}_G = \mathbf{y} - \hat{\mathbf{y}}_G$. Note that both $\hat{\mathbf{y}}_G$ and $\mathbf{e}_G$ are $n \times 1$ vectors; they are obtained from coefficients that are estimated on just the subgroup G, but they contain estimates for the *entire* dataset.

2.4 CLASSIFICATION PROBLEMS

We note that in the EMM framework, the problem of finding a pattern with high quality may essentially be described as a binary classification problem. That is, we are looking to classify records as belonging to a subgroup, such that the quality of this subgroup is maximized.

We make use of this observation in the development of our search heuristic for solving the EMM problem. In particular, our method of finding patterns that describe high quality subgroups is largely based on the CART algorithm [4] for training classification trees [4, 30]. In Section 2.4.1, we briefly formalize the binary classification problem. Section 2.4.2 then describes the basic classification tree model, as well as a basic method for training this model, in order to introduce the notation and concepts that we will require for the development of our algorithm in Section 4.3.1.

2.4.1 *Binary Classification*

To formalize the notion of binary classification problems, assume we have a dataset $\mathcal{D}_C$ consisting of n records $r_i, 1 \leq i \leq n$, of the form

$$r_i \equiv \langle a_1^i, \ldots, a_k^i, t^i \rangle,$$

where we again refer to the various a_j^i as the record's attributes, and $t^i \in \{0, 1\}$ is referred to as the record's target. As before, we write a^i for the entire set of attributes of the i-th record, and we assume a^i is taken from some domain $\mathcal{A} = \mathbb{A}_1 \times \ldots \times \mathbb{A}_k$, $\mathbb{A}_j \in \{\mathbb{R}, \mathbb{N}_j\}$ for all $1 \leq j \leq k$. Again, if $\mathbb{A}_j = \mathbb{N}_j$, attribute a_j is a categorical attribute with possible values $\mathbb{N}_j \subset \mathbb{N}$.

We assume that we have a classification model f that is parameterized by some collection of parameters θ, and which we may express as a function $f : \mathcal{A} \times \theta \to [0, 1]$. The function's output over the attributes a^i of the i-th record is the model's prediction of the target value t^i. We denote this prediction with $\hat{t}^i$, and it is specified to lie in the range $[0, 1]$ so that the model may express a certain amount of 'confidence' in its prediction. Clearly, due to this specification, the outputs may also be interpreted as probabilities, which is to say, as $\hat{t}^i = \widehat{\Pr}(t^i = 1 | a^i, \theta)$.

Furthermore, we have some error function that we wish to minimize, say the function $E : [0, 1]^n \times \{0, 1\}^n \to \mathbb{R}$. This function takes the predictions by the model, along with the true target values in the dataset, and computes some distance function between them that quantifies the (lack of) accuracy of the predictions.

The binary classification problem may now be described as finding the parameters θ^* such that

$$\theta^* = \arg\min_{\theta} \Big\{ E(f(a^1, \theta), \ldots, f(a^n, \theta), t^1, \ldots, t^n) \Big\},$$

which is to say, the parameters θ^* that minimize the error function on the given dataset.

2.4.2 *Classification Trees*

We are now in a position to describe the classification tree model [4, 30], which may be used to solve the binary classification problem. As before, let θ denote the parameters of such a model. In the following, we will largely drop θ from our notation, but it is assumed to capture the entire set of properties of the model we are considering. Let T be a binary tree with m vertices $V_0, \ldots, V_{m-1}$. Let $\rho(V_j)$ denote the parent of the j-th vertex, and let $l(V_j)$ and $r(V_j)$ denote the left and right child of V_j, respectively. Let V_0 denote the root of the tree, i.e., let $\rho(V_0) = \emptyset$. For any leaf V_j in the tree, we write $l(V_j) = r(V_j) = \emptyset$.

Let $\mathcal{V}$ denote the class of vertices in such a tree. Each vertex V_j is assumed to be a tuple $\langle d^j(\cdot), o^j \rangle$, such that $o^j \in [0, 1]$ is the vertex's output, and $d^j(\cdot)$ is a function $d^j : \mathcal{A} \to \{l(V_j), r(V_j)\}$. If a vertex V_j is a leaf, its output o^j may be used to make predictions about records' target values. The function $d^j(\cdot)$ may be referred to as a *split* function, and we say that it *assigns* the record whose set of attributes it is given to either of the j-th vertex's children.

Practically speaking, we restrict $d^j(\cdot)$ to be a function over a single attribute a_c, where $c \in \{1, \ldots, k\}$. Furthermore, we restrict this function depending on the type of the attribute that the function is defined over. In particular, if $d^j(\cdot)$ is defined over attribute a_c, and $\mathbb{A}_c = \mathbb{R}$, we restrict $d^j(\cdot)$ to be a linear separation of $\mathbb{R}$. In other words, $d^j(\cdot)$ would be a function

$$d^j(a_1, \ldots, a_k) = \begin{cases} l(V_j) & \text{if } a_c > \tau, \text{ and} \\ r(V_j) & \text{otherwise,} \end{cases}$$

for some threshold value $\tau \in \mathbb{R}$. Otherwise, a_c is a categorical attribute with domain $\mathbb{N}_c \subset \mathbb{N}$, and $d^j(\cdot)$ is a function

$$d^j(a_1, \ldots, a_k) = \begin{cases} l(V_j) & \text{if } a_c \in v, \text{ and} \\ r(V_j) & \text{otherwise,} \end{cases}$$

for some non-empty strict subset $v \subset \mathbb{N}_c$ of the possible values of the categorical attribute. We refer to the class of split functions that satisfy these criteria as $\mathcal{S}$.

We define an auxiliary function $g : \mathcal{A} \times \mathcal{V} \to [0,1]$ that computes a *subtree's* prediction $\hat{t}^i$ of the target value t^i for a given set of attributes a^i. Let such a subtree be rooted at vertex V_j. The function $g(\cdot)$ is then recursively defined as

$$g(a^i, V_j) \equiv \begin{cases} o^j & \text{if } l(V_j) = r(V_j) = \emptyset, \text{ and} \\ g(a^i, d^j(a^i)) & \text{otherwise.} \end{cases}$$

We may now write the classification function $f(\cdot)$ of the classification tree model as

$$f(a^i, \theta) \equiv g(a^i, V_0).$$

Before we move on to describe how one may train, or *grow*, such classification trees on a given dataset, let us first briefly examine how this model works. When a set of attributes a^i is passed to the function $f(\cdot)$, it is first passed on to the root V_0. If V_0 does not have any children, it immediately returns its output o^0, and we have $\hat{t}^i = o^0$. If V_0 does have children, its function $d^0(\cdot)$ assigns the input to either of the root's children, which without loss of generality we will call V_1. The function $g(\cdot)$ is then called with the attributes a^i and vertex V_1 as its arguments. If V_1 is a leaf, it returns o^1 and we have $\hat{t}^i = o^1$. If V_1 is not a leaf, the function $d^1(\cdot)$ is called to decide which of V_1's children the input should be assigned to, and $g(\cdot)$ is again called to pass on the input to the appropriate child. This process repeats recursively until the input is finally assigned to a leaf, say V_j, and this vertex's output o^j is returned so that we have $\hat{t}^i = o^j$.

To determine which leaf a given input a^i is ultimately assigned to when it is recursively passed down a subtree rooted at V_j, we may define the function $h : \mathcal{A} \times \mathcal{V} \to \mathbb{N}$ in analogue to the definition of $g(\cdot)$. Let $h(\cdot)$ be given by

$$h(a^i, V_j) \equiv \begin{cases} j & \text{if } l(V_j) = r(V_j) = \emptyset, \text{ and} \\ h(a^i, d^j(a^i)) & \text{otherwise.} \end{cases}$$

For a given leaf V_j, we may define the set R_{V_j} containing the indices of all records assigned to it as

$$R_{V_j} \equiv \left\{ i \in \mathbb{N} \quad \middle| \quad r_i \in \mathcal{D}_C, \quad h(a^i, V_0) = j \right\}.$$

This definition immediately allows us to specify the output value o^j of leaf V_j. Let this output be given by

$$o^j \equiv \bar{t}_j \equiv \frac{1}{|R_{V_j}|} \cdot \sum_{i \in R_{V_j}} t^i.$$

Thus, the output of a leaf is simply the mean of the target values of the records assigned to it, and we also write this mean as $\bar{t}_j$.

Before we can move on to describe a training algorithm for this model in full, we need several more definitions. First, we require the notion of an *impurity* function, which is a function $\phi : \mathcal{V} \to \mathbb{R}$ that, as the name suggests, quantifies the 'impurity' of a given vertex V_j. Intuitively, what this measures is the relative frequency of the true values of the target attribute, for those records that are assigned to a given vertex. Several different impurity functions are used throughout the literature. For example, the CART algorithm [4] uses the Gini-index, which is given by

$$\phi_{\text{Gini}}(V_j) \equiv \bar{t}_j(1 - \bar{t}_j).$$

Another measure is the entropy impurity function, which is employed by C4.5 [30], and is given by

$$\phi_{\text{Entropy}}(V_j) \equiv -\bar{t}_j \log_2 \bar{t}_j - (1 - \bar{t}_j) \log_2(1 - \bar{t}_j).$$

We furthermore need the notion of validity of split functions. For a given leaf V_j, denote with S_{V_j} the set of valid split functions it could use, *had this vertex not been a leaf*. In other words, suppose we add two new vertices to the tree, say $V_{j'}$ and $V_{j''}$, and let $l(V_j) = V_{j'}$ and $r(V_j) = V_{j''}$. The set of valid split functions for this vertex is then defined as

$$S_{V_j} \equiv \left\{ d(\cdot) \in \mathcal{S} \quad \middle| \quad \exists_{i \in R_{V_j}} : d(a^i) = l(V_j), \quad \exists_{i \in R_{V_j}} : d(a^i) = r(V_j) \right\}.$$

Thus, we require valid split functions to assign at least one record to each of the newly created children.

Finally, we will formalize the notion of the quality of a split. For a given split function $d^j(\cdot)$ and a vertex V_j, denote with $\pi\left(l(V_j)\right)$ and $\pi\left(r(V_j)\right)$ the proportion of records assigned to V_j's left and right child, respectively. Thus, $\pi(\cdot)$ is a function $\pi : \mathcal{V} \to [0, 1]$ and is given by

$$\pi\left(V_{j'}\right) \equiv \frac{|R_{V_{j'}}|}{|R_{\rho(V_{j'})}|}.$$

The quality of a split is now expressed as the reduction of impurity that would be achieved if a leaf V_j would add two children to the tree and used some split function $d^j(\cdot) \in S_{V_j}$, compared to not using any split and remaining a leaf. This split quality is a function $\Delta_\phi : \mathcal{V} \times \mathcal{S} \to \mathbb{R}$ and is defined as

$$\Delta_\phi(V_j, d^j) \equiv \phi(V_j) - \left(\pi\left(l(V_j)\right)\phi\left(l(V_j)\right) + \pi\left(r(V_j)\right)\phi\left(r(V_j)\right)\right).$$

Thus, the quality of a split is essentially defined as the given vertex's impurity, minus the weighted average of the impurity of its children.

Algorithm 1 Training Algorithm for Classification Trees

Input: A dataset $\mathcal{D}_C$
Output: A classification tree T

$T \leftarrow V_0$
$o^0 \leftarrow \bar{t}_0$
$m \leftarrow 1$
OpenList $\leftarrow \{V_0\}$
while |OpenList| > 0 **do**
  $V_j \leftarrow V_* \in$ OpenList
  OpenList $\leftarrow$ OpenList $\setminus V_j$
  if $\left(\phi(V_j) > 0\right) \wedge \left(|S_{V_j}| > 0\right)$ **then**
    $d^j \leftarrow \arg\max_{d \in S_{V_j}} \Delta_\phi(V_j, d)$
    $T \leftarrow T \cup \{V_m, V_{m+1}\}$
    $l(V_j) \leftarrow V_m, \quad r(V_j) \leftarrow V_{m+1}$
    $o^m \leftarrow \bar{t}_m, \quad o^{m+1} \leftarrow \bar{t}_{m+1}$
    OpenList $\leftarrow$ OpenList $\cup \{V_m, V_{m+1}\}$
    $m \leftarrow m + 2$
  end if
end while

We now have all the elements required for the specification of the training algorithm, which is given by Algorithm 1.

Briefly, what this algorithm does is the following. We start by initializing the tree with just a root vertex, the output of which is set to be the mean of the target values of the records assigned to it, which here corresponds to all records in the dataset. This vertex is added to the OpenList, which is essentially the set of all leafs not yet considered for expansion. As long as there are vertices in this list, we consider them in turn and remove them from the list. When a given leaf V_j is thus considered, we first check if it has any impurity at all and whether it has any valid splits. If either is not the case, this leaf clearly need not, or cannot, expand any further and it will remain a leaf. Otherwise, we find the split function with maximum quality in its set of valid split functions, which is subsequently assigned to be the split function $d^j(\cdot)$ that the vertex under consideration will use. Then, two new vertices are added to the tree, and they are assigned to be V_j's children. Clearly, these children are now the leafs of the subtree rooted at V_j, and they are in turn added to the OpenList to be considered for expansion. This process repeats until no more leafs can be expanded.

It should be noted that this algorithm will have a tendency to overfit to the given data. This can be seen by the fact that leafs will continue to expand as long as they are impure and can still perform a valid split. In extreme cases, this may result in a tree where each leaf is only as-

signed a single record, and the model will make perfect predictions on all records on which it is trained. In practice, one may increase the minimum number of records that need to be assigned to each child for a split to be considered valid. Another technique may be to only allow splitting of a leaf when it itself is assigned at least a minimum number of records. Other methods from the literature include post-processing algorithms which aim to reduce the complexity of the tree and, through this process, improve the generalization performance of the model.

3

RELATED WORK

The EMM framework was first introduced by Leman *et al.* [24] in 2008. In this work, applications to several model classes were discussed, specifically correlation models, (simple) linear regression, and various classification models. Furthermore, the authors suggested possible quality measures for these different models. Here, the use of a beam search [34] strategy as a search heuristic was also suggested. In particular, the authors mention that this algorithm was chosen because of the balancing that it achieves between a greedy method and the maintaining of multiple candidate solutions.

In 2010, Duivesteijn *et al.* [11] showed an application of EMM where Bayesian Networks [26] were used as the target model. Here, a beam search strategy was also used for their experiments.

Also in 2010, van Leeuwen [22] discussed an application of EMM where the distribution of the data was used as the target model. Specifically, he used both Kullback-Leibler divergence [21] and a novel measure, based on Minimum Description Length [32] and KRIMP [33], as quality measures to quantify the exceptionality of the data distributions. Also in this work, the possibility that different algorithms may outperform beam search is mentioned. Van Leeuwen therefore introduces a novel algorithm that is more specifically tailored to the EMM problem. He refers to this algorithm as Exception Maximization and Description Minimization, or EMDM for short. Briefly, the algorithm is essentially a two-step iterative process that optimizes an initial candidate solution. In the first (EM) step, a new subgroup is constructed from the (previous) candidate solution, such that the exceptionality of this new subgroup is optimized. In the second (DM) step, the algorithm searches for a description of this new subgroup, by making use of the RIPPER [5] classification algorithm. Van Leeuwen goes on to show that this algorithm can indeed outperform the beam search algorithm on the specific problem he considers. It is, however, not immediately obvious how this EMDM algorithm, and the EM step in particular, may be generalized to different target models for the EMM problem.

Work by Duivesteijn *et al.* [12] from 2012 discussed an application of EMM to linear regression models. Here, a quality measure was suggested that is more straightforward to apply to multiple linear regression than the quality measure suggested earlier by Leman *et al.* [24]. The experiments presented in this work again used the beam search strat-

egy as a search heuristic, and it was shown that interesting results could indeed be obtained on real world datasets.

Van Leeuwen *et al.* [23] in 2012 published work regarding a novel search heuristic designed to improve performance in both the Subgroup Discovery and EMM settings. This algorithm is largely based on beam search, and is referred to as Diverse Subgroup Set Discovery. Basically, the algorithm is designed to simultaneously maximize the quality of, and diversity between, the candidate solutions in the beam search algorithm's beam.

Because of the apparent pervasiveness of (variations of) the beam search strategy throughout the related literature, we have chosen this algorithm as the baseline against which we will compare our proposed algorithm. For the sake of completeness, Section 3.1 briefly formalizes this algorithm for use in the EMM context.

3.1 BEAM SEARCH

This section describes the beam search algorithm that is often used as a heuristic for solving the EMM problem. Using a less strict interpretation of the goal of EMM defined in Section 2.2, our goal will be to find a set of patterns with 'high' quality. Here, a pattern will be considered of 'high' quality when its quality is greater than that of some subset of the patterns that we have considered during the search process. Clearly, this is a weaker statement than the original formulation of the problem, in which we wanted to find *all* patterns with a quality greater than a given value. This weakening of the problem ties back to the heuristic approach to solving it, and makes the problem tractable.

Recall from Section 2.2 that a pattern is defined as a set of disjunctions of conjunctions of logical conditions on the domain of attributes $\mathcal{A}$. Abstractly, we will model conditions of this form as functions $c : \mathcal{A} \to \{0, 1\}$ whose domain is the values of a single attribute $\mathbb{A}_j$. Let $c(a^i) = 1$ if the condition is logically satisfied on a given set of attributes a^i, and $c(a^i) = 0$, otherwise. Let $\mathcal{C}$ denote the set of all such conditions. A set of conjunctions of conditions λ is referred to as a *rule*, and defined such that $\lambda \subset \mathcal{C}$. The set of all rules $\mathcal{R}$ is defined as the powerset of $\mathcal{C}$, i.e., $\mathcal{R} \equiv \wp(\mathcal{C})$.

Typically, when used in the EMM setting, the beam search algorithm only searches for single rules, and does not consider disjunctions of rules. In this context, then, a pattern simply evaluates a given rule λ. That is, with a slight abuse of notation, the function of the pattern $P(\cdot)$ may be written as

$$P(a^i, \lambda) = \min_{c(\cdot) \in \lambda} \left\{ c(a^i) \right\}.$$

Briefly, the beam search algorithm works by performing a level-wise search through the space $\mathcal{R}$. In other words, it only *adds* conditions to rules already considered. Because rules are conjunctions of conditions, adding a new condition essentially acts as a refinement operator. That is, the number of records covered by the pattern defined by the rule that is being extended can only decrease. Furthermore, rather than storing and refining all rules considered thus far, the algorithm only keeps track of a limited set of rules. This set of rules is referred to as the *beam*, and the maximum size of the beam is called the *beam width*, which is a parameter of the algorithm. Finally, the algorithm only searches for rules up to a given complexity, which is referred to as the search *depth* of the algorithm.

More formally, let ω denote the beam width, and let $\mathcal{B}$ be the algorithm's beam such that $\mathcal{B} \subset \mathcal{R}$ and $|\mathcal{B}| \leq \omega$. Let $V_\lambda \subset \mathcal{C}$ denote the set of valid conditions that may be added to λ. Explicit definition of V_λ is rather involved; we will return to this later in this section. Finally, let $\mathcal{R}_\lambda^+$ be the set of all valid refinements of λ, so that we have

$$\mathcal{R}_\lambda^+ \equiv \Big\{ \lambda \cup \{c(\cdot)\} \quad \Big| \quad c(\cdot) \in V_\lambda \Big\}.$$

The beam search algorithm is now given by Algorithm 2.

Algorithm 2 Beam Search

Input: A dataset $\mathcal{D}$, beam width ω, and maximum depth MaxDepth.
Output: A set of patterns $\mathcal{B}$

$\mathcal{B} \leftarrow \emptyset$
for Depth $= 1 \rightarrow$ MaxDepth **do**
    Candidates $\leftarrow \mathcal{B} \cup \bigcup_{\lambda \in \mathcal{B}} \mathcal{R}_\lambda^+$
    $\lambda_{(1)}, \ldots, \lambda_{(|\text{Candidates}|)} \leftarrow \text{Sort}(\text{Candidates}, \varphi_\mathcal{D}(\cdot))$
    $\mathcal{B} \leftarrow \Big\{ \lambda_{(1)}, \ldots, \lambda_{(\omega)} \Big\}$
end for
return $\mathcal{B}$

The algorithm starts with an empty beam, and then searches up to a specified depth MaxDepth. At each level, it generates a set Candidates, which contains all rules that may be included in the beam at this depth. At Depth $= 1$, this corresponds to the set of all valid single conditions. At deeper levels, this set contains all refinements of rules included in the beam at the previous depth, along with those rules themselves. All rules in the set Candidates are then sorted in order of decreasing quality. Somewhat more formally, but with a slight abuse of notation, write $\varphi_\mathcal{D}(\lambda)$ for the quality of the pattern defined by the rule λ. The set of candidates is ordered as $\lambda_{(1)}, \ldots, \lambda_{(|\text{Candidates}|)}$ such that $\varphi_\mathcal{D}(\lambda_{(1)}) \geq \ldots \geq \varphi_\mathcal{D}(\lambda_{(|\text{Candidates}|)})$. The best ω rules are then selected from this ordered

list of candidates and together specify the new beam. If there are less than ω candidates at this level, the entire set of candidates is selected for inclusion in the beam. After all candidates at the maximum depth have been considered, the resulting beam is returned, containing the best ω rules encountered during the search.

We now return to the specification of the set V_λ containing all conditions that may be validly added to a given rule λ. There are essentially two characteristics that must be satisfied for a condition to be considered valid for adding to a rule. The first is that the *support* of the resulting subgroup must not be lower than a given parameter specifying the minimum support. That is, let MinSup be a parameter given to the algorithm that specifies the minimum size of a valid subgroup. Let $\lambda' = \lambda \cup \{c(\cdot)\}$ be a refinement of λ, and let $G_{\lambda'}$ denote the subgroup defined by the refined rule λ'. Then, $c(\cdot) \in V_\lambda$ only (but not necessarily) if $|G_{\lambda'}| \geq \text{MinSup}$.

The second characteristic of valid conditions pertains to the parameterization of such conditions. That is, it specifies on which values of an attribute the condition may be defined. Recall from Section 2.2 that if a condition $c(\cdot)$ is defined over a categorical attribute $\mathbb{A}_j$, it takes the logical form $(a_j \in v)$, where $v \subset \mathbb{N}_j$ is a non-empty finite strict subset of the possible values $\mathbb{N}_j$ of the j-th attribute. The additional restriction that is now imposed for $c(\cdot)$ to be in V_λ is that there does not exist a condition $c'(\cdot) \in \lambda$, that is also defined over the j-th attribute, and which takes the form $(a_j \in v')$ such that $v' \subset \mathbb{N}_j$ and $v \cap v' = \emptyset$. This restriction also follows directly from the first characteristic of valid conditions, i.e., that the support of the resulting subgroup must be of a minimum size (assuming MinSup > 0). This can be seen by the fact that $(a_j \in v) \wedge (a_j \in v')$ can never logically hold if $v \cap v' = \emptyset$, and which would thus result in an empty subgroup. However, we feel that making this restriction explicit may help in the interpretation of valid conditions.

The alternative is that a condition $c(\cdot)$ is defined over the j-th attribute, and that $\mathbb{A}_j = \mathbb{R}$. In this case, the logical form of the condition is either $(a_j > \tau)$ or $(a_j \leq \tau)$, for some choice of $\tau \in \mathbb{R}$. The restriction here pertains to how τ may be chosen. We here follow the implementation of the Cortana [19] framework, which works by distributing all values of the j-th attribute of records in G_λ into m equal-frequency bins. Here, m is again a parameter of the algorithm that may be specified by the user. More formally, the values of the j-th attribute of all records $r_{(i)} \in G_\lambda$ are ordered such that $a_j^{(1)} \leq \ldots \leq a_j^{(|G_\lambda|)}$. These values are then distributed in order into m equal-frequency bins, and the valid choices for τ occur at the boundaries between two such bins. That is, there are $m - 1$ *split points*, i.e., choices for τ, occurring at the boundaries of the first and second, second and third, and so on to the $(m-1)$-th and m-th bin.

Formally, the set S_λ^j of valid split points on the j-th attribute that may be used in conditions to add to λ, is defined by

$$S_\lambda^j \equiv \left\{ a_j^{(k)} \quad \middle| \quad k = \left\lfloor \frac{|G_\lambda| \cdot l}{m} \right\rfloor, 1 \leq l < m \right\}.$$

A condition $c(\cdot)$ defined on the j-th attribute may then satisfy $c(\cdot) \in V_\lambda$ only if it is of the form $(a_j > \tau)$ or $(a_j \leq \tau)$, and $\tau \in S_\lambda^j$. Clearly, the restriction on the minimum support of the resulting subgroup also still needs to be satisfied for $c(\cdot) \in V_\lambda$ to hold.

4

THE GRADCAT-EMM ALGORITHM

In this chapter, we describe in detail the novel search heuristic that we propose for solving the EMM problem with linear regression models. We begin by discussing the quality measure used in Section 4.1. Section 4.2 proceeds with discussing the idea of making this measure differentiable with respect to individual records' inclusion in the subgroup. In Section 4.3, we bring together the notion of classification trees and this differentiable objective function. Section 4.3.1 proceeds with the exposition of the GRADCAT strategy, in which we introduce an adaption to classification trees that allows us to perform optimization on an arbitrary differentiable function.

Some auxiliary procedures are then discussed, specifically how to perform random restarts in Section 4.4, and the post-processing of the subgroup description, in Section 4.5. Finally, Section 4.6 brings all these ideas together, and formulates the GRADCAT-EMM algorithm proper.

4.1 QUALITY MEASURE AND COOK'S DISTANCE

As was noted in Section 2.2, in the EMM framework we require a quality measure $\varphi_{\mathcal{D}}(\cdot)$ that quantifies the exceptionality of a given model M_G. We also noted that this measure may essentially be regarded as a distance measure between the model fitted to the subgroup covered by a given pattern and the model fitted to the entire dataset.

Given that we are considering linear regression models, it makes sense to let our quality measure be some function of the models' coefficient vectors $\hat{\beta}_G$ and $\hat{\beta}$. An intuitive first proposal may then be to use the (squared) Euclidean distance between these vectors, i.e., to let

$$\varphi_{\mathcal{D}}(P) = \left(\hat{\beta}_{G_P} - \hat{\beta}\right)^\top \left(\hat{\beta}_{G_P} - \hat{\beta}\right).$$

As was noted by Duivesteijn *et al.* [12], using this measure has the disadvantage that it ignores the variances of, and covariances between, the various coefficients in these vectors. Clearly, one could compensate for this by using the measure

$$\varphi_{\mathcal{D}}(P) = \left(\hat{\beta}_{G_P} - \hat{\beta}\right)^\top \left(\widehat{\mathrm{Cov}}[\hat{\beta}]\right)^{-1} \left(\hat{\beta}_{G_P} - \hat{\beta}\right), \tag{4.1}$$

where $\widehat{\mathrm{Cov}}[\hat{\beta}]$ is the sample estimate of the covariance matrix of $\hat{\beta}$. However, Duivesteijn *et al.* instead propose to use a measure called *Cook's*

Distance [6, 7], which is equivalent to this measure up to a constant scaling factor of $\frac{1}{p}$. Using Cook's Distance has the advantage that it allows for an arguably more intuitive interpretation of the resulting quality values. It was originally proposed by Cook [6] as a measure of the influence of individual records on the estimation of $\hat{\beta}$.

The covariance matrix $\mathrm{Cov}[\hat{\beta}]$ may be estimated from the data by

$$\widehat{\mathrm{Cov}}[\hat{\beta}] = s^2 \left(\mathbf{X}^\top \mathbf{X}\right)^{-1}.$$

Here, s^2 is the unbiased estimator of σ^2 and is given by

$$s^2 = \frac{\mathbf{e}^\top \mathbf{e}}{n-p}.$$

Cook's Distance[1] D_G is then defined as

$$\begin{aligned} D_G &\equiv \frac{\left(\hat{\beta}_G - \hat{\beta}\right)^\top \left(\widehat{\mathrm{Cov}}[\hat{\beta}]\right)^{-1} \left(\hat{\beta}_G - \hat{\beta}\right)}{p} \\ &= \frac{\left(\hat{\beta}_G - \hat{\beta}\right)^\top \mathbf{X}^\top \mathbf{X} \left(\hat{\beta}_G - \hat{\beta}\right)}{p s^2}. \end{aligned}$$

The interpretation of D_G is motivated as follows. Let $F(p, n-p, 1-\alpha)$ denote the $1-\alpha$ probability point of the central F-distribution with p and $n-p$ degrees of freedom. Cook [6] notes that according to normal theory, the $(1-\alpha) \times 100\%$ confidence ellipsoid of $\hat{\beta}$ is given by all vectors β^* satisfying

$$\frac{\left(\beta^* - \hat{\beta}\right)^\top \mathbf{X}^\top \mathbf{X} \left(\beta^* - \hat{\beta}\right)}{p s^2} \leq F(p, n-p, 1-\alpha).$$

Thus, for example, if for a given subgroup G we have $D_G \approx F(p, n-p, 0.5)$, we may interpret this by saying that a model fitted to G moves the estimate of the coefficient vector to the edge of the 50% confidence ellipsoid of $\hat{\beta}$.

Despite the advantages offered by using Cook's Distance as a quality measure for the EMM problem, we note that it still has two properties

1 Some remarks on the notation used. Cook originally wrote D_I and $\hat{\beta}_{(I)}$ for, respectively, the distance measure and estimate of the coefficient vector on a subset of the data. Here, I corresponds to a list of the records *left out* of this subset, and (I) indicates that the coefficients are estimated on all records *not* left out. Thus, using our notation for subgroups of the data G, using this convention would denote the distance measure and coefficients as, respectively, $D_{\mathcal{D}\setminus G}$ and $\hat{\beta}_{(\mathcal{D}\setminus G)}$. Although Cook's notation is arguably more clear in the context of evaluating the influence of individual records on the estimation of the model parameters, we consider it rather confusing in the context of EMM. As such, we are instead using a notation that we feel should be more intuitive for the problem at hand.

that we consider undesirable in such a quality measure. First, although it takes into account the variances of, and covariances between, the coefficients in the vector $\hat{\beta}$, it does depend on the number of regression coefficients because of the term $\frac{1}{p}$. In particular, because we want to eventually perform gradient ascent on this quality measure, we find it desirable that the derivative of the quality measure be scale-invariant to this quantity. Therefore, we propose to drop this scaling factor and use a measure that we will refer to as the Squared Standardized Prediction Difference (SSPD), which was already given by Equation 4.1. We redefine this measure here as

$$\begin{aligned}\text{SSPD}(G) &\equiv \frac{\left(\hat{\beta}_G - \hat{\beta}\right)^\top \mathbf{X}^\top \mathbf{X} \left(\hat{\beta}_G - \hat{\beta}\right)}{s^2} \\ &= \frac{\left(\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\right)^\top \left(\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\right)}{s^2} \\ &= \frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{s^2}.\end{aligned}$$

This measure sacrifices the somewhat intuitive interpretation that Cook's Distance offered, but in exchange we gain a different interpretation in terms of the squared standardized prediction difference, which we argue is still intuitive. We note that because p is constant with respect to the dataset, using this measure still yields the same optima as those given by Cook's Distance.

The second property that we consider desirable in a quality measure for the EMM problem, but which is not satisfied by Cook's Distance, is the taking into account of the *support* of a subgroup. That is, we prefer larger subgroups over smaller ones, all other things being equal. We incorporate this support into the quality measure in an admittedly somewhat heuristic fashion, by defining the Support Weighted Squared Standardized Prediction Difference (SWSSPD) as

$$\begin{aligned}\text{SWSSPD}(G) &\equiv \frac{|G|}{n} \cdot \frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{s^2} \\ &= \frac{\text{Tr}(\mathbf{W})}{n} \cdot \frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{s^2},\end{aligned}$$

where $|G|$ denotes the size of the subgroup G, $\mathbf{W}$ denotes the $n \times n$ diagonal matrix that is used to estimate the vector $\hat{\beta}_G$ as described in Section 2.3, and $\text{Tr}(\mathbf{W})$ denotes the trace of $\mathbf{W}$, that is, $\text{Tr}(\mathbf{W}) = \sum_{i=1}^{n} w_{ii} = |G|$.

The final form of the quality measure that we will use is half the SWSSPD, where the scale factor of $\frac{1}{2}$ is introduced for later convenience. We formalize the quality measure as

$$\varphi_{\mathcal{D}}(P) \equiv \frac{\text{SWSSPD}(G_P)}{2}. \tag{4.2}$$

4.2 SUBGROUP OPTIMIZATION

As was noted before, the goal of EMM is essentially to find those patterns that yield a high value on the employed quality measure. Another way to say this is that we are looking for descriptions of high quality *subgroups*. In this section, we look at how to optimize the quality of a subgroup, by expressing the quality measure as a function of the inclusion of the various records in a subgroup. We will then consider how to find patterns that *describe* these subgroups in Section 4.3.

The approach that we use here is to make the quality measure differentiable with respect to the inclusion of records in the subgroup; this will allow us to perform gradient ascent to optimize the quality measure. Recall from Section 2.3 that for the estimation of the coefficients $\hat{\beta}_G$, we represent the inclusion of each record using the $n \times n$ diagonal zero-one matrix $\mathbf{W}$. We now generalize the notion of inclusion of a given record r_i in a subgroup G to that of a *soft* inclusion in this subgroup. To represent this, we let $w_{ii} \in [0,1]$, for all $1 \leq i \leq n$. In the sequel, we will for notational convenience denote the i-th diagonal element of $\mathbf{W}$ as w_i, and refer to these elements as *inclusion weights*. Intuitively, these inclusion weights work as follows. As w_i moves toward zero, the record r_i is increasingly "removed", or rather, less strongly included, in the subgroup G. Likewise, as w_i approaches one, r_i's inclusion in the subgroup becomes increasingly strong.

It can easily be seen that as w_i goes to zero, the influence of the i-th record on the estimation of $\hat{\beta}_G$ decreases, and completely disappears in the limit where $w_i = 0$. The converse obviously holds for the case where w_i goes to one.

Note also that the formula for $\hat{\beta}_G$ now coincides with the solution for the regression coefficients β in the weighted least squares (WLS) problem [7]. The WLS problem formulates a linear regression model in the same way as that outlined in Section 2.3, except it is assumed that $\mathrm{Var}[\varepsilon] = \sigma^2 \mathbf{W}^{-1}$, where $w_i > 0$ for all $1 \leq i \leq n$. Although strictly speaking the inclusion weight matrix $\mathbf{W}$ does not conform to this definition, it does offer an interesting interpretation. In particular, if we consider the inclusion weights in this setting, the limit where $w_i = 0$ may be interpreted as saying that the error ε_i of the i-th record has infinite variance, which means that the i-th record does not carry any information that is useful for the estimation of the regression coefficients. Hence, it can safely be ignored in the estimation of these coefficients.

We are now in a position to explicitly optimize our quality measure through gradient ascent. We first rewrite the quality measure given by

Equation 4.2 as an objective function $O : [0,1]^n \to \mathbb{R}$ that is a function of the inclusion weights, so that we have

$$O(w_1, \ldots, w_n) \equiv \frac{\mathrm{Tr}(\mathbf{W})}{n} \cdot \frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2s^2}. \tag{4.3}$$

It should be noted that $O(\cdot)$ is not a very well-behaved function. Besides in general containing multiple local optima, it is not defined everywhere. This can be seen by the term $(\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}$ that occurs in the definition of $\hat{\beta}_G$. Clearly, for some choices of $\mathbf{W}$, this inverse does not exist, and hence $O(\cdot)$ will not be defined at that position. Whenever we encounter such a situation during the optimization process, we simply restart the optimization from a different starting position.

To optimize the quality of a subgroup, we take partial derivatives of $O(\cdot)$ with respect to the inclusion weights. The derivation of these partial derivatives is, however, somewhat involved, so we refer the interested reader to Appendix B for the full proof. The partial derivatives are given by the equality

$$\frac{\partial O(\cdot)}{\partial w_i} = \frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2ns^2} - \frac{\mathrm{Tr}(\mathbf{W})}{ns^2} \cdot \left(\mathrm{diag}(\mathbf{e}_G)\mathbf{X} \left(\mathbf{X}^\top \mathbf{W}\mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{e}_G \right)_i .$$

Inspection of this equality reveals that, if one is careful with the order and method of multiplication of the various matrices involved, the entire set of derivatives can be computed in a time complexity order that is linear in n. The remainder of this section explains how this may be done.

In particular, care should be taken that multiplication with the diagonal matrices $\mathbf{W}$ and $\mathrm{diag}(\mathbf{e}_G)$ is not performed naively. Then, the derivatives can be computed in the following way. We begin by showing how to compute $\hat{\beta}_G$ in time $\mathcal{O}(p^2 n)$.

Start with the computation of $\mathbf{X}_G = \mathbf{W}\mathbf{X}$, which if not done naively can be performed in $\mathcal{O}(pn)$. Form the $p \times p$ matrix $\mathbf{M} = (\mathbf{X}^\top \mathbf{W}\mathbf{X})$ by computing $\mathbf{X}^\top \mathbf{X}_G$, which takes $\mathcal{O}(p^2 n)$. Likewise, form the $p \times 1$ vector $\mathbf{z} = \mathbf{X}^\top \mathbf{W}\mathbf{y}$ from $\mathbf{X}_G^\top \mathbf{y}$, taking $\mathcal{O}(pn)$. Find $\mathbf{M}^{-1}$, the complexity of which depends on the specific algorithm used, but a straightforward implementation takes $\mathcal{O}(p^3)$. Finally compute $\hat{\beta}_G = \mathbf{M}^{-1}\mathbf{z}$ in $\mathcal{O}(p^2)$. The total time complexity is thus $\mathcal{O}(pn + p^2 n + pn + p^3 + p^2) = \mathcal{O}(p^3 + p^2 n)$. By the assumption from Section 2.3 that $\mathbf{X}$ has full column rank, we have $n \geq p$. Hence, $p^2 n \geq p^3$, and $\mathcal{O}(p^3 + p^2 n) = \mathcal{O}(p^2 n)$.

Having obtained $\hat{\beta}_G$ for the current $\mathbf{W}$, we compute the estimates of the regression target values and the vector of residuals. We compute $\hat{\mathbf{y}}_G = \mathbf{X}\hat{\beta}_G$, taking $\mathcal{O}(pn)$. We then find $\mathbf{e}_G = \mathbf{y} - \hat{\mathbf{y}}_G$ in $\mathcal{O}(n)$.

As an aside, we assume that $\hat{\mathbf{y}}$ and s^2 are already known at this point, but from the above it is easily seen that these can also be found in $\mathcal{O}(p^2 n + pn + n + n) = \mathcal{O}(p^2 n)$ by setting $\mathbf{W} = \mathbf{I}_{n \times n}$. Here, we have assumed that the estimate of the error variance s^2 is computed in $\mathcal{O}(n)$ from $s^2 = \frac{\mathbf{e}^\top \mathbf{e}}{n-p}$, having obtained $\mathbf{e}$ as outlined above.

We now have all the terms required to compute the partial derivatives in time $\mathcal{O}(pn)$. The first summand can clearly be computed in $\mathcal{O}(n)$ using the pre-computed vectors $\hat{\mathbf{y}}_G$ and $\hat{\mathbf{y}}$. Further, this term does not depend on i, so we do not need to re-compute it for every partial derivative. Similarly, the scalar of the second summand, $\frac{\mathrm{Tr}(\mathbf{W})}{ns^2}$, only needs to be computed once, also taking $\mathcal{O}(n)$.

We now show that the entire vector in the derivative's second summand can be computed in $\mathcal{O}(pn)$. Start by computing the $p \times 1$ vector $\mathbf{a} = \mathbf{X}^\top \mathbf{e}_G$ in time $\mathcal{O}(pn)$. Having previously obtained $\mathbf{M}^{-1} = (\mathbf{X}^\top \mathbf{W} \mathbf{X})^{-1}$, compute the $p \times 1$ vector $\mathbf{b} = (\mathbf{X}^\top \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{e}_G = \mathbf{M}^{-1} \mathbf{a}$ in time $\mathcal{O}(p^2)$. Form the $n \times 1$ vector $\mathbf{c} = \mathbf{X}\mathbf{b}$ in $\mathcal{O}(pn)$. Taking care not to multiply by the diagonal matrix naively, we finally find the entire vector $\mathbf{d}$ in the derivative's second summand by computing $\mathbf{d} = \mathrm{diag}(\mathbf{e}_G)\mathbf{c}$ in time $\mathcal{O}(n)$. Thus, the total time complexity to find $\mathbf{d}$ is $\mathcal{O}(pn + p^2 + pn + n) = \mathcal{O}(pn)$.

The i-th partial derivative can now be computed in $\mathcal{O}(1)$ from the pre-computed first summand, the pre-computed scalar in the second summand, and the i-th element in $\mathbf{d}$.

Thus, the total time complexity is $\mathcal{O}(p^2 n)$ to find $\hat{\beta}_G$, $\mathcal{O}(pn)$ for the estimates $\hat{\mathbf{y}}_G$ and $\mathbf{e}_G$, $\mathcal{O}(pn)$ to find the vector $\mathbf{d}$, and n times $\mathcal{O}(1)$ to compute the individual derivatives, for a total of $\mathcal{O}(p^2 n + pn + pn + n) = \mathcal{O}(p^2 n)$, which is linear in n.

4.3 PATTERN OPTIMIZATION

As was mentioned before, the problem of finding a pattern with high quality can essentially be described as a binary classification problem; find the pattern that classifies records as belonging to a subgroup, such that the quality of this subgroup is maximized. The membership of records to such a high quality subgroup are then the targets of the classification problem that we wish to solve. Intuitively, then, we might want to consider standard classification algorithms from the machine learning literature to solve this problem.

There appear to be at least two distinct ways of going about solving this problem. First, we may assume that we are given a high quality subgroup G, and try to solve the classification problem explicitly. In other words, we would start by finding a high quality subgroup G in terms of the inclusion of the individual records, and then try to fit a model on the attributes of the records in our dataset, using this model to predict membership of the records to the subgroup G. This has the obvious drawback that it would require that membership to this subgroup is expressible in our pattern language, i.e., by whatever classification model we are using. More to the point, it would require that membership is expressible in our pattern language *using the records' attributes*. In other words, finding

a high quality subgroup *only* in terms of the records' inclusion in no way guarantees that the resulting subgroup will have meaningful structure in terms of the values of the included records' attributes.

Some initial experiments confirmed that this is indeed problematic in practice. Here, we first searched for a subgroup G by performing gradient ascent on the records' inclusion weights, in order to maximize the objective function $O(\cdot)$ given by Equation 4.3. We then attempted to fit a classification model to predict membership to the resulting subgroup. Specifically, we here used the CART [4] algorithm as our classifier of choice. These experiments showed that the resulting subgroup's records' attributes indeed lacked a structure that could be properly expressed by the classification model.

To circumvent these issues, we have instead chosen to apply a more integrated approach. Because we have an objective function that may be explicitly optimized with respect to the inclusion of records in a subgroup, we may use this objective function to train a classification model directly. In other words, we are reverting to the interpretation of the problem as that of finding a pattern with high quality directly, but specify an intermediate step that provides an interface that we may use to train classification models. We do this by replacing the error function that would normally be optimized during the training of such models with the objective function $O(\cdot)$ given by Equation 4.3.

A hurdle in this approach is the nature of many of the classification models from the literature that are typically trained by some form of gradient ascent. For example, models like Logistic Regression [2] or Neural Networks [2] would fit this category. The problem is that such models typically do not satisfy the form of pattern language that was stipulated in Section 2.2. Clearly, given this specification of pattern language, what we are looking for is a rule-based classification model. However, due to the discrete nature of such classification rules, this kind of model typically is not trained through gradient ascent (although such models typically do use some form of hill-climbing algorithm for training). Models that fit this category would for example be Classification Trees like CART [4] and C4.5 [30], or rule-based algorithms like RIPPER [5].

Because we have not been able to find in the literature an off-the-shelf model that appears to fit our specific needs, we have chosen to adapt an existing model for training with arbitrary differentiable objective functions. In particular, this model will be based on classification trees, and, specifically, on the CART algorithm. We will refer to this new algorithm as GRADCAT, for GRadient Ascent with Differentiable ClAssification Trees.

4.3.1 *GRADCAT*

We now describe how to adapt the classification tree algorithm discussed in Section 2.4.2 for use with arbitrary differentiable objective functions. The notation that we use here will largely follow the definitions used in Section 2.4.2, and the relevant changes will be discussed as required.

We start by moving away from the setting of binary classification problems, and toward a more general setting. We now assume that we have access to a dataset $\mathcal{D}_A$ consisting of n records of the form

$$r_i \equiv \langle a_1^i, \ldots, a_k^i \rangle,$$

where the entire set of attributes of r_i is again denoted a^i and is assumed to be taken from the domain $\mathcal{A}$ as before. Note that we no longer have access to any target attributes that we wish to predict.

We consider an arbitrary differentiable objective function $O : [0,1]^n \to \mathbb{R}$, which we will write as $O(o_1, \ldots, o_n)$. We assume that we have access to the n partial derivatives of this function with respect to its inputs, which is to say, we assume that we are given

$$\frac{\partial O(\cdot)}{\partial o_i}, \quad \text{for all } 1 \leq i \leq n.$$

Our model will again be expressed as a function $f : \mathcal{A} \times \theta \to [0,1]$ that is parameterized by some collection of parameters θ. Because we are now considering more general *objective* functions, as opposed to *error* functions, we change the problem to that of a maximization problem. Thus, we now seek to find parameter values θ^* such that

$$\theta^* = \arg\max_{\theta} \Big\{ O(f(a^1, \theta), \ldots, f(a^n, \theta)) \Big\}.$$

Clearly, for general objective functions $O(\cdot)$, we cannot hope to find θ^* such that it is globally optimal. Instead, we introduce the algorithm that is to come as a heuristic method that may find local or global optima, depending on the specific objective function that one is considering.

For a given vertex V_j in the differentiable classification tree model, let such a vertex be defined as a tuple $\langle d^j(\cdot), v^j \rangle$. The function $d^j(\cdot) \in \mathcal{S}$ is again a split function assumed to be taken from the same class of functions $\mathcal{S}$ that we considered in Section 2.4.2. The value $v^j \in \mathbb{R}$ is a numeric attribute that will be used to compute a leaf's output value. The corresponding change to the recursive assignment function $g(\cdot)$ is now given by

$$g(a^i, V_j) \equiv \begin{cases} v^j & \text{if } l(V_j) = r(V_j) = \emptyset, \text{ and} \\ g(a^i, d^j(a^i)) & \text{otherwise.} \end{cases}$$

We define the function $\sigma : \mathbb{R} \to [0, 1]$ to be the logistic sigmoid function,

$$\sigma(v) \equiv \left(1 + e^{-v}\right)^{-1}.$$

The model's output function $f(\cdot)$ is now defined as

$$f(a^i, \theta) \equiv \sigma\left(g(a^i, V_0)\right).$$

Mechanically, this model largely works the same as the one discussed in Section 2.4.2. A given set of attributes a^i is passed down the tree until a leaf V_j is reached, and the leaf's output value v^j is returned. The main difference is that this output is now numeric, rather than bounded to the interval $[0, 1]$, and so we pass it through the function $\sigma(\cdot)$ to enforce the constraint on the model's output domain.

We now move on to discuss how we may train such a model. As was noted before, we no longer have access to the target values that we wish to estimate. We do, however, have access to the gradient information of the objective function, and so we may wish to use this information for the estimation of the model's parameters. We will first look at how one may optimize the output value v^j of a given leaf V_j. Essentially, what we will do is make use of a gradient ascent algorithm to optimize these output values.

Let R_{V_j} denote the set of records assigned to leaf V_j as before. To employ gradient ascent, we are interested in the partial derivative of the objective function with respect to this vertex's output v^j. Using the chain rule for partial derivatives, along with the definition of $\sigma(\cdot)$, we find after some simplification

$$\frac{\partial O(\cdot)}{\partial v^j} = \sigma(v^j)\left(1 - \sigma(v^j)\right) \cdot \sum_{i \in R_{V_j}} \frac{\partial O(\cdot)}{\partial f(a^i, \theta)}.$$

A gradient ascent algorithm that optimizes the value of v^j may now work as follows. Assume v^j initially has some starting value, say v^j_0, and that we are using a step size which we denote with η. At each time step $\tau > 0$ of such an algorithm, we may compute a new value v^j_τ for v^j as

$$v^j_\tau \equiv v^j_{\tau-1} + \eta \cdot \left.\frac{\partial O(\cdot)}{\partial v^j}\right|_{\tau-1},$$

where the term $\left.\frac{\partial O(\cdot)}{\partial v^j}\right|_{\tau-1}$ is understood to be the gradient of $O(\cdot)$ with respect to v^j, evaluated using the values of all the leafs' outputs at time $\tau - 1$.

It may be worth emphasizing at this point that, due to the use of an arbitrary objective function $O(\cdot)$, updating the output of a given leaf may

change the partial derivative of the objective function with respect to the output of a *different* leaf. This is a distinct difference from the objective functions usually used for the binary classification problem, where the partial derivatives of the error function with respect to the outputs of the model are typically independent of each other.

We will next consider the question of how, and when, to split a given leaf V_j. Recall from Section 2.4.2 that the quality of a vertex, as well as the quality of a split, was expressed in terms of the vertices' impurity, which in turn was expressed in terms of the true target values in the dataset. Clearly, we no longer have access to these target values, and so we will need a different way to measure a vertex's quality.

Consider that we do have access to the partial derivatives of the objective function with respect to each output of the model, which is to say, we know $\frac{\partial O(\cdot)}{\partial f(a^i,\theta)}$ for all $1 \leq i \leq n$. Furthermore, the sign of the corresponding gradient information obviously tells us something about the direction that these outputs should be moved in. Also, although optimization of the outputs of the model is clearly restricted by the structure of the tree, we may still use the gradient information of the individual outputs to make inferences about the individual 'desired' outputs.

In particular, positive or negative signs of these individual derivatives tell us if the corresponding outputs of the model should be made higher or lower, respectively, in order to increase the objective function's value. Alternatively, if the partial derivative of the objective function with respect to a particular record is zero, this indicates that the objective function's value is stationary on the corresponding output dimension, and should not be changed (or, rather, the effect of changing this output is unknown).

Consider that for records that are assigned to the same leaf, all corresponding outputs will have to be updated *en bloc*; that is, those records' output values all correspond to the output value of that leaf. Hence, if some of these records have a positive derivative, and others a negative one, there will exist a 'disagreement' about which direction the leaf's output should be moved in. It thus appears to make sense, when looking for a way to quantify a vertex's impurity, to use some quantification of this notion of 'disagreement'. In particular, if we do this, and then split leafs such that the impurity of the resulting children is minimized, we end up with vertices where this disagreement is low; this allows the outputs for as many records as possible with a positive derivative to be increased, and *mutatis mutandis*, for records with a negative derivative.

Observe also the similarity that arises here between the signs of the individual partial derivatives, and the values of the target variable in the binary classification scenario. That is, for the binary classification problem we wanted to split leafs such that as many records as possible with the same target value ended up in the same child vertex; here, we

want to accomplish the same thing, but so that as many records as possible with the same sign of their partial derivatives end up in the same child. We might thus plausibly use a similar quantification of the notion of impurity as was used for the binary classification case. In particular, we will use the Gini-index as a measure of a vertex's impurity, treating the signs of records' partial derivatives as the corresponding 'classes'. We do note that, here, we are dealing with three such classes; positive, negative, and zero partial derivatives. We will thus use the multi-class generalization of the Gini-index to quantify a vertex's impurity.

Finally, we note that, because we are using a general objective function, we cannot assume that the signs of these partial derivatives are always equal for the same record; that is, the value of the sign may depend on the current global position of the model's outputs in the objective function's input space. We may model this by making the quantification of the impurity dependent on the current iteration of the gradient ascent algorithm.

Formally, let $R^{+}_{V_j,\tau}$, $R^{-}_{V_j,\tau}$, and $R^{0}_{V_j,\tau}$ denote the sets of indices of records assigned to vertex V_j, which at time step τ have positive, negative, and zero partial derivatives, respectively. Thus, for example,

$$R^{+}_{V_j,\tau} \equiv \left\{ i \in R_{V_j} \quad \middle| \quad \left. \frac{\partial O(\cdot)}{\partial f(a^i, \theta)} \right|_{\tau} > 0 \right\},$$

and similarly for $R^{-}_{V_j,\tau}$ and $R^{0}_{V_j,\tau}$. We write $\varrho^{+}_{V_j,\tau}$, $\varrho^{-}_{V_j,\tau}$, and $\varrho^{0}_{V_j,\tau}$ for the relative frequency of records with the obvious corresponding values of these signs, e.g.,

$$\varrho^{+}_{V_j,\tau} \equiv \frac{|R^{+}_{V_j,\tau}|}{|R_{V_j}|},$$

and again similarly for $\varrho^{-}_{V_j,\tau}$ and $\varrho^{0}_{V_j,\tau}$.

The impurity measure is now quantified as a function $\phi : \mathcal{V} \times \mathbb{N} \to \mathbb{R}$ that computes the multi-class Gini-index of these relative frequencies in vertex V_j at time step τ. This function is given by the equality

$$\begin{aligned} \phi(V_j, \tau) \equiv\ & \varrho^{+}_{V_j,\tau-1} \cdot \left(1 - \varrho^{+}_{V_j,\tau-1}\right) \\ & + \varrho^{-}_{V_j,\tau-1} \cdot \left(1 - \varrho^{-}_{V_j,\tau-1}\right) \\ & + \varrho^{0}_{V_j,\tau-1} \cdot \left(1 - \varrho^{0}_{V_j,\tau-1}\right). \end{aligned}$$

Observe that if $\phi(\cdot)$ is evaluated at time step τ, we use the values of the signs of the partial derivatives at time step $\tau - 1$. This merely says that we compute the impurity measure at time step τ *before* the output of any vertex is updated.

Now that we have defined an impurity function, we may again use this to define the quality of a split $\Delta_\phi(\cdot)$. This split quality is defined in analogue to the split quality described in Section 2.4.2, with the only difference being that we have to take into account the fact that a vertex's impurity now depends on the current iteration of the gradient ascent algorithm. Thus, we now write the split quality as a function $\Delta_\phi : \mathcal{V} \times \mathcal{S} \times \mathbb{N} \to \mathbb{R}$. As before, this function assumes that a leaf V_j adds two children and uses a valid split function $d^j(\cdot) \in S_{V_j}$. Again denote with $\pi\left(l(V_j)\right)$ and $\pi\left(r(V_j)\right)$ the proportion of records assigned to the left and right child of V_j, respectively. The quality of the split is expressed as the reduction in impurity that would be achieved if the leaf would split at time τ, as opposed to remaining a leaf and updating itself. Thus, we have

$$\Delta_\phi\left(V_j, d^j(\cdot), \tau\right) \equiv \\ \phi\big(V_j, \tau\big) - \left(\pi\left(l(V_j)\right)\phi\big(l(V_j), \tau\big) + \pi\left(r(V_j)\right)\phi\big(r(V_j), \tau\big)\right).$$

Here, when we compute the impurity of the resulting children, i.e., $\phi\big(l(V_j), \tau\big)$ and $\phi\big(r(V_j), \tau\big)$, we do *not* re-compute any partial derivatives. That is, we simply use the values of the signs of the partial derivatives of the records assigned to these children, as computed at the previous time step. The reason that this is valid will be made clear when we consider the final question that we need to address, which is how to initialize the outputs of these children after the best split has been determined.

To discuss this, we will assume that a leaf V_j has been split using some split function $d^j(\cdot) \in S_{V_j}$. As mentioned, the relevant change compared to the functionality described in Section 2.4.2 is that we have to consider how to initialize the output of the newly added children. For the binary classification problem, we were able to analytically compute the outputs of these children, using the true target values of the records assigned to them. Here, we make the observation that we are dealing with a differentiable function, which we optimize by making use of the information of the gradient. Therefore it makes sense to maintain differentiability with respect to the outputs of the model, when new vertices are introduced to the tree. Another way to put this is that when we split a leaf into two children, we would like the corresponding outputs of the model to only change infinitesimally, given that the step size η of our gradient ascent algorithm is small enough. Clearly, we may ensure that this is the case by initializing the outputs of the newly introduced children to the current (that is, non-updated) output of their parent. However, doing so would not change, at the current time step, the corresponding outputs of the model. We may thus also want to update the outputs of the new children in the standard fashion, after they have been initialized in this way.

More formally, we may implement this in the following way. Assume that at time τ of the algorithm, we would like to split a vertex V_j using some function $d^j(\cdot)$. Let the newly introduced children be given by $l(V_j) = V_{j'}$ and $r(V_j) = V_{j''}$ as before. We write $R_{V_{j'}}$ and $R_{V_{j''}}$ for the sets of indices of records that are assigned to the left and right child of V_j, respectively.

What we do to maintain differentiability is pretend that this split was already performed at the previous time step, and that the children's outputs were both initialized to the output of their parent. That is, we pretend that the outputs of the model, at time step $\tau - 1$, were generated using $v^{j'}_{\tau-1} = v^{j''}_{\tau-1} = v^{j}_{\tau-1}$. Clearly, had this split indeed been performed in this way at the previous time step, we would have obtained the same outputs of the model as those that were in fact observed. It follows that the partial derivatives of the objective function with respect to the records assigned to these children are also the same as those that were, in fact, observed. From here, then, we may at time τ update the outputs of the children in the normal fashion, that is, using the equality

$$\begin{aligned} v^{j'}_{\tau} &\equiv v^{j'}_{\tau-1} + \eta \cdot \left.\frac{\partial O(\cdot)}{\partial v^{j'}}\right|_{\tau-1} \\ &= v^{j}_{\tau-1} + \eta \cdot \sum_{i \in R_{V_{j'}}} \left(\frac{\partial O(\cdot)}{\partial f(a^i, \theta)} \frac{\partial f(a^i, \theta)}{\partial v^j} \right) \Bigg|_{\tau-1}, \end{aligned}$$

and similarly for $v^{j''}_{\tau}$. Here, the expansion of the equality follows from the fact that we assumed $v^{j'}_{\tau-1} = v^{j}_{\tau-1}$. We can use this fact so that we do not need to re-compute any partial derivatives, but can simply use those that were already computed for V_j. This also explains why we do not re-compute any partial derivatives when determining the quality of a split: we assumed that the children's outputs were initialized to the output of their parent *before* updating, so re-computing the impurity would just give the same result.

We now have all the parts required to specify the basic algorithm for growing differentiable classification trees. This algorithm is given by Algorithm 3. The algorithm starts by adding a root vertex V_0 to the tree, and initializing the root's output as $v^0 = 0$. This corresponds to saying that we are completely ambivalent about the 'desired' outputs, i.e., we have $f(a^i, \theta) = \frac{1}{2}$, for all $1 \leq i \leq n$. Equivalently, this says that we start in the center of the n-dimensional input space of the objective function. The root is then added to the LeafList, which, as the name might suggest, keeps track of all the leafs in the tree. The algorithm then iterates until the maximum number of iterations (time steps) is reached.

At each iteration of the algorithm, we consider each leaf V_j in turn. If V_j is impure at this iteration, i.e., $\phi(V_j, \tau) > 0$, and has at least one valid

split, we remove V_j from the list of leafs and find the best split function to use. We then introduce two new children to the tree as children of V_j. The outputs of these children are computed, and they are added to the NewLeafList, which keeps track of which vertices will be the tree's leafs at the next iteration. Alternatively, if V_j was pure or had no valid splits to perform, V_j remains a leaf and its output is updated correspondingly.

Naturally, it need not always be necessary to run the algorithm for the maximum number of allowed iterations, if we implement a way to test for convergence. Although being a perhaps somewhat crude approach, we have implemented this by testing whether the gradient with respect to all the leafs' outputs becomes smaller than a given threshold value. If this is the case, and at least a minimum number of iterations have elapsed, we stop the training process even if the maximum number of iterations is not yet reached. The minimum number of iterations proved necessary in our early testing because the algorithm seemed to consistently display a relatively small gradient during the first few iterations. Clearly, only testing whether the gradient vanishes does not tell us whether or not a local maximum is indeed reached. This is to say, the gradient would also vanish at saddle points and local minima of the objective function. However, even in those cases we clearly cannot continue training due to lack of a gradient, and hence this test suffices as an early-stopping condition.

As was the case for the classification tree algorithm described in Section 2.4.2, this algorithm may still have a tendency to overfit, depending on the objective function that is used. Here, one may also want to make use of specifying a minimum number of records (which is to say, more than one) that have to be assigned to each child for a split to be considered valid. Requiring a minimum number of records to be assigned to a leaf itself, in order to allow this leaf to split, may also be beneficial. Finally, we recommend using a post-processing algorithm to reduce the complexity of the tree after the training is completed. We note that for such a post-processing algorithm to meaningfully make decisions about how best to reduce the tree's complexity, it should probably be tailored to the specific objective function that one is using. In Section 4.5, we describe the post-processing algorithm that we have used for the EMM problem considered in this work.

To make clear the necessity for a final extension of this algorithm, we note that, as described, the algorithm is completely deterministic. While for convex objective functions this may not be particularly troublesome, this may not be desired when considering arbitrary objective functions. In particular, if our objective function is highly irregular over its n-dimensional input space, and would thus have many local optima, we may want the algorithm to be able to find more than one local optimum.

Algorithm 3 GRADCAT

Input: A dataset $\mathcal{D}_A$, step size η, minimum iterations $\tau_{\min}$, maximum iterations $\tau_{\max}$, and minimum gradient $\nabla_{\min}$

Output: A classification tree T

$T \leftarrow V_0$
$v^0 \leftarrow 0$
$m \leftarrow 1$
LeafList $\leftarrow \{V_0\}$
for $\tau = 1 \rightarrow \tau_{\max}$ **do**
    NewLeafList $\leftarrow$ LeafList
    for all $V_j \in$ LeafList **do**
        if $\left(\phi(V_j, \tau) > 0\right) \wedge \left(|S_{V_j}| > 0\right)$ **then**
            NewLeafList $\leftarrow$ NewLeafList $\setminus V_j$
            $d^j \leftarrow \arg\max_{d^j \in S_{V_j}} \Delta_\phi(V_j, d^j, \tau)$
            $T \leftarrow T \cup \{V_m, V_{m+1}\}$
            $l(V_j) \leftarrow V_m, \quad r(V_j) \leftarrow V_{m+1}$
            $v^m \leftarrow v_\tau^m, \quad v^{m+1} \leftarrow v_\tau^{m+1}$
            NewLeafList $\leftarrow$ NewLeafList $\cup \{V_m, V_{m+1}\}$
            $m \leftarrow m + 2$
        else
            $v^j \leftarrow v_\tau^j$
        end if
    end for
    LeafList $\leftarrow$ NewLeafList
    if $(\tau \geq \tau_{\min}) \wedge \left(\sum_{V_j \in \text{LeafList}} \left\{\text{Abs}\left(\frac{\partial O(\cdot)}{\partial v^j}\Big|_\tau\right)\right\} < \nabla_{\min}\right)$ **then**
        return T
    end if
end for
return T

One obvious way to solve this problem would be to introduce random restarts, but the question remains of how to implement this. In particular, although the model's output space is continuous over $[0,1]^n$, the discrete nature of the structure of the tree prevents us from easily starting at arbitrary positions within this space. A straightforward solution would be to vary the initial value of the output of the tree's root, i.e., to randomly set v_0^0 at each restart of the algorithm. However, this still has the limitation that these initial positions would be restricted to a one-dimensional subspace of the objective function's n-dimensional input space, viz., the diagonal line between $\mathbf{0}_{n\times 1}$ and $\mathbf{1}_{n\times 1}$.

Although being a perhaps somewhat *ad hoc* solution, what we propose instead is to 'fake' the model's outputs at time $\tau = 0$. This is to say, let $\mathbf{s} \in [0,1]^n$ be a vector in the objective function's input space. We refer to $\mathbf{s}$ as a solution *seed*. We now pretend that at the initial time step $\tau = 0$, the outputs of the model are given by this seed $\mathbf{s}$. Thus, at the initial time step, all gradient information is computed as if $\mathbf{s}$ was the model's true output, and the various impurity measures are computed accordingly. The model's initial update and/or choice of split function to use for the tree's root may then position the model such that further, normal, optimization leads to a local optimum that may be different for various choices of $\mathbf{s}$.

To generate different seeds, we define a probability distribution Ψ over the space $[0,1]^n$, and let $\mathbf{s} \sim \Psi$. Clearly, one could make Ψ be the uniform distribution over $[0,1]^n$, but, whenever possible, it may be beneficial to construct Ψ such that it incorporates information about the objective function that one is using.

It should be noted that after the initial update, the model's output is highly unlikely to actually represent the seed that was used for the restart. Thus, the seed generation process should not be viewed as a way to introduce an initial *position* to use, so much as a way to introduce an initial *direction*. Different seeds within the objective function's input space will reveal, through the function's gradient, different interactions between the function's input variables, and their influence on the value of the objective function. It is this information, and its use during the model's initial update, through which the actual initial *position* is then found.

4.4 SEED GENERATION

As was mentioned in Section 4.3.1, the seed generation process for the GRADCAT algorithm should ideally be tailored to the objective function that one uses. This section describes the distribution of seeds Ψ that we have used for the specific case where the SWSSPD is the objective function used. We do want to remark at this point that the construction

of Ψ was done in a rather *ad hoc* fashion, and was mostly a process of trial and error in which different distributions were tested.

One property that would seem useful to have in a way to generate seeds, is that the seed generation process would not be prohibitively computationally intensive. This is to say, we would for example like to prevent having to run a separate optimization just to generate a seed to use. Furthermore, we want to incorporate information about the objective function into the generation process. One apparent way to satisfy both criteria would be to see if we can find some way to generate a seed using only information about the influence of individual records on the quality measure. This would prevent having to take into account the interactions between the various records.

A result by Cook [6] allows us to compute such individual information efficiently. In particular, this result considers the case where only a single record is left out of the dataset, i.e., the case where $G = \mathcal{D} \setminus r_i$, for some choice $i \in \{1, \ldots, n\}$. Let $\text{Var}[\hat{\mathbf{y}}]$ and $\text{Var}[\mathbf{e}]$ denote the covariance matrices of $\hat{\mathbf{y}}$ and $\mathbf{e}$ respectively. These are given by

$$\begin{aligned}\text{Var}[\hat{\mathbf{y}}] &= \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \sigma^2, \\ \text{Var}[\mathbf{e}] &= \left(\mathbf{I}_{n\times n} - \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top\right)\sigma^2,\end{aligned}$$

where σ^2 denotes the true variance of the errors ε as specified in Section 2.3. Cook shows that the computation of Cook's Distance, when the subgroup G equals the entire dataset minus the i-th record, reduces to

$$D_{\mathcal{D}\setminus r_i} = \frac{t_i^2}{p} \cdot \frac{\text{Var}[\hat{y}_i]}{\text{Var}[e_i]},$$

where t_i is the i-th studentized residual. In particular, this result shows that we do not need to estimate the vector of coefficients $\hat{\beta}_{\mathcal{D}\setminus r_i}$ for each record in the dataset. Rather, we may compute the distance obtained by excluding a single record using just the information from fitting a model on the entire dataset.

To generate a seed $\mathbf{s}$, we start by computing these individual distances for all n records in the dataset. For each record, we are then left with two tasks; to determine whether to 'include' this record in the seed, and to determine the strength with which the record is included. To determine this inclusion, we use a distance-proportional rejection sampling. This is to say, the probability for the i-th record to be included will be given by

$$\Pr(r_i \text{ is included}) \equiv 1 - \frac{D_{\mathcal{D}\setminus r_i}}{\max\limits_{1\leq j\leq n}\left\{D_{\mathcal{D}\setminus r_j}\right\}}.$$

The reasoning here is that we would like the seed $\mathbf{s}$ to somewhat represent a very crude guess at a soft subgroup with high quality. Note that

a high value for $D_{\mathcal{D}\setminus r_i}$ indicates a large distance when the i-th record is *excluded*, and so we would like to include this record in the subgroup with *low* probability. We would like to emphasize at this point that in no way does having a subgroup of records with individual low distances imply anything about the quality of that subgroup as a whole. In fact, the interactions between the various included records are, in general, *very* influential on the subgroup's quality. Hence, this procedure should really only be viewed as a heuristic for selecting a number of records that may or may not be jointly 'interesting'.

The strength of inclusion (or exclusion) can now be computed as follows. Let $v^1, \ldots, v^n$ denote n dummy variables that are used to determine this inclusion strength, such that $v^i \in \mathbb{R}$ for all $1 \leq i \leq n$. These variables are set to

$$v^i \equiv \begin{cases} D_{\mathcal{D}\setminus r_i} & \text{if } r_i \text{ is included, and} \\ 0 & \text{otherwise.} \end{cases}$$

Thus, if the i-th record is 'included', its inclusion strength is related to the distance $D_{\mathcal{D}\setminus r_i}$. Note that records with smaller distances are more likely to be included than ones with large distances. Hence, this inclusion strength will generally be relatively small compared to the average distance of all individual records.

We still need to bound this inclusion strength to the interval $[0, 1]$ in order for $\mathbf{s}$ to be a valid position within the objective function's input space. We do this using the logistic sigmoid function, so that we have

$$s_i \equiv \left(1 + e^{-v^i}\right)^{-1}, \quad \text{for all } 1 \leq i \leq n.$$

Note that this results in a seed in which no record is truly excluded; if the i-th record is 'excluded', this corresponds to $s_i = \frac{1}{2}$. If, on the other hand, the i-th record is 'included', we have $s_i \geq \frac{1}{2}$, but because these records will typically not have a very large inclusion strength, the actual value of s_i may still be rather close to $\frac{1}{2}$.

Note that the initial output of the GRADCAT model specified in Section 4.3.1 is $\frac{1}{2}$ for all records. Hence, seeds will typically occur in relatively close proximity to this starting position. The gradient information resulting from this seed may thus still roughly approximate the gradient at the true starting position, while still allowing us to introduce some variation into the initial update step.

4.5 PATTERN POST-PROCESSING

In Section 4.3.1, it was noted that the resulting tree should ideally be post-processed after training, in order to prevent over-fitting on the training data. This section describes how this post-processing procedure was

implemented for the specific objective function and problem we are considering.

We first note that the resulting tree does not yet strictly conform to the specification of a pattern in the EMM problem; optimization was done using a soft inclusion measure for subgroups, but this conceptualization is rather meaningless in the context of EMM. Our first post-processing step will therefore be to convert the resulting tree into a pattern in our pattern language $\mathcal{P}$.

We do this by first rounding the outputs of the trained model so that we again obtain a 'hard' inclusion measure. We then convert the tree into a collection of positive classification rules. That is, for each leaf with (rounded) output 1, i.e., the leafs that specify inclusion in the subgroup, the path from the root of the tree to that leaf specifies an inclusion rule. The resulting pattern P will use the set of these rules to specify inclusion. Exclusion is then implicitly defined for records that do not satisfy any of the rules in the pattern.

Somewhat more formally, let a *condition* be a function $c : \mathcal{A} \to \{0, 1\}$ whose domain is the values of a single attribute $\mathbb{A}_j$. We refer to the class of all valid conditions as $\mathcal{C}$. Furthermore, let λ be a *rule* such that it is a set of conditions $\lambda \subset \mathcal{C}$. We define a mapping function $\Gamma : \mathcal{V} \to \wp(\mathcal{C})$ that maps a leaf vertex onto the set of all rules (i.e., the powerset of $\mathcal{C}$). This function $\Gamma(\cdot)$ works by taking a leaf vertex, and mapping all split functions in the vertices on the path from that leaf to the root onto conditions. The resulting set of conditions then yields the rule that is the output of $\Gamma(\cdot)$. We denote the set of all positive classification rules induced on the given tree with Λ, so that we have

$$\Lambda \equiv \left\{ \lambda_j = \Gamma(V_j) \quad \middle| \quad l(V_j) = r(V_j) = \emptyset, \quad \text{Round}\left(\sigma\left(v^j\right)\right) = 1 \right\}.$$

Here, V_j and v^j again denote the j-th vertex in the tree, and its output, respectively. The function $\sigma(\cdot)$ is again the logistic sigmoid function as defined in Section 4.3.1. Finally, the class $\mathcal{L}$ of all possible rule sets may be written as $\mathcal{L} \equiv \wp(\wp(\mathcal{C}))$.

Recall from Section 2.2 that a pattern $P \in \mathcal{P}$ is a function $P : \mathcal{A} \to \{0, 1\}$ that works by checking whether a given set of attributes a satisfies a disjunction of conjunctions of conditions over $\mathcal{A}$. Each rule corresponds to such a collection of conjunctions, and the set of rules corresponds to the disjunctions of these conjunctions. We can thus formalize the function $P(\cdot)$ as

$$P(a) = \max_{\lambda_j \in \Lambda} \left\{ \min_{c_i(\cdot) \in \lambda_j} c_i(a) \right\}.$$

It is this pattern, or rather, this set of rules, on which we will perform our actual post-processing. To allow for this, we will with a slight abuse

of notation write the function of the pattern as $P(a, \Lambda)$, so that we may differentiate between different collections of rules. Likewise, to easily differentiate between the regression coefficients estimated on various subgroups, we will write these as $\hat{\beta}_\Lambda$. Similarly, we write $\varphi_{\mathcal{D}}(\Lambda)$ for the quality of the subgroup defined by a given set of rules.

The post-processing procedure is essentially a three step process, which is executed using a validation dataset that was not used during the training of the tree. Regardless, we will use the same notation that was used previously when discussing the training data. The three post-processing steps are the following.

In the first step, we try to remove rules, and conditions in rules, to maximize the quality on the validation dataset. The motivation for doing this is two-fold. Firstly, because the post-processing is performed on a held-out validation dataset, this procedure works as a way to reduce overfitting that may have occurred on the training data. Secondly, the rule set that we are working with was obtained by performing gradient ascent in a (highly) irregular space, and we are thus almost guaranteed to have ended up in a local optimum. While we cannot expect to reach a global optimum by pruning rules or conditions in rules, this process does allow us to evaluate positions in the search space that may have been unreachable by the gradient ascent procedure.

In the second step, we try to remove superfluous conditions in the remaining rules. It deserves emphasizing at this point that we are *not* trying to improve the quality of the rule set in this step, *per se*. Rather, we try to remove conditions without significantly changing the model that is induced on the subgroup. Firstly, this is done as an application of Occam's razor, that is, we consider a less complex description of a subgroup 'better', in some sense, than a more complex description of the (more 'refined') subgroup, assuming that the model induced on these groups remains the same. This can also be motivated by the fact that, by the statement of the EMM problem, we are looking for subgroups with exceptional *models*. Hence, if a model induced on a subgroup is roughly the same as a model induced on a superset of that subgroup, we are roughly equally interested in these subgroups. However, because the second subgroup has a less complex description, and typically will have larger support in the data, we may thus consider this result 'better', and furthermore expect the subgroup description to generalize better. The second motivation comes more from an end-user point of view, that is, we may expect less complex subgroup descriptions to be more easily interpretable, and thus for such results to have more value for the user, all other things being equal.

In the third and final step, we check whether the regression model induced on the resulting subgroup is significantly different from the model induced on the entire validation dataset. If we do not find a sig-

nificant difference in this final step, the entire result is discarded. The reasoning here again ties back to the occurrence of local optima in the search space. In particular, although we may expect the rule set under consideration to be *a* local optimum, we have no guarantees that it is, in fact, 'interesting'. By testing whether the model fitted to this subgroup is indeed significantly different from the model fitted to the entire dataset, we hope to filter out 'uninteresting' subgroups, so that we may present only 'interesting' results to the end-user.

The remainder of this section discusses these various steps in more detail.

4.5.1 *Quality Maximization*

In the first step, we start by trying to remove entire rules so that the quality of the pruned rule set is maximized. This is done by testing for each rule the quality that would be obtained when this rule is removed. If the highest resulting quality is not lower than the quality of the unpruned rule set, the corresponding rule is removed. Somewhat more formally, we compute

$$\lambda^* = \arg\max_{\lambda_j \in \Lambda} \left\{ \varphi_{\mathcal{D}} \left(\Lambda \setminus \lambda_j \right) \right\},$$

and let the new rule set Λ' be

$$\Lambda' = \begin{cases} \Lambda \setminus \lambda^* & \text{if } \varphi_{\mathcal{D}} \left(\Lambda \setminus \lambda^* \right) \geq \varphi_{\mathcal{D}} \left(\Lambda \right), \text{ and} \\ \Lambda & \text{otherwise.} \end{cases}$$

This process is repeated (using the updated rule set) until no more rules can be removed. A similar, but even greedier, approach is then used to prune individual conditions from the remaining rules.

In particular, we compute for each condition in the remaining rules the quality that would be obtained if that condition were removed. All conditions are then considered in order, starting from the highest resulting quality and working to lower resulting quality. For each condition, we test whether removing it will still improve the quality, given the conditions that were already removed before this condition was considered.

Denote the rule set with which we start this process as Λ_0, which is the rule set resulting from pruning entire rules. We write Λ_m for the rule set that is obtained after the m-th condition has been considered. Let $c_i^j(\cdot)$ be the m-th condition under consideration, and let this be the i-th condition in the j-th rule. We write $\Lambda \setminus c_i^j(\cdot)$ as shorthand for the rule set

with this condition removed, that is, $\Lambda \setminus c_i^j(\cdot) \equiv (\Lambda \setminus \lambda_j) \cup \{\lambda_j \setminus c_i^j(\cdot)\}$. We update the remaining rule set Λ_m such that

$$\Lambda_m = \begin{cases} \Lambda_{m-1} \setminus c_i^j(\cdot) & \text{if } \varphi_{\mathcal{D}}\left(\Lambda_{m-1} \setminus c_i^j(\cdot)\right) \geq \varphi_{\mathcal{D}}\left(\Lambda_{m-1}\right), \text{ and} \\ \Lambda_{m-1} & \text{otherwise.} \end{cases}$$

After all conditions are considered in this way, we have completed the first step of the post-processing, and have hopefully been able to both reduce the amount of over-fitting that occurred during the training process, and moved to a higher quality (local) optimum than the original rule set.

4.5.2 *Pattern Simplification*

In the next step, we try to remove superfluous conditions in order to obtain a simplified representation of the pattern.

Algorithmically, this second step largely works the same as the pruning of individual conditions in the first post-processing step. The difference is that here, we do not try to improve the quality of the subgroup *per se*, but try to prune conditions that do not significantly change the regression coefficients of model induced on the subgroup. The first step is again to order all conditions in the rule set, this time sorted by their influence on the regression coefficients. We do this by computing Cook's Distance between the subgroup defined by the remaining rule set, and the subgroup defined by that rule set but with a condition removed. Somewhat more formally, let Λ and Λ' be two rule sets, such that $\Lambda' \equiv \Lambda \setminus c_i^j(\cdot)$, for some choice of $c_i^j(\cdot) \in \lambda_j \in \Lambda$. Thus, Λ' is obtained by removing a single condition from Λ. We write $D(\Lambda, \Lambda')$ for Cook's Distance computed from the subgroup defined by Λ to the subgroup defined by Λ'.

Let Λ_0 denote the rule set that remains after the first post-processing, i.e., Quality Maximization, step. We then compute for each condition $c_i^j(\cdot)$ the distance between the subgroup defined by Λ_0, and the subgroup defined by $\Lambda_0 \setminus c_i^j(\cdot)$, which is to say, we compute

$$D\left(\Lambda_0, \Lambda_0 \setminus c_i^j(\cdot)\right), \quad \text{for all } c_i^j(\cdot) \in \lambda_j \text{ and } \lambda_j \in \Lambda_0.$$

We order the conditions using these resulting distances, and consider them in order of increasing distance. Thus, the first condition that will be considered is the one that *least* changes the regression coefficients of the model when it is removed, removing the second condition (instead of the first) changes the coefficients a bit more, and so forth.

Let $c_i^j(\cdot)$ and Λ_m again denote the m-th condition under consideration, and the rule set obtained after the m-th condition has been considered,

respectively. We test if removing this condition from the remaining rule set Λ_{m-1} significantly changes the regression coefficients $\hat{\beta}_{\Lambda_{m-1}}$. We define the function $\delta : \mathcal{L} \times \mathcal{L} \times [0,1] \to \{0,1\}$ that indicates whether the coefficients of two subgroups defined by different rule sets are significantly different at the specified confidence level, which is to say

$$\delta(\Lambda, \Lambda', \alpha) \equiv \begin{cases} 1 & \text{if } \hat{\beta}_\Lambda \text{ and } \hat{\beta}_{\Lambda'} \text{ are significantly different} \\ & \text{at the } (1-\alpha) \times 100\% \text{ confidence level, and} \\ 0 & \text{otherwise.} \end{cases}$$

The function $\delta(\cdot)$ works by testing for intersection of the $(1-\alpha) \times 100\%$ confidence ellipsoids of the two coefficient vectors $\hat{\beta}_\Lambda$ and $\hat{\beta}_{\Lambda'}$. The procedure for doing this is, however, somewhat involved, and so we refer the interested reader to Appendix C for its explication. We do want to note here that our use of this test in this setting is not statistically rigorous, in the sense that $\hat{\beta}_\Lambda$ and $\hat{\beta}_{\Lambda'}$ may have been estimated using overlapping records in the dataset, and were thus not obtained independently. Hence, this procedure should really be viewed as a heuristic method for testing whether these coefficient vectors are different, as opposed to a test from which meaningful statistical inferences can be made.

The actual pruning procedure largely follows the algorithm that was used in the first post-processing step. Let Λ_m denote the rule set obtained after the m-th condition has been considered, and let $c_i^j(\cdot)$ denote this m-th condition as before. We update the rule set as

$$\Lambda_m = \begin{cases} \Lambda_{m-1} \setminus c_i^j(\cdot) & \text{if } \delta\left(\Lambda_{m-1}, \Lambda_{m-1} \setminus c_i^j(\cdot), \alpha\right) = 0, \text{ and} \\ \Lambda_{m-1} & \text{otherwise.} \end{cases}$$

Thus, we try to prune conditions without significantly changing the regression coefficient vector of the subgroup. Once we have considered all conditions in this fashion, the second post-processing step is completed.

4.5.3 *Interestingness Assertion*

In the third and final step, we check whether the regression coefficients of the resulting subgroup differ significantly from the coefficients obtained from fitting a model to the entire dataset. To describe this procedure using our existing notation, we specify a rule set $\bar{\Lambda}$ that covers the *entire* dataset, i.e., we choose $\bar{\Lambda}$ such that $P(a^i, \bar{\Lambda}) = 1$, for all $1 \leq i \leq n$. Furthermore, simply denote with Λ the rule set that is obtained after the second post-processing step. This set of rules, or rather the corresponding pattern, is accepted if and only if $\delta(\Lambda, \bar{\Lambda}, \alpha) = 1$. Thus, patterns that describe subgroups whose regression coefficients do not, at the specified confidence level, differ significantly from the coefficients of the model fitted to the entire dataset, are discarded. Clearly, our earlier reservations

regarding the interpretation of the results of $\delta(\cdot)$ as a rigorous statistical test also apply here.

4.6 GRADCAT-EMM

In this section, we formalize how the GRADCAT algorithm discussed in Section 4.3.1, and the seed generation and post-processing functionality discussed in Section 4.4 and Section 4.5 respectively, tie together as a heuristic for solving the EMM problem for linear regression models.

We assume that we are given a training dataset $\mathcal{D}$ of the form discussed in Section 2.2. Furthermore, we assume the specification of a desired number of patterns to return, which we will denote with ω. The GRADCAT-EMM algorithm is then given by Algorithm 4.

Algorithm 4 GRADCAT-EMM for Linear Regression

Input: A dataset $\mathcal{D}$ and desired number of results ω
Output: A multiset of patterns Ω

$\Omega \leftarrow \emptyset$
while $|\Omega| < \omega$ **do**
  $\mathcal{D}_{\text{Train}} \leftarrow \text{Sample}(\mathcal{D})$
  $\mathcal{D}_{\text{Test}} \leftarrow \mathcal{D} \setminus \mathcal{D}_{\text{Train}}$
  $\mathbf{s} \leftarrow \text{GenerateSeed}(\mathcal{D}_{\text{Train}})$
  $T \leftarrow \text{GRADCAT}(\mathbf{s}, \mathcal{D}_{\text{Train}})$
  $P \leftarrow \text{PostProcess}(T, \mathcal{D}_{\text{Test}})$
  if $P \neq \emptyset$ **then**
    $\Omega \leftarrow \Omega \cup \{P\}$
  end if
end while

The algorithm starts with an empty multiset of results Ω, and then tries to find the specified number of 'interesting' patterns. At each iteration, the dataset $\mathcal{D}$ is first uniformly sub-sampled into a training and test dataset. The function GenerateSeed$(\cdot)$ then generates a seed $\mathbf{s}$ from the dataset $\mathcal{D}_{\text{Train}}$ using the procedure outlined in Section 4.4. The function GRADCAT$(\cdot)$ takes this seed $\mathbf{s}$ and the training data $\mathcal{D}_{\text{Train}}$, and generates a classification tree T using the GRADCAT algorithm from Section 4.3.1. Obviously, the objective function used here is the function $O(\cdot)$ defined by Equation 4.3 in Section 4.2. The resulting tree T is subsequently post-processed on the dataset $\mathcal{D}_{\text{Test}}$ using the function PostProcess$(\cdot)$. This function implements the post-processing procedure detailed in Section 4.5, and yields a pattern P. If the pattern was rejected as not significantly different during this post-processing, we write $P = \emptyset$. Alternatively, if the pattern was accepted, it is added to the mul-

tiset of results Ω. The algorithm then iterates until the desired number of patterns have been found.

Note that we have left a number of parameters for the algorithm implicit in this specification. In particular, the training speed for the GRADCAT algorithm, as well as the minimum number of records required for a split in the classification tree to be considered valid, still need to be specified. We also need the specification of the stop-condition parameters of the GRADCAT algorithm, viz., the minimum and maximum number of iterations, and the minimum gradient threshold. Finally, we require the specification of a confidence level $(1 - \alpha)$ to use during the post-processing step of the algorithm.

The resulting multiset of patterns Ω may now be evaluated on a different dataset $\mathcal{D}_{\text{Evaluation}}$, and the resulting patterns ranked in order of decreasing quality. These results may then be presented to an end-user for interpretation.

5

QUALITATIVE ANALYSIS

In this chapter, we discuss qualitatively some observations regarding the GRADCAT-EMM algorithm. We begin by discussing several interpretations of the GRADCAT strategy in Section 5.1. A comparison is then made, in Section 5.2, with a different tree-based search heuristic, which on first sight may seem like a more straightforward approach to use. We discuss, qualitatively, some differences in how GRADCAT-EMM and Beam Search solve different problems encountered in the EMM setting, in Section 5.3. In Section 5.4, we show that the general problem of Subgroup Discovery can be seen to be a special case of EMM with linear regression models, and that as such the GRADCAT-EMM algorithm can also immediately be used in that setting. We finish by giving in Section 5.5 a worked numeric example of how three different algorithms solve the EMM problem on a small example dataset.

5.1 INTERPRETATIONS OF GRADCAT

Here, we discuss two different interpretations of the GRADCAT model detailed in Section 4.3.1. The first interpretation pertains mostly to how the model works mechanically, and corresponds to the adapted classification tree interpretation which was used when introducing the model. A different interpretation of the model, or more precisely of the algorithm, is as a particular form of constrained gradient ascent. The latter interpretation allows for an arguably better explanation of what the model does conceptually, which is to say, of the purpose that it serves within the context of EMM.

The remainder of this section discusses these interpretations in more detail.

5.1.1 *Differentiable Classification Tree*

The 'mechanistic', or 'differentiable classification tree', interpretation of the GRADCAT model corresponds to how the algorithm was introduced in Section 4.3.1. This interpretation mostly explains *how* the model works, and we feel that it allows for one of the more intuitive expositions of the workings of the algorithm.

This is mostly because it allows us to relate the model to the notion of classification trees, which is a rather well-known model from the literature. This is, of course, not a coincidence; the classification tree model

is, as mentioned, the basis from which the GRADCAT algorithm was developed. Because this interpretation was also how we initially presented the model, we will not expand on it much longer.

However, as a final note on this interpretation, we have a somewhat interesting result regarding the relationship between GRADCAT and the CART classification tree algorithm. If one uses the log-likelihood function as the objective function being optimized by the GRADCAT model, it can be shown that, under some mild assumptions, the resulting model will converge to the CART model, assuming of course that the same dataset is used for both models. Specifically, this convergence holds for the structure of the resulting trees, and the outputs of the models for all records in the dataset. Particularly, the pruning algorithm employed by CART is not taken into account. We do not give the corresponding proof here, but it is included in Appendix A for the interested reader.

5.1.2 *Tree-Constrained Gradient Ascent*

A different interpretation of the GRADCAT algorithm is as a particular form of constrained gradient ascent. That is, the inclusion of records in the subgroup is optimized through gradient ascent, but under the constraint that this inclusion must be optimized *en bloc* for all records falling into the same leaf of the tree.

Under this interpretation, then, there are two largely distinct phases implicit in the algorithm. In the first phase, the actual tree is grown, whereas in the second phase, the outputs of the tree's leafs are optimized. Loosely speaking, in the first phase, the algorithm finds out which constraints to use, and in the second phase, these constraints are used to perform a form of constrained gradient ascent[1].

This implicit division into two phases largely follows from our use of the Gini-index as the impurity function of the tree's vertices. Briefly, without going into the full proof, it follows from the convexity of the Gini-index that at any given time step of the algorithm, the quality of a split can never be negative. Hence, as long as vertices are still impure and have at least one valid split they can perform, the algorithm will do so. Clearly, this is a process that happens only during the first few iterations of the algorithm, because the number of records assigned to a leaf is on average halved at each split. Thus, the expected number of

1 Strictly speaking, this is not entirely correct, but we consider the conceptualization to be helpful. Specifically, if one were to perform constrained gradient ascent using numerical optimization, one would have to project the gradient of the objective function onto the constraint-surface. In the GRADCAT algorithm, this can be shown to correspond simply to dividing the gradient ascent step-size, for each leaf, by the number of records assigned to that leaf. In our development of the algorithm, we have instead used the standard calculus approach, in which this re-scaling cannot be derived.

iterations that the algorithm will spend on growing the tree is roughly logarithmic in the size of the dataset.

The only leafs that may still split later in the process, are those that were initially considered pure, but which later became impure due to a subsequent sign change of the derivatives of records assigned to it. That is, it is possible that all records assigned to a leaf initially have the same signs of their derivatives, but that this later changes due to the global position of the model's outputs in the objective function's input space. Recall that this can happen due to the, in general, non-zero second order mixed partial derivatives of the objective function with respect to the tree's outputs.

In general, then, it may be assumed that this growing of the tree is largely completed well before the outputs of the model have converged. Hence, the remainder of the time will be spent optimizing the fully grown tree's outputs, which we here refer to as the 'second phase' of the algorithm.

To see how this interpretation may shed some light on how the algorithm works, recall that the impurity of a vertex was expressed in terms of the signs of the derivatives of the records assigned to it. Reducing this impurity, then, involves finding the split that ensures that each resulting partition contains as many records with the same sign of their derivative as possible. Clearly, this allows the gradient ascent update-step to move the inclusion weights of as many records as possible into their 'desired' directions. Informally, the 'first phase' of the algorithm may thus be explained as trying to find the *weakest* possible constraint on the objective function, that still conforms to the tree structure. That is, we are trying to find the structure of the tree that gets in the way of the 'ideal', i.e., unconstrained, update as little as possible.

Mainly, this serves two purposes. The first, of course, is that the resulting constraints will not interfere with the optimization too much, allowing the gradient ascent part of the algorithm to find a (hopefully) good local optimum. More important, however, is that it gives us a representation of this resulting optimum that is expressible in a rule-based format.

Thus, with the GRADCAT algorithm, we try to maintain the best of two worlds; we use the fine-grained information contained in individual records' gradients, and have the gradual optimization of the model's outputs, which we consider desirable properties of gradient ascent. On the other hand, we use an easily interpretable tree structure to grow a 'scaffolding' within which the gradient ascent algorithm's optimum must be obtained, enabling us to readily express this optimum using a rule-based representation.

5.2 GRADCAT-EMM VS. 'TREE-BASED PARTITIONING'

One question that could legitimately be asked about the use of the GRADCAT algorithm pertains to the advantages that if offers over more 'traditional' tree-based approaches. We will here briefly compare the GRADCAT-EMM algorithm to one of the more straightforward conceptualizations of such a more standard approach. We will refer to this other tree-based method as 'tree-based partitioning', or TBP for short.

Essentially, the TBP method will be based on the observation that most tree-based methods, in the current context, may be regarded as a partitioning of an attribute space $\mathcal{A}$, such that all partitions are disjoint, and where the union over all partitions again yields the complete space $\mathcal{A}$. Furthermore, such a model will then generally formulate a certain property, that it stipulates holds equally for all records assigned to the same partition. For example, if we interpret the CART model in this fashion, then each leaf vertex corresponds to such a partition of $\mathcal{A}$, and the model stipulates a prediction $\hat{t}$ for a record's target value that is equal for all records assigned to the same leaf.

Similarly, the GRADCAT tree in the GRADCAT-EMM algorithm constructs a partitioning of $\mathcal{A}$, and the degree of inclusion in the subgroup is stipulated to be equal for all records falling into the same leaf.

Now, we formulate the TBP model as a tree-based partitioning of $\mathcal{A}$, with the stipulation that all records assigned to a given leaf together constitute a subgroup. Thus, the TBP model expresses one subgroup for every leaf in such a tree. Conceptually, this appears to be a lot more straightforward than the approach used by the GRADCAT-EMM algorithm.

Further, because each leaf here corresponds to a subgroup proper, we could grow such a tree by simply expressing the quality of a split in terms of the quality $\varphi_{\mathcal{D}}(\cdot)$ of the subgroups defined by the involved vertices. Formally, let $\varphi_{\mathcal{D}}(V_j)$ be the quality of the subgroup defined by a leaf V_j. For a given split $d^j(\cdot)$ that splits a vertex V_j into children $V_{j'}$ and $V_{j''}$, let the quality of this split be given by

$$\Delta_{\varphi}(V_j, d^j(\cdot)) \equiv \Big(\pi(V_{j'})\varphi_{\mathcal{D}}(V_{j'}) + \pi(V_{j''})\varphi_{\mathcal{D}}(V_{j''})\Big) - \varphi_{\mathcal{D}}(V_j). \quad (5.1)$$

Thus, we express the quality of a split as the weighted average of the quality of the newly created children, minus the quality of the original leaf. We then grow the tree by splitting leafs using the split with maximum quality, whenever the quality of this best split is non-negative.

Now, although both methods are based on a tree-based partitioning of $\mathcal{A}$, comparison of TBP with GRADCAT-EMM will reveal that the methods are in fact not very similar. Besides the fact that the TBP tree expresses multiple subgroups simultaneously, and the GRADCAT tree just

a single one, some consideration will reveal that TBP is much greedier, on a per-subgroup level, than GRADCAT-EMM.

In fact, this can be attributed *precisely* to the property that every partition in TBP expresses a distinct subgroup. Contrariwise, if we consider the GRADCAT tree with outputs rounded to zero or one, we see that the subgroup is defined by the union of all partitions whose (rounded) degree of inclusion is equal to one. Thus, TBP may return low quality local optima due to an 'erroneous' split early on, whereas the GRADCAT tree might still recover by taking the union over the resulting branches.

Further, by only considering the quality of complete subgroups, TBP ignores information contained in individual records' influence on the subgroup's quality. In this sense, and because it searches for multiple subgroups in parallel, it is perhaps in fact more similar to beam search than to GRADCAT-EMM. However, because all these subgroups must, by construction, be disjoint, and together cover the entire space $\mathcal{A}$, TBP can be seen to be much more restricted than beam search.

In summary, it is not obvious that a more 'traditional' tree-based search heuristic would have advantages over either GRADCAT-EMM or beam search. Further, although GRADCAT performs a tree-based partitioning of $\mathcal{A}$, the *subgroup* that is being optimized by GRADCAT-EMM does not, in general, conform to a single such partition. In this sense, GRADCAT-EMM is conceptually rather hard to compare with TBP, although it seems clear that it is mechanically less greedy than the latter.

5.3 COMPARISON TO BEAM SEARCH

As was the case with the 'tree-based partitioning' method outlined in Section 5.2, it is somewhat hard to find obvious conceptual similarities between beam search and GRADCAT-EMM. We may, however, examine some of the common problems that would typically be encountered by both search strategies, and see how they are respectively solved by the two algorithms.

One observation is that we are in the EMM problem generally not just interested in the single highest-quality subgroup, but rather in a *set* of high quality subgroups. The GRADCAT-EMM algorithm handles this by both sub-sampling the training data, which may give rise to different local optima in the search space, and by using random 'seeds' from which the tree is grown, which may give rise to different solutions even on the same, or very similar, training data. By performing multiple random restarts in this fashion, the algorithm can find collections of different subgroups.

In contrast, the beam search algorithm explicitly keeps track of the best ω subgroups it has encountered during the search. This has the advantage that, if a given subgroup *can* be found by the algorithm, it

will always be in the set of returned results. Contrariwise, the random restarts employed by GRADCAT-EMM do not give such guarantees, in that it may in principle be possible to find a certain subgroup, but that it is never found because the required seed is never used.

Another, albeit somewhat impractical, advantage of the beam width parameter ω, is that it makes it possible to reduce the algorithm to an exhaustive search. That is, on any given dataset, it is possible to choose ω so that the *entire* search space is included in the algorithm's beam. Hence, we might say that, in this rather theoretical sense, the beam search algorithm is more powerful than GRADCAT-EMM, because it can in principle find all possible subgroups. Of course, it is debatable whether this result has much value in practice.

A second difference between the two algorithms is how they search for high quality subgroups. If we consider for a moment the special case of beam search where the beam width $\omega = 1$, we see that the algorithm is essentially a straightforward hill-climber, and is hence very greedy when viewed on a per-subgroup level. Of course, this is balanced by the fact that it maintains multiple candidate solutions in its beam, which reduces the risk of getting stuck in a single local optimum. Somewhat more to the point, however, is that this greedy hill-climbing is performed in the attribute space $\mathcal{A}$, and on the quality $\varphi_{\mathcal{D}}(\cdot)$ of a proper subgroup.

In contrast, GRADCAT-EMM does *not* directly seek to optimize $\varphi_{\mathcal{D}}(\cdot)$ in the space $\mathcal{A}$. Rather, the quality $\varphi_{\mathcal{D}}(\cdot)$ is optimized through gradient ascent in a subspace of the inclusion-weight space $[0,1]^n$, and it is this *subspace* which is constructed by hill-climbing in $\mathcal{A}$. That is, the GRADCAT tree performs hill-climbing on some implicit objective function, by partitioning the attribute space $\mathcal{A}$. Gradient ascent is then performed *en bloc* on the inclusion weights of all records falling into the same partition. Of course, the implicit objective function that is optimized by constructing this partitioning still *relates* $\varphi_{\mathcal{D}}(\cdot)$ to the greedy hill-climbing performed in $\mathcal{A}$. However, it is not obvious how this affects the algorithm's greediness with respect to searching for the *subgroup*. That is, while the partitioning itself is very greedy, it does not by itself imply anything about records' inclusion in the subgroup. Instead, it is fully possible for the gradient ascent part of the algorithm to initially 'include' the records in a given partition in the subgroup, and to later 'remove' them again due to a sign change of their derivatives. This latter property is due to the gradual optimization of $\varphi_{\mathcal{D}}(\cdot)$ in the subspace of $[0,1]^n$, and in this sense one may say that $\varphi_{\mathcal{D}}(\cdot)$ is optimized less greedily than it would be with a direct hill-climber in $\mathcal{A}$.

Another point that the algorithms address rather differently is how to correctly choose the proper complexity of the subgroups' descriptions. In the beam search algorithm this is largely parameterized, and hence chosen by the user, viz., by stipulating the search depth of the algorithm.

The complexity of the results may also be controlled less directly by specifying the minimum support of the subgroups.

One limitation of beam search pertains specifically to the requirement that this search depth must be specified in advance. That is, if one does not search deep enough, the algorithm may fail completely to find 'interesting' subgroups. On the other hand, if one searches only one level too deep, the algorithm will have a tendency to saturate its set of results with minor, largely non-informational, variations of the best subgroups found on the previous level [23]. If, ideally, all 'correct' subgroups have the same complexity, i.e., can be found on the same search depth, this would not be that much of a problem. That is, one could simply search up to various depths, and choose a set of results having both high quality and large diversity within the set of results. However, this may be an issue if multiple interesting subgroups exist at different search depths, because saturation of the beam with variations of one such subgroup may prevent the algorithm from being able to reach these other groups.

In contrast, the GRADCAT-EMM algorithm does not offer a parameter to directly control the subgroups' complexity. Rather, the GRADCAT part of the algorithm will typically find a tree where several branches together specify a subgroup with a rather complex description. Subsequent post-processing of this subgroup will then maximize its quality on a held-out evaluation dataset by reducing the description's complexity, in order to both reduce overfitting, and to escape from local optima. The post-processing procedure will next attempt to remove superfluous conditions from the remaining subgroup description, in order to further reduce complexity. Thus, the complexity of the GRADCAT-EMM algorithm's results can be influenced indirectly by the user through the specification of the pruning level α. Further, related to the minimum support parameter of the beam search algorithm, the complexity may also be influenced by specifying the minimum number of records required for a split in the GRADCAT tree to be valid.

It is of note that, with exception of the part where the GRADCAT tree is converted into a proper set of 'hard' inclusion rules, the post-processing procedure employed by GRADCAT-EMM is completely agnostic as to where this set of rules originated from. Hence, it may be worthwhile to investigate whether we can improve on the beam search algorithm's performance by also applying this pruning process to *its* returned results. Therefore, we will also evaluate the performance of this composite algorithm in the experiments described in Chapter 6. This may also help to elucidate whether or not any observed difference in performance between beam search and GRADCAT-EMM should be attributed solely to either the GRADCAT part, or the post-processing element, of the algorithm proper.

5.4 APPLICATION TO SUBGROUP DISCOVERY

We note that by applying the EMM framework to linear regression models, there arises an interesting special case that relates it to Subgroup Discovery [18] (SD). Recall that in the SD setting, we are looking for subgroups in which a *single* target variable is either substantially higher or lower than the value of that variable in the entire dataset. Typically, we would for example look at the average of that target variable in a subgroup, and try to find subgroups for which this average is as high as possible.

Now, consider the special case of EMM with linear regression, where we use only an intercept term in the regression model, i.e., we do not use any explanatory variables. Thus, we assume the use of a dataset with records of the form

$$r \equiv \langle a_1, \ldots, a_k, x \rangle.$$

Then, the only coefficient of the regression model, say β_0, is the intercept, and it is well known that the least squares estimate of β_0 is $\hat{\beta}_0 = \bar{x}$. If we now consider the objective function $O(\cdot)$ defined by Equation 4.3, we find after some simplification

$$O(\cdot) = \mathrm{Tr}(\mathbf{W}) \cdot \frac{(\bar{x}_G - \bar{x})^2}{2s^2}.$$

Ignoring for a moment the factor $\mathrm{Tr}(\mathbf{W})$, we clearly see that maximization of this function will yield subgroups in which the average target variable's value is either substantially higher or lower than the average value in the entire dataset. Mainly, the difference with SD, as we defined it previously, is that we are not looking explicitly for either a high *or* a low value, but consider both to be equally interesting. Finally, the factor $\mathrm{Tr}(\mathbf{W}) = \sum_{i=1}^{n} w_i$ clearly introduces a trade-off between the support of the resulting subgroup, and the exceptionality of the subgroup's average target value.

Furthermore, if we consider the case where the inclusion weights are taken to be discrete, that is, $w_i \in \{0, 1\}$ for all $1 \leq i \leq n$, we may clearly write the quality measure as

$$\begin{aligned} O(\cdot) =& |G| \cdot \frac{(\bar{x}_G - \bar{x})^2}{2s^2} \\ \propto& |G| \cdot (\bar{x}_G - \bar{x})^2. \end{aligned}$$

Observe the similarity between this measure, and the *Mean Test* (MT) quality measure proposed for the SD problem by Klösgen [17], which it was noted [23], if one is interested in both exceptionally high and low target values, may be written as

$$\mathrm{MT}(\cdot) = \sqrt{|G|} \cdot \left(|\bar{x}_G - \bar{x}|\right).$$

Here, the factor $|\bar{x}_G - \bar{x}|$ denotes the absolute value of the difference between $\bar{x}$ and $\bar{x}_G$. Clearly, we see that the objective function $O(\cdot)$ is proportional to the square of the MT function. Hence, an ordering by quality of all subgroups that can be induced on a given dataset will be the same under both measures.

In summary, these relationships show that we may regard conventional SD as a special case of EMM with linear regression models. Hence, we will, in the experiments described in Chapter 6, also take into account this instantiation of the problem in the evaluation of the GRADCAT-EMM algorithm.

5.5 A NUMERIC EXAMPLE

In this section, we will illustrate with a simple numeric example how the different algorithms, and particularly the GRADCAT-EMM algorithm, seek to find high quality subgroups.

For simplicity, we will make use of the special case of EMM with linear regression models that was described in Section 5.4. That is, we use only a single target variable, which means that we may interpret this EMM problem as an instantiation of Subgroup Discovery. To further simplify matters, the target variable and the descriptive attributes are all binary variables.

The dataset that we will use is shown in Table 5.1. There are 15 records in total, each having three binary descriptive attributes $a_1, \ldots, a_3$, and a binary target variable x. The global mean of the target variable, $\bar{x}$, is $\bar{x} = \frac{8}{15}$. The global variance, s^2, is $s^2 = \frac{8}{30}$.

This dataset was constructed specifically to have a 'decoy' effect for single-condition subgroup descriptions. That is, in the space of purely conjunctive subgroup descriptions, the global optimum of $\varphi_{\mathcal{D}}(\cdot)$ is obtained at $(a_1 = 1) \wedge (a_2 = 1)$. However, out of the six possible single-condition subgroup descriptions, the subgroups described by $(a_1 = 1)$ and $(a_2 = 1)$ have the lowest quality. This was done specifically to illustrate how the different algorithms cope with such a decoy effect.

For ease of reference, Table 5.2 shows the complete space of purely conjunctive subgroup descriptions. Included here is the subgroup description, the size $|G|$ of the described subgroup, the number of records with $x = 1$ within that subgroup, and the corresponding quality $\varphi_{\mathcal{D}}(\cdot)$.

In the following sections, we will show how the different algorithms work on this dataset. We begin by demonstrating the beam search algorithm in Section 5.5.1. Section 5.5.2 then briefly illustrates the 'tree-based partitioning' method which we discussed in Section 5.2. We finally give a more in-depth demonstration of the GRADCAT-EMM algorithm in Section 5.5.3. Because of the relatively small size of the dataset, the minimum support of a subgroup is stipulated to be 1 for all algorithms.

ID	a_1	a_2	a_3	x	ID	a_1	a_2	a_3	x	ID	a_1	a_2	a_3	x
1	1	1	1	1	6	0	0	1	1	11	1	0	1	1
2	1	1	1	1	7	0	0	1	1	12	1	0	1	0
3	1	1	1	1	8	0	0	1	0	13	0	1	1	1
4	1	0	0	0	9	0	1	0	0	14	0	1	1	0
5	1	0	0	0	10	0	1	0	0	15	1	1	0	1

Table 5.1: Dataset used in the numeric example. Each record has 3 binary descriptive attributes, $a_1, \ldots, a_3$, and a single binary target variable x.

5.5.1 *Beam Search*

To use the beam search algorithm, we will first need to specify the search depth and beam width that the algorithm will use. We will here demonstrate the algorithm with MaxDepth $= 3$ and $\omega = 4$. This choice of the algorithm's beam width ω ensures that the algorithm is affected by the 'decoy' effect that we mentioned in Section 5.5. That is to say, only by choosing a larger beam width will the algorithm be able to reach the global optimum. However, we feel that this choice of ω may be useful for didactic purposes precisely because of this reason.

The algorithm starts by evaluating all single-condition subgroup descriptions, and adding the best ω of these descriptions to its beam $\mathcal{B}$. Inspection of Table 5.2 shows that the algorithm's beam at Depth $= 1$ is correctly shown by Table 5.3.

The algorithm proceeds to generate a set of candidates to include at Depth $= 2$, which corresponds to the set of all refinements of the descriptions included in its beam at Depth $= 1$, along with the set of these descriptions themselves. This set of candidates is shown in order of decreasing quality in Table 5.4. Note that the global optimum $(a_1 = 1) \wedge (a_2 = 1)$ is the only two-condition description that could not be reached from the descriptions contained in the algorithm's beam at the previous depth. The algorithm now selects the best ω subgroup descriptions from this set of candidates, which together constitute the algorithm's beam at Depth $= 2$. This set of subgroup descriptions is shown in Table 5.5.

As before, the algorithm now generates a set of candidates to use at Depth $= 3$, which is the union of all refinements of the descriptions in its beam at the previous depth, and the descriptions in the beam themselves. Note that because the description $(a_3 = 0)$ was already included in the set of candidates at the previous level, it no longer needs to be evaluated for refinement, although it will again be included itself in the new set of candidates. These candidates are shown in Table 5.6.

DESCRIPTION	$\lvert G \rvert$	$\sum_{r_i \in G} x^i$	$\varphi_{\mathcal{D}}(\cdot)$
$\emptyset$	15	8	0
$(a_1 = 0)$	7	3	0.144
$(a_1 = 1)$	8	5	0.126
$(a_2 = 0)$	7	3	0.144
$(a_2 = 1)$	8	5	0.126
$(a_3 = 0)$	5	1	1.042
$(a_3 = 1)$	10	7	0.521
$(a_1 = 0) \wedge (a_2 = 0)$	3	2	0.100
$(a_1 = 0) \wedge (a_2 = 1)$	4	1	0.602
$(a_1 = 1) \wedge (a_2 = 0)$	4	1	0.602
$(a_1 = 1) \wedge (a_2 = 1)$	4	4	1.633
$(a_1 = 0) \wedge (a_3 = 0)$	2	0	1.067
$(a_1 = 0) \wedge (a_3 = 1)$	5	3	0.042
$(a_1 = 1) \wedge (a_3 = 0)$	3	1	0.225
$(a_1 = 1) \wedge (a_3 = 1)$	5	4	0.667
$(a_2 = 0) \wedge (a_3 = 0)$	2	0	1.067
$(a_2 = 0) \wedge (a_3 = 1)$	5	3	0.042
$(a_2 = 1) \wedge (a_3 = 0)$	3	1	0.225
$(a_2 = 1) \wedge (a_3 = 1)$	5	4	0.667
$(a_1 = 0) \wedge (a_2 = 0) \wedge (a_3 = 0)$	0	0	$\emptyset$
$(a_1 = 0) \wedge (a_2 = 0) \wedge (a_3 = 1)$	3	2	0.100
$(a_1 = 0) \wedge (a_2 = 1) \wedge (a_3 = 0)$	2	0	1.067
$(a_1 = 0) \wedge (a_2 = 1) \wedge (a_3 = 1)$	2	1	0.004
$(a_1 = 1) \wedge (a_2 = 0) \wedge (a_3 = 0)$	2	0	1.067
$(a_1 = 1) \wedge (a_2 = 0) \wedge (a_3 = 1)$	2	1	0.004
$(a_1 = 1) \wedge (a_2 = 1) \wedge (a_3 = 0)$	1	1	0.408
$(a_1 = 1) \wedge (a_2 = 1) \wedge (a_3 = 1)$	3	3	1.225

Table 5.2: The exhaustive space of all purely conjunctive subgroup descriptions of the dataset shown in Table 5.1. The first column shows the subgroup descriptions. The second column shows the size of the described subgroup, and the third column the number of records within that subgroup with target value $x = 1$. The rightmost column shows the descriptions' quality.

BEAM SEARCH'S BEAM - DEPTH $= 1$			
DESCRIPTION	$\varphi_{\mathcal{D}}(\cdot)$	DESCRIPTION	$\varphi_{\mathcal{D}}(\cdot)$
$(a_3 = 0)$	1.042	$(a_1 = 0)$	0.144
$(a_3 = 1)$	0.521	$(a_2 = 0)$	0.144

Table 5.3: Subgroup descriptions, and their quality, in the depth 1 beam of the beam search algorithm.

BEAM SEARCH'S CANDIDATES - DEPTH $= 2$			
DESCRIPTION	$\varphi_{\mathcal{D}}(\cdot)$	DESCRIPTION	$\varphi_{\mathcal{D}}(\cdot)$
$(a_1 = 0) \wedge (a_3 = 0)$	1.067	$(a_1 = 1) \wedge (a_3 = 0)$	0.225
$(a_2 = 0) \wedge (a_3 = 0)$	1.067	$(a_2 = 1) \wedge (a_3 = 0)$	0.225
$(a_3 = 0)$	1.042	$(a_1 = 0)$	0.144
$(a_1 = 1) \wedge (a_3 = 1)$	0.667	$(a_2 = 0)$	0.144
$(a_2 = 1) \wedge (a_3 = 1)$	0.667	$(a_1 = 0) \wedge (a_2 = 0)$	0.100
$(a_1 = 0) \wedge (a_2 = 1)$	0.602	$(a_1 = 0) \wedge (a_3 = 1)$	0.042
$(a_1 = 1) \wedge (a_2 = 0)$	0.602	$(a_2 = 0) \wedge (a_3 = 1)$	0.042
$(a_3 = 1)$	0.521		

Table 5.4: Beam search's depth 2 candidate subgroup descriptions, and their quality.

BEAM SEARCH'S BEAM - DEPTH $= 2$			
DESCRIPTION	$\varphi_{\mathcal{D}}(\cdot)$	DESCRIPTION	$\varphi_{\mathcal{D}}(\cdot)$
$(a_1 = 0) \wedge (a_3 = 0)$	1.067	$(a_3 = 0)$	1.042
$(a_2 = 0) \wedge (a_3 = 0)$	1.067	$(a_1 = 1) \wedge (a_3 = 1)$	0.667

Table 5.5: Subgroup descriptions, and their quality, in the depth 2 beam of the beam search algorithm.

BEAM SEARCH'S CANDIDATES - DEPTH $= 3$			
DESCRIPTION	$\varphi_{\mathcal{D}}(\cdot)$	DESCRIPTION	$\varphi_{\mathcal{D}}(\cdot)$
$(a_1 = 1) \wedge (a_2 = 1) \wedge (a_3 = 1)$	1.225	$(a_1 = 1) \wedge (a_2 = 0) \wedge (a_3 = 0)$	1.067
$(a_1 = 0) \wedge (a_3 = 0)$	1.067	$(a_3 = 0)$	1.042
$(a_2 = 0) \wedge (a_3 = 0)$	1.067	$(a_1 = 1) \wedge (a_3 = 1)$	0.667
$(a_1 = 0) \wedge (a_2 = 1) \wedge (a_3 = 0)$	1.067	$(a_1 = 1) \wedge (a_2 = 0) \wedge (a_3 = 1)$	0.004

Table 5.6: Beam search's depth 3 candidate subgroup descriptions, and their quality.

BEAM SEARCH'S BEAM - DEPTH $= 3$			
DESCRIPTION	$\varphi_{\mathcal{D}}(\cdot)$	DESCRIPTION	$\varphi_{\mathcal{D}}(\cdot)$
$(a_1 = 1) \wedge (a_2 = 1) \wedge (a_3 = 1)$	1.225	$(a_2 = 0) \wedge (a_3 = 0)$	1.067
$(a_1 = 0) \wedge (a_3 = 0)$	1.067	$(a_1 = 0) \wedge (a_2 = 1) \wedge (a_3 = 0)$	1.067

Table 5.7: Subgroup descriptions, and their quality, in the depth 3 beam of the beam search algorithm.

Finally, the algorithm again selects the best ω candidate subgroup descriptions for inclusion in its beam, shown in Table 5.7, after which this set of subgroup descriptions is returned to the user for evaluation. Observe that even on this relatively simple example, this set of returned results already contains redundant descriptions. That is, both $(a_1 = 0) \wedge (a_3 = 0)$ and $(a_1 = 0) \wedge (a_2 = 1) \wedge (a_3 = 0)$ are included in the beam, but inspection of Table 5.1 reveals that these descriptions cover exactly the same records. This is an example of the kind of non-informational 'noise' conditions that the algorithm may add to its results. Although detecting this effect is here fairly trivial due to the small dataset, this is in general not the case when considering real world datasets.

5.5.2 *'Tree-Based Partitioning'*

In this section, we will illustrate how the 'tree-based partitioning' (TBP) algorithm described in Section 5.2 performs on the dataset shown in Table 5.1.

Because, by stipulation, every leaf in the TBP tree conforms to a proper subgroup, we may interpret the root of this tree as the subgroup without a description, viz., as the entire dataset. The algorithm then starts by considering which attribute would yield the best split, using the split

LEAF:	V_0
ATTRIBUTE	$\Delta_\varphi(\cdot)$
a_1	0.1344
a_2	0.1344
a_3	0.6944

Table 5.8: Split qualities of the root vertex V_0 in the TBP algorithm.

quality $\Delta_\varphi(\cdot)$ given by Equation 5.1. We here demonstrate this process with the split performed on attribute a_1.

Clearly, because all attributes are binary, there is only one split which can be performed on a_1, that is, the split yielding children described by $(a_1 = 0)$ and $(a_1 = 1)$. Denote the child vertices corresponding to the partitions containing the records covered by these descriptions with V_1 and V_2, respectively. Reference to Table 5.2 shows that 7 records are assigned to V_1, and 8 records to V_2. Hence, the proportions of records in the root V_0 that are assigned to these children are $\pi(V_1) = \frac{7}{15}$ and $\pi(V_2) = \frac{8}{15}$. Also finding the subgroups' qualities from Table 5.2, we fill in the values in Equation 5.1. With a slight abuse of notation to denote the attribute that the split is performed on, we find

$$\begin{aligned}\Delta_\varphi(V_0, a_1) &= (\pi(V_1)\varphi_{\mathcal{D}}(V_1) + \pi(V_2)\varphi_{\mathcal{D}}(V_2)) - \varphi_{\mathcal{D}}(V_0) \\ &= \left(\frac{7}{15}0.144 + \frac{8}{15}0.126\right) - 0 \\ &= 0.1344.\end{aligned}$$

This process is then repeated for attributes a_2 and a_3. The resulting split qualities are shown in Table 5.8.

Clearly, we see that the highest quality split is obtained by splitting on attribute a_3. Furthermore, this split is legal because it both assigns at least one record to each of the children, and has non-negative quality. Hence, the root V_0 is split into children V_1 and V_2, corresponding to the subgroups described by $(a_3 = 0)$ and $(a_3 = 1)$, respectively.

The algorithm will next evaluate the quality of the splits that can be performed on the newly introduced leafs. Both of these leafs can split on either a_1 or a_2. We demonstrate with leaf V_1 and attribute a_1.

A split of V_1 on a_1 results in two leafs, corresponding to the subgroups described by $(a_1 = 0) \wedge (a_3 = 0)$ and $(a_1 = 1) \wedge (a_3 = 0)$, which we will denote with V_3 and V_4, respectively. Reference to Table 5.2 shows that the subgroup defined by V_1, corresponding to the description $(a_3 = 0)$, contains 5 records. In similar fashion, we find that splitting on a_1 would assign 2 records to V_3, and 3 records to V_4. Thus, the proportions

LEAF:	V_1		V_2	
	ATTRIBUTE	$\Delta_\varphi(\cdot)$	ATTRIBUTE	$\Delta_\varphi(\cdot)$
	a_1	-0.5252	a_1	-0.1665
	a_2	-0.5252	a_2	-0.1665

Table 5.9: Split qualities of leafs V_1 and V_2 in the TBP algorithm.

assigned to these children are $\pi(V_3) = \frac{2}{5}$ and $\pi(V_4) = \frac{3}{5}$. Finding the subgroups' qualities from Table 5.2, we compute

$$\begin{aligned}\Delta_\varphi(V_1, a_1) &= (\pi(V_3)\varphi_{\mathcal{D}}(V_3) + \pi(V_4)\varphi_{\mathcal{D}}(V_4)) - \varphi_{\mathcal{D}}(V_1) \\ &= \left(\frac{2}{5}1.067 + \frac{3}{5}0.225\right) - 1.042 \\ &= -0.5252.\end{aligned}$$

Thus, we see that splitting V_1 on a_1 has a negative split quality, corresponding to the fact that the weighted average of the quality of V_3 and V_4 is lower than the quality of V_1.

The split qualities of all splits that can be performed on V_1 and V_2 are shown in Table 5.9. Here, we see that in fact *all* splits have a negative quality. As such, the algorithm splits neither of the tree's leafs, and returns the subgroups $(a_3 = 0)$ and $(a_3 = 1)$ to the user for evaluation.

Although these final subgroups do not, relatively speaking, have a very low quality, we do here see the effect described in Section 5.2, which is to say, that the initial split turned out to result in a local optimum from which the algorithm could not escape.

5.5.3 *GRADCAT-EMM*

In this section, we will give a detailed demonstration of the GRADCAT-EMM algorithm, using the dataset shown in Table 5.1. We will, however, make some allowances for the fact that we are dealing with a rather small toy problem. Before we can run the algorithm, there are several parameters that we will need to specify.

We will say that only a single record must be assigned to each child for a split in the GRADCAT tree to be considered valid, corresponding roughly to the minimum support of 1 that was stipulated at the beginning of Section 5.5.

We will take the gradient ascent step size η to be 10. Further, we will ignore for simplicity the convergence conditions that one would normally have to specify.

Given the small size of the dataset, we will take the pruning parameter of the post-processing procedure to be $\alpha = 0.10$, corresponding to prun-

ing by testing for the intersection of the 90% confidence ellipsoids of the subgroups' means. Furthermore, we will here use the entire dataset for both the training and post-processing procedure, rather than subsampling the data into separate datasets. This is again done because of the very small dataset involved.

Finally, we will here only demonstrate a single random restart of the algorithm, rather than searching for multiple subgroups.

Given these parameters, the algorithm starts by randomly generating a seed to use for the GRADCAT algorithm.

Seed Generation

To generate a seed for the GRADCAT algorithm using the method detailed in Section 4.4, we begin by computing for each record the Cook's Distance obtained when that record is removed from the dataset. We here simply use the straightforward method of computation, as opposed to the method described in Section 4.4, although the latter is much more efficient for real datasets.

On the dataset under consideration, when we construct a subgroup $G = \mathcal{D} \setminus r_i$ by excluding the i-th record from the dataset, we may write Cook's Distance as

$$D_{\mathcal{D}\setminus r_i} = n \cdot \frac{\left(\bar{x}_{\mathcal{D}\setminus r_i} - \bar{x}\right)^2}{s^2}.$$

Some consideration will reveal that, on the specific example used, this measure can only take two different values. That is, this quantity depends only on i through $\bar{x}_{\mathcal{D}\setminus r_i}$, and this subgroup mean is identically influenced by all records with the same value of x.

Computation reveals that $D_{\mathcal{D}\setminus r_i} \approx 0.0816$ when $x^i = 0$, and $D_{\mathcal{D}\setminus r_i} = 0.0625$ when $x^i = 1$. Recall that we will next randomly determine whether or not to 'include' the i-th record in the seed, using

$$\Pr(r_i \text{ is included}) = 1 - \frac{D_{\mathcal{D}\setminus r_i}}{\max_j \left\{ D_{\mathcal{D}\setminus r_j} \right\}}.$$

Because the individual Cook's Distances can here only take two possible values, we immediately see that records with $x = 0$ can never be included in this sense. We also find $\Pr(r_i \text{ is included} \mid x^i = 1) \approx 0.234$.

Reference to Table 5.1 shows that there are 8 records with $x = 1$. Using some appropriate method to determine these records' inclusion, we find that two records are selected in this fashion, say records r_1 and r_7. In fact, it turns out that on this dataset it does not really matter how many records are selected, so long as this number is non-zero. This pathological case will cause the objective function's derivative to be zero at the

starting position, which in turn means that the algorithm will not return a subgroup for the restart that used this seed.

We now have the required elements to construct the seed, which we will denote as a 15×1 vector $\mathbf{s}$. If the i-th record was not selected for inclusion, we have $s_i = \frac{1}{2}$. For the included records, we find

$$\begin{aligned} s_1 = s_7 &= \sigma\left(D_{\mathcal{D}\setminus r_7}\right) \\ &= \sigma(0.0625) \\ &\approx 0.516. \end{aligned}$$

This seed $\mathbf{s}$ is subsequently passed to the GRADCAT algorithm, which we will discuss in the following section.

GRADCAT

Recall that, in the GRADCAT model, the model's outputs represent the records' degree of inclusion in the subgroup being optimized. On the dataset that we are considering here, where we only have a single target variable, this means that the subgroup mean $\bar{x}_G$ can simply be written as the weighted average of all the records' target values,

$$\bar{x}_G = \frac{\sum_{i=i}^{15} w_i x^i}{\sum_{i=1}^{15} w_i}.$$

Here, w_i represents the inclusion weight of the i-th record.

On the initial iteration of the GRADCAT algorithm, the previously constructed seed $\mathbf{s}$ is set to be the model's output. Recall from the previous section that $s_1 = s_7 \approx 0.516$, and $s_i = \frac{1}{2}$ for all other records. After reference to Table 5.1 to find the corresponding target values, we find that the subgroup mean $\bar{x}_G^{(0)}$ at the zeroth iteration is

$$\begin{aligned} \bar{x}_G^{(0)} &= \frac{\sum_{i=i}^{15} s_i x^i}{\sum_{i=1}^{15} s_i} \\ &\approx \frac{4.031}{7.531} \\ &\approx 0.5352. \end{aligned}$$

The algorithm next computes the partial derivatives of the objective function $O(\cdot)$, given by Equation 4.3, with respect to the inclusion weights. Leaving the proof of the derivation as an exercise for the reader, we find that because we are here dealing with a single target variable, the expression for the i-th partial derivative may be written as

$$\frac{\partial O(\cdot)}{\partial w_i} = \frac{(\bar{x}_G - \bar{x})^2}{2s^2} - \frac{(\bar{x}_G - \bar{x})(\bar{x}_G - x^i)}{s^2}.$$

We see that this expression depends only on i through the term x^i, which again implies that the derivative can only take two different values, for respectively the records with $x = 0$ and $x = 1$. Computation reveals that for records with $x = 0$, we find at iteration $\tau = 0$

$$\begin{aligned}\left.\frac{\partial O(\cdot)}{\partial w_i}\right|_{\tau=0,x^i=0} &= \frac{\left(\bar{x}_G^{(0)} - \bar{x}\right)^2}{2s^2} - \frac{\left(\bar{x}_G^{(0)} - \bar{x}\right)\left(\bar{x}_G^{(0)} - 0\right)}{s^2} \\ &\approx \frac{(0.5352 - 0.5333)^2}{0.5333} - \frac{(0.5352 - 0.5333)(0.5352)}{0.2667} \\ &\approx -0.0039.\end{aligned}$$

Similarly, we compute for the remaining records $\left.\frac{\partial O(\cdot)}{\partial w_i}\right|_{\tau=0,x^i=1} \approx 0.0034$.

Having obtained these partial derivatives, the algorithm will compute the impurity of all the leafs in the tree, which at this point is only the root. Let $R^+_{V_0,(0)}$, $R^-_{V_0,(0)}$, and $R^0_{V_0,(0)}$ denote the sets of indices of records assigned to the root V_0, which at time $\tau = 0$ have positive, negative, and zero-valued partial derivatives, respectively. From our earlier results, we have $|R^+_{V_0,(0)}| = 8$, $|R^-_{V_0,(0)}| = 7$, and $|R^0_{V_0,(0)}| = 0$.

The impurity $\phi(V_0, (1))$ of the root V_0 at the first iteration $\tau = 1$ is given by the multi-class Gini-index, which we compute to be

$$\begin{aligned}\phi(V_0, (1)) &= \frac{|R^+_{V_0,(0)}|}{|R_{V_0}|}\left(1 - \frac{|R^+_{V_0,(0)}|}{|R_{V_0}|}\right) \\ &\quad + \frac{|R^-_{V_0,(0)}|}{|R_{V_0}|}\left(1 - \frac{|R^-_{V_0,(0)}|}{|R_{V_0}|}\right) \\ &\quad + \frac{|R^0_{V_0,(0)}|}{|R_{V_0}|}\left(1 - \frac{|R^0_{V_0,(0)}|}{|R_{V_0}|}\right) \\ &= \frac{8}{15}\left(1 - \frac{8}{15}\right) + \frac{7}{15}\left(1 - \frac{7}{15}\right) \\ &\approx 0.498.\end{aligned}$$

The algorithm will next consider the split quality that is obtained when the root is split on any of the attributes $a_1, \ldots, a_3$. We demonstrate with attribute a_1.

Splitting on a_1 will yield two child vertices, which we will denote with V_1 and V_2, corresponding to the sets of records (but *not* the subgroups, in the EMM sense) described by $(a_1 = 0)$ and $(a_1 = 1)$, respectively. Referring to Table 5.2, we find that the proportions of records assigned to these children are $\pi(V_1) = \frac{7}{15}$ and $\pi(V_2) = \frac{8}{15}$. Furthermore, from

LEAF:	V_0
ATTRIBUTE	$\Delta_\phi(\cdot)$
a_1	0.0192
a_2	0.0192
a_3	0.1111

Table 5.10: Split qualities of the root vertex V_0 in the GRADCAT algorithm.

the observation that the sign of the derivative for the i-th record here depends only on the value of x^i, we compute the impurity of V_1 to be

$$\begin{aligned}\phi(V_1,(1)) &= \frac{3}{7}\left(1-\frac{3}{7}\right)+\frac{4}{7}\left(1-\frac{4}{7}\right)\\ &\approx 0.490.\end{aligned}$$

In similar fashion, we compute $\phi(V_2,(1)) \approx 0.469$. Filling in the appropriate values, we find that the quality of splitting V_0 at time $\tau = 1$ on attribute a_1 is

$$\begin{aligned}\Delta_\phi(V_0,a_1,(1)) &= \phi(V_0)-(\pi(V_1)\phi(V_1,(1))+\pi(V_2)\phi(V_2,(1)))\\ &\approx 0.498-\left(\frac{7}{15}0.490+\frac{8}{15}0.469\right)\\ &\approx 0.0192.\end{aligned}$$

This process is repeated to compute the quality of splitting on a_2 or a_3. The split qualities are shown in Table 5.10.

We observe that splitting on attribute a_3 yields the best split quality, and hence this split is performed to obtain two children V_1 and V_2, which are assigned the records described by $(a_3 = 0)$ and $(a_3 = 1)$, respectively. The output values v^1 and v^2 of these children are now initialized to the output value v^0 of their parent, which so far amounts to setting $v^1 = v^2 = 0$.

Gradient ascent is subsequently performed to update these output values. We first compute the partial derivative of the objective function with respect to the j-th output value v^j, which is given by

$$\begin{aligned}\frac{\partial O(\cdot)}{\partial v^j} &= \frac{\partial}{\partial v^j}\left[\sigma(v^j)\right]\cdot\sum_{r_i\in R_{V_j}}\frac{\partial O(\cdot)}{\partial w_i}\\ &= \sigma(v^j)(1-\sigma(v^j))\cdot\sum_{r_i\in R_{V_j}}\frac{\partial O(\cdot)}{\partial w_i}.\end{aligned}$$

ID	$w_i^{(1)}$	$\frac{\partial O(\cdot)}{\partial w_i}\Big\vert_{\tau=1}$	ID	$w_i^{(1)}$	$\frac{\partial O(\cdot)}{\partial w_i}\Big\vert_{\tau=1}$	ID	$w_i^{(1)}$	$\frac{\partial O(\cdot)}{\partial w_i}\Big\vert_{\tau=1}$
1	0.508	0.0034	6	0.508	0.0034	11	0.508	0.0034
2	0.508	0.0034	7	0.508	0.0034	12	0.508	−0.0039
3	0.508	0.0034	8	0.508	−0.0039	13	0.508	0.0034
4	0.492	−0.0039	9	0.492	−0.0039	14	0.508	−0.0039
5	0.492	−0.0039	10	0.492	−0.0039	15	0.492	0.0034

Table 5.11: Inclusion weights and derivatives of all records at time $\tau = 1$.

Filling in the appropriate values, we demonstrate for the output value $v^1_{(0)}$ of V_1 at time $\tau = 0$,

$$\begin{aligned}
\frac{\partial O(\cdot)}{\partial v^1}\Big|_{\tau=0} &= \sigma\left(v^1_{(0)}\right)\left(1-\sigma\left(v^1_{(0)}\right)\right)\cdot\sum_{r_i\in R_{V_1}}\frac{\partial O(\cdot)}{\partial w_i}\Big|_{\tau=0} \\
&= \sigma(0)(1-\sigma(0))\cdot\left(4\,\frac{\partial O(\cdot)}{\partial w_i}\Big|_{\tau=0,x^i=0} + 1\,\frac{\partial O(\cdot)}{\partial w_i}\Big|_{\tau=0,x^i=1}\right) \\
&= \frac{1}{2}\left(1-\frac{1}{2}\right)\cdot(4\cdot -0.0039 + 0.0034) \\
&\approx -0.00305.
\end{aligned}$$

The output value v^1 is then updated by gradient ascent to the value $v^1_{(1)}$ at the current time step $\tau = 1$, using

$$\begin{aligned}
v^1_{(1)} &= v^1_{(0)} + \eta\cdot\frac{\partial O(\cdot)}{\partial v^1}\Big|_{\tau=0} \\
&\approx 0 + 10\cdot -0.00305 \\
&= -0.0305.
\end{aligned}$$

Similarly, this process is repeated for the output value v^2 of vertex V_2.

Finally, the iteration is completed by computing the new inclusion weights, using

$$w_i^{(1)} = \begin{cases} \sigma\left(v^1_{(1)}\right) & \text{if } i\in R_{V_1}, \text{ and} \\ \sigma\left(v^2_{(1)}\right) & \text{if } i\in R_{V_2}, \end{cases}$$

where $w_i^{(1)}$ denotes the inclusion weight of the i-th record at time $\tau = 1$. The updated inclusion weights are shown for all records in Table 5.11.

On the next time step, $\tau = 2$, the algorithm will begin by computing the partial derivatives obtained from the outputs of the *previous* time step. The values of these derivatives are shown in Table 5.11. The algorithm will then evaluate the quality of splitting the leafs V_1 and V_2. We will briefly demonstrate for leaf V_1 and attribute a_1.

LEAF:	V_1		V_2	
	ATTRIBUTE	$\Delta_\phi(\cdot)$	ATTRIBUTE	$\Delta_\phi(\cdot)$
	a_1	0.0533	a_1	0.02
	a_2	0.0533	a_2	0.02

Table 5.12: Split qualities of leafs V_1 and V_2 in the GRADCAT algorithm.

From Table 5.1, we find that leaf V_1, corresponding to the description $(a_3 = 0)$, contains the records r_4, r_5, r_9, r_{10}, and r_{15}. Looking up the signs of these records' derivatives, we see that there are 4 records with a negative derivative, and one with a positive derivative. We thus compute the impurity of V_1 at time $\tau = 2$ to be

$$\begin{aligned}\phi(V_1,(2)) &= \frac{1}{5}\left(1-\frac{1}{5}\right)+\frac{4}{5}\left(1-\frac{1}{5}\right)\\ &= 0.32.\end{aligned}$$

We evaluate a split on a_1, which constructs two children V_3 and V_4, corresponding to the records described by $(a_1 = 0) \wedge (a_3 = 0)$ and $(a_1 = 1) \wedge (a_3 = 0)$, respectively. Reference to Table 5.1 shows that this split assigns records r_9 and r_{10} to V_3, and records r_4, r_5, and r_{15} to V_4. From the signs of these records' derivatives, we thus compute that $\phi(V_3,(2)) = 0$, and $\phi(V_4,(2)) = 0.44$. Also filling in the appropriate values for the proportion of records that is assigned to each child, we compute

$$\begin{aligned}\Delta_\phi(V_1,a_1,(2)) &= \phi(V_1,(2)) - (\pi(V_3)\phi(V_3,(2)) + \pi(V_4)\phi(V_4,(2)))\\ &= 0.32 - \left(\frac{2}{5}0+\frac{3}{5}0.44\right)\\ &= 0.0533.\end{aligned}$$

The quality of all splits that the algorithm considers at time $\tau = 2$ are shown in Table 5.12.

We see that neither of the leafs has a preference for splitting on either a_1 or a_2, and that splitting on either of these attributes is valid for both leafs. We will stipulate that in the case of tied split qualities, a leaf will be split on the attribute with the lowest index from among these tied attributes. That is, we will assume here that both V_1 and V_2 perform a split on attribute a_1.

The newly introduced vertices' outputs are then updated using gradient ascent in the familiar fashion. New values for the inclusion weights are subsequently computed for all records, which are shown in Table 5.13. The second iteration is then finished.

Skipping ahead to what we will interpret as convergence, we check back with the algorithm at time $\tau = 20$. The resulting tree is shown

ID	$w_i^{(2)}$	ID	$w_i^{(2)}$	ID	$w_i^{(2)}$
1	0.518	6	0.510	11	0.518
2	0.518	7	0.510	12	0.518
3	0.518	8	0.510	13	0.510
4	0.488	9	0.484	14	0.510
5	0.488	10	0.484	15	0.488

Table 5.13: Inclusion weights of all records at time $\tau = 2$.

ID	$w_i^{(20)}$	ID	$w_i^{(20)}$	ID	$w_i^{(20)}$
1	0.994	6	0.040	11	0.012
2	0.994	7	0.040	12	0.012
3	0.994	8	0.040	13	0.012
4	0.003	9	0.003	14	0.012
5	0.003	10	0.003	15	0.981

Table 5.14: Inclusion weights of all records at time $\tau = 20$.

in Figure 5.1, and the inclusion weights are given by Table 5.14. The resulting tree is now passed to the post-processing procedure, which we will describe next.

Post-Processing

The first step in post-processing the GRADCAT tree is to convert the tree structure into a disjunctive set of 'hard' inclusion rules. To do this, we consider the leafs V_j for which $\sigma(v^j) \geq \frac{1}{2}$, that is, the leafs assigning inclusion weights that are not lower than $\frac{1}{2}$.

From Figure 5.1, we see that this here corresponds to leafs V_8 and V_{12}. For each of these leafs, the path from the root of the tree to that leaf specifies a conjunctive inclusion rule λ, and the set of these conjunctive rules forms the disjunctive rule set which we will denote as Λ. Reading these paths from the tree, we will write

$$\begin{aligned}
\lambda_1 &= (a_1 = 1) \wedge (a_2 = 1) \wedge (a_3 = 0) \quad \text{for } V_8, \\
\lambda_2 &= (a_1 = 1) \wedge (a_2 = 1) \wedge (a_3 = 1) \quad \text{for } V_{12}, \text{ and} \\
\Lambda &= \{\lambda_1, \lambda_2\}.
\end{aligned}$$

Because Λ forms a disjunctive rule set, it is easy to see that, logically,

$$\begin{aligned}
\Lambda &\Leftrightarrow \Big((a_3 = 0) \vee (a_3 = 1)\Big) \wedge (a_1 = 1) \wedge (a_2 = 1) \\
&\Leftrightarrow (a_1 = 1) \wedge (a_2 = 1).
\end{aligned}$$

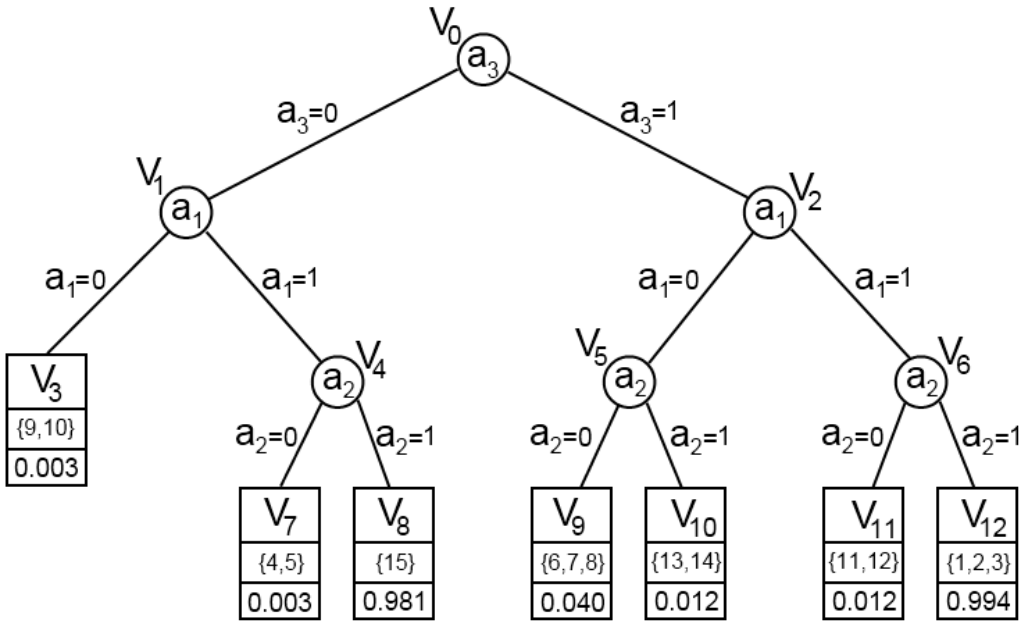

Figure 5.1: The GRADCAT tree grown on the example data, after 20 iterations. Branches denote the value of the split-attribute that records must satisfy to be passed down that branch. The leafs are shown as partitioned rectangles, showing from top to bottom the vertex's identifier, the indices of the records assigned to this leaf, and the inclusion weights of the corresponding records.

This is to say, a subgroup described by Λ is equivalent to a subgroup described by $(a_1 = 1) \wedge (a_2 = 1)$. Hence, reference to Table 5.2 shows that we have managed to find the global optimum of this search space, although the *description* that we currently have is still somewhat redundant. We will now show how the remainder of the post-processing procedure takes care of this.

The first phase of the actual post-processing is the 'Quality Maximization' phase. Here, we begin by trying to remove entire rules from Λ. A rule will be removed when we can do so without lowering the described subgroup's quality. If multiple rules can be removed, the one yielding the highest quality is chosen for removal. With some abuse of notation, we can find from Table 5.2,

$$\varphi_{\mathcal{D}}(\Lambda) = 1.633,$$

$$\begin{aligned}\varphi_{\mathcal{D}}(\Lambda \setminus \lambda_1) &= \varphi_{\mathcal{D}}(\lambda_2) \\ &= 1.225,\end{aligned}$$

$$\begin{aligned}\varphi_{\mathcal{D}}(\Lambda \setminus \lambda_2) &= \varphi_{\mathcal{D}}(\lambda_1) \\ &= 0.408.\end{aligned}$$

Λ'	$\varphi_{\mathcal{D}}(\cdot)$	Λ'	$\varphi_{\mathcal{D}}(\cdot)$	Λ'	$\varphi_{\mathcal{D}}(\cdot)$
$\Lambda \setminus a_1^{\lambda_1}$	0.200	$\Lambda \setminus a_2^{\lambda_1}$	0.200	$\Lambda \setminus a_3^{\lambda_1}$	1.633
$\Lambda \setminus a_1^{\lambda_2}$	1.013	$\Lambda \setminus a_2^{\lambda_2}$	1.013	$\Lambda \setminus a_3^{\lambda_2}$	1.633

Table 5.15: Qualities of rule set Λ with conditions removed.

We thus see that the quality cannot be improved by removing entire rules from Λ. The algorithm next tries to remove single conditions from the individual rules.

Say that we try to remove the condition on attribute a_1 from λ_1. We will denote the rule set with this condition removed as $\Lambda' = \Lambda \setminus a_1^{\lambda_1}$. We compute the resulting qualities using the data from Table 5.1, which we show in Table 5.15. The conditions in the rules are then ranked in order of decreasing quality of their removal. We obtain the ordering $a_3^{\lambda_1}, a_3^{\lambda_2}, a_1^{\lambda_2}, a_2^{\lambda_2}, a_1^{\lambda_1}, a_2^{\lambda_1}$.

We now consider each of these conditions in order, and remove them from the rule set so long as this does not decrease the quality. If a condition is removed, the removal-quality will be re-computed for subsequent attributes, but without changing the ordering.

We first consider removing $a_3^{\lambda_1}$. We see that this does not change the quality of the described subgroup. Of course, this is not surprising because $\lambda_1 \setminus a_3 \Leftrightarrow \Lambda$. Because the quality is not lowered, the attribute is removed. In this case, we need not re-compute the other removal-qualities, because the subgroup description is logically unchanged.

Similarly, we find that we can remove $a_3^{\lambda_2}$ without lowering the quality. Again, the logical subgroup description is unchanged, so there is no need to recompute anything. When we consider the remaining attributes for removal, we see that we cannot do so without obtaining a lower quality subgroup. Therefore, none of the other conditions are removed.

Because Λ now contains rules $\lambda_1 = \lambda_2$, either one is removed from the set. We will for brevity in the sequel simply write the remaining rule as $\lambda = (a_1 = 1) \wedge (a_2 = 1)$.

The post-processing procedure now moves on to the 'Description Simplification' phase. Here, we try to remove superfluous conditions from the remaining rules. In the current example, there are clearly only two conditions that we need to consider for removal.

The subgroups resulting by removing a_1^{λ} or a_2^{λ} are described by, respectively, $(a_2 = 1)$ and $(a_1 = 1)$. Normally, we would now compute Cook's Distance between the subgroup described by λ, and the subgroups described by $(a_2 = 1)$ and $(a_1 = 1)$. We would then order a_1^{λ} and a_2^{λ}, in order of increasing Cook's Distance resulting from their removal. The intuition here is that we first try to remove the conditions

whose removal only results in a small difference between the described subgroups' means. However, inspection of Table 5.1 reveals that these subgroups have the same size and subgroup mean, which must result in an equivalent Cook's Distance, so we will simply stipulate the ordering a_1^λ, a_2^λ.

The algorithm now again considers these condition in order. Thus, we start by considering a_1^λ. This condition is removed from λ if and only if the $(1-\alpha)\times 100\%$ confidence ellipsoids of $\bar{x}_\lambda$ and $\bar{x}_{\lambda\setminus a_1^\lambda}$ intersect. Here, $\bar{x}_\lambda$ and $\bar{x}_{\lambda\setminus a_1^\lambda}$ denote the means of x in the subgroups described by the rule in their subscript.

Because we are here dealing with only a single target variable, the confidence ellipsoid of $\bar{x}_\lambda$ reduces to a simple confidence interval given by all values x^* satisfying

$$|G_\lambda| \cdot \frac{(x^* - \bar{x}_\lambda)^2}{s_\lambda^2} \leq F(1, |G_\lambda| - 1, 1-\alpha).$$

Here, $|G_\lambda|$ and s_λ^2 denote the size of, and the variance of the target variable x within, the subgroup described by λ, respectively. The quantity $F(1, |G_\lambda| - 1, 1-\alpha)$ denotes the $1-\alpha$ probability point of the central F distribution with 1 and $|G_\lambda| - 1$ degrees of freedom.

By our earlier stipulation, we here use $\alpha = 0.10$. Inspection of Table 5.1 shows that the subgroup described by λ has $\bar{x}_\lambda = 1$, and hence $s_\lambda^2 = 0$. As such, this is a somewhat pathological case, in that we cannot divide by the variance. However, multiplying by s_λ^2 on both sides, we simply find that the 90% confidence interval of $\bar{x}_\lambda$ is given by

$$\bar{x}_\lambda \pm 0.$$

For the subgroup described by $\lambda \setminus a_1^\lambda$, that is, the subgroup described by $(a_2 = 1)$, we find from Table 5.1 that $\bar{x}_{\lambda\setminus a_1^\lambda} = \frac{5}{8}$ and $s^2_{\lambda\setminus a_1^\lambda} \approx 0.268$. Further, we find $F(1, 8-1, 0.90) \approx 3.589$. Taking x^* to be at the boundary of the confidence interval by setting the inequality above to a strict equality, filling in the appropriate values, and solving for the difference $(x^* - \bar{x}_{\lambda\setminus a_1^\lambda})$, we find that the distance of $\bar{x}_{\lambda\setminus a_1^\lambda}$ to the boundary of its 90% confidence interval is given by

$$\begin{aligned}
8 \cdot \frac{\left(x^* - \bar{x}_{\lambda\setminus a_1^\lambda}\right)^2}{s^2_{\lambda\setminus a_1^\lambda}} &= F(1, 8-1, 0.90) \\
\left(x^* - \bar{x}_{\lambda\setminus a_1^\lambda}\right)^2 &= \frac{s^2_{\lambda\setminus a_1^\lambda} \cdot F(1, 8-1, 0.90)}{8} \\
x^* - \bar{x}_{\lambda\setminus a_1^\lambda} &= \pm\sqrt{\frac{s^2_{\lambda\setminus a_1^\lambda} \cdot F(1, 8-1, 0.90)}{8}}.
\end{aligned}$$

After some computation we see that the corresponding confidence interval is

$$\bar{x}_{\lambda \setminus a_1^\lambda} \pm 0.3467.$$

We see that the confidence intervals do not intersect, because

$$\begin{aligned} \bar{x}_{\lambda \setminus a_1^\lambda} + 0.3467 &< \bar{x}_\lambda - 0 \\ \frac{5}{8} + 0.3467 &< 1 - 0 \\ 0.972 &< 1. \end{aligned}$$

Therefore, the condition a_1^λ is not removed from λ. As we mentioned earlier, the subgroups described by $(a_1 = 1)$ and $(a_2 = 1)$ are identical, so repeating the above will yield the same result for the condition a_2^λ. Hence, the second post-processing phase is now completed.

In the last, 'Interestingness Assertion', phase, we will compare the subgroup described by λ with the 'global' population. This is again done by testing for intersection of the confidence ellipsoids. Some computation shows that the 90% confidence interval for the global mean $\bar{x}$ is given by

$$\bar{x} \pm 0.2348.$$

Testing against the subgroup reveals that

$$\begin{aligned} \bar{x} + 0.2348 &< \bar{x}_\lambda - 0 \\ \frac{8}{15} + 0.2348 &< 1 - 0 \\ 0.7682 &< 1, \end{aligned}$$

and hence that the confidence intervals do not overlap. As such, the subgroup described by λ is accepted. Had the confidence intervals overlapped, the result would have been discarded, and a new random restart would be performed.

This run of the GRADCAT-EMM algorithm is now completed in full, and more subgroups may be retrieved in similar fashion. This set of subgroups is then returned to the user for evaluation.

6

EXPERIMENTS

This chapter discusses the experiments that we have performed for evaluating the GRADCAT-EMM algorithm discussed in Chapter 4. Here, we quantitatively compare the GRADCAT-EMM algorithm with beam search (BS). We also compare both of these algorithms with a composite algorithm in which the post-processing procedure described in Section 4.5 is applied to the results returned by beam search. We will refer to this composite algorithm simply as 'beam search with post-processing', or alternatively as 'Beam Search with Pruning' (BSP) for brevity.

Essentially, the experiments may be divided into two parts; the first part pertains to experiments with synthetic data, whereas the second part considers experiments on real world datasets. Sections 6.1 and 6.2 discuss these different sets of experiments, respectively. We finally close in Section 6.3 with an exposition of some illustrative results that were obtained by GRADCAT-EMM on the real world datasets.

6.1 SYNTHETIC DATA

As mentioned, the first part of the experiments considers results on synthetic data. We consider this useful because of the inherent unsupervised nature of the EMM problem; that is, how do we know whether a given algorithm has succeeded in retrieving the interesting subgroups from a given dataset, if we do not ourselves know the 'correct' answers?

By performing experiments on synthetic data, we have, to a large extent, control over how many 'interesting' subgroups are in the data, and how complex the descriptions of these subgroups are. Furthermore, it allows us to generate as many of these datasets as we like, allowing for large sample sizes. Finally, it gives us control over a variety of parameters, enabling us to test the algorithms on datasets with several distinct characteristics.

6.1.1 *Data Generation*

The synthetic datasets were generated as follows. First, several linear regression models were formulated over a set of explanatory variables, with one of these models representing the 'global' population, and the others representing the models of subgroups. The subgroup models were formulated such that the slope of their regression line was perpendicular to the slope of the 'global' model. Furthermore, all explanatory

variables were drawn from a normal distribution, although each model's explanatory variables could have different parameters for their distribution.

A distribution of descriptive attributes was constructed and coupled to the set of regression models. For each subgroup model, a number of these attributes were chosen as that model's descriptive attributes, such that for all subgroups, these sets of descriptive attributes were disjoint. All descriptive attributes were distributed uniformly and independently, except for the descriptive attributes of the subgroup models. For the latter, the attributes were constructed such that they could be sampled in either of two ways. First, if sampled for the subgroup, they followed a distribution conforming to that subgroup's description; if not sampled for the subgroup, they were distributed uniformly, but not independently in the sense that we ensured that they could never satisfy the subgroup's description.

Taken together, this gives us a joint distribution from which we can sample entire records for any of the formulated models, be it the 'global' model, or any of the subgroups. An example of the regression data sampled from such a distribution is shown in Figure 6.1.

6.1.2 *Experimental Setup*

Synthetic datasets were generated with a number of different parameter settings using the method outlined in Section 6.1.1. These collections of datasets differed in the number of subgroups, the number of explanatory variables of the regression models, and the type and number of the descriptive attributes. This gives three collections of experiments, for which in each, one of these characteristics was varied, and the others kept constant. Furthermore, for each of these parameter settings, we also varied the standard deviation of the errors of the 'global' regression model, and the number of descriptive attributes used for each of the subgroup models. Finally, the means of the models' explanatory variables' distributions were determined stochastically for each constructed dataset-distribution. The different settings of the parameters used are summarized in Table 6.1.

For each parameter setting, we constructed 50 dataset-distributions as previously outlined, and a training and validation dataset was sampled from each of these distributions. These datasets all contained 3000 and 1500 records, respectively. For the training datasets, we sampled 250 records from each of the subgroup models, and the remainder was sampled from the global model. Likewise, for the evaluation data, we sampled 125 records from each of the subgroup models, and completed the datasets by sampling from the global model.

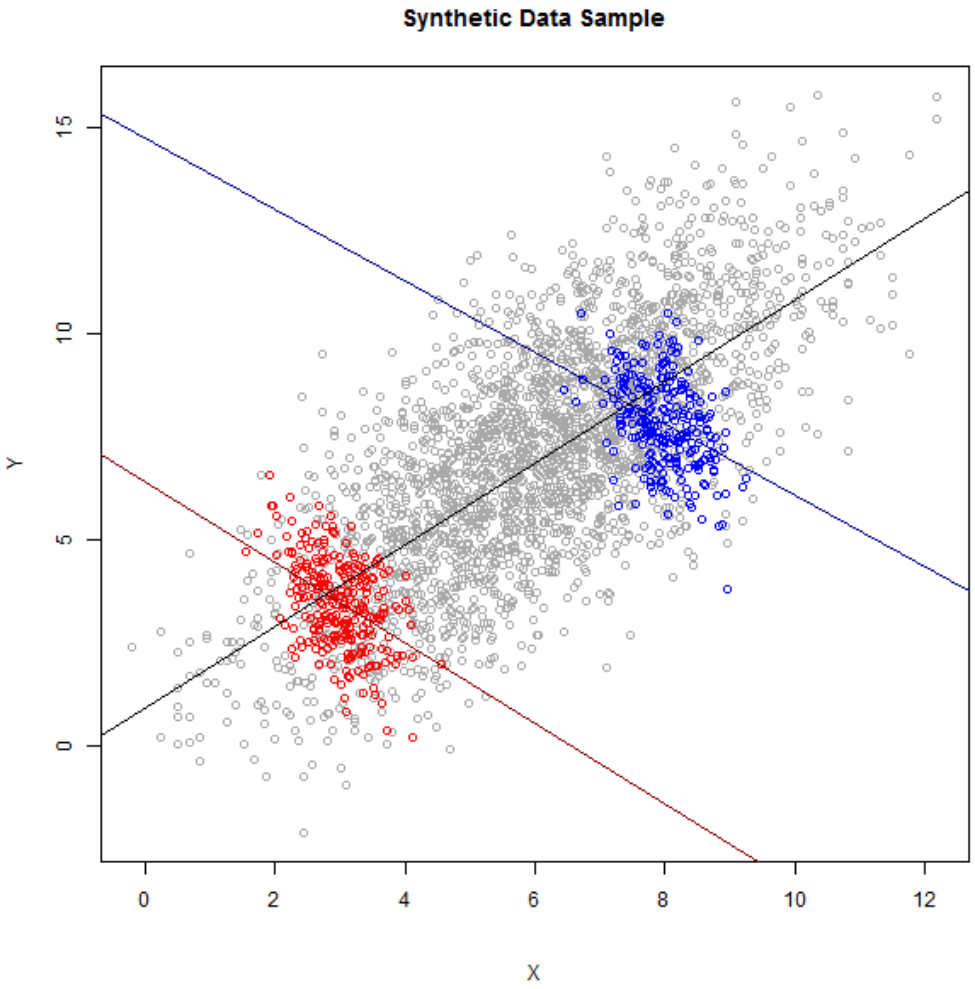

Figure 6.1: An example of the regression data sampled from a synthetic distribution. Data sampled from the global 'noise' model is shown in grey, whereas data points from the two 'correct' subgroups are shown in red and blue. Also shown are the regression lines fitted to each of the subgroups, as well as the model induced on the entire dataset.

PARAMETER	EXP 1	EXP 2	EXP 3
Global Error Stdev.	0.5, 2	0.5, 2	0.5, 2
Num Attributes	64	64	32
Num Descriptive	4, 5	4, 5	2, 4
Num Subgroups	2, 4, 6	4	4
Num Explanatory	1	2, 3, 4	2
Attribute Type	Binary	Binary	Numeric
Total Settings	12	12	4

Table 6.1: Overview of the different parameter settings used in each of the three sets of experiments. The first column shows the parameter being varied, and the other columns show the settings of that parameter for the first, second, and third set of experiments.

On each dataset, we evaluated the performance of the various algorithms under a couple of different parameters settings of each algorithm. These parameter settings, as well as the parameters that were not varied, were determined from some preliminary testing on related, but different, datasets. These settings of the algorithms are shown in Table 6.2.

We remark at this point that, although the post-processing procedure described in Section 4.5 is readily applied to the subgroup descriptions returned by beam search, there is one issue that needs to be resolved. Specifically, the post-processing procedure, as used by GRADCAT-EMM, is performed on a separate held-out testing dataset. Furthermore, this data is randomly sub-sampled for each restart of the algorithm. Clearly, beam search, as described, does not do this, so the question arises on which dataset the BSP algorithm should perform its pruning. Here, we have chosen to simply perform the post-processing on the complete set of training data.

By doing this, the results will be obtained on the same data that was used by the (normal) beam search algorithm, which we argue allows for better comparison between these two algorithms. There are at least two other approaches that could have been used, but neither seems very appealing. In particular, one option would be to sub-sample the data once for the BSP algorithm, and then perform training and post-processing separately on the two resulting datasets. This has the disadvantage that the results may be influenced by idiosyncrasies in the particular sub-sampling used, which we consider undesirable. A second alternative would be to give the BSP algorithm some additional data to use for post-processing, but this would also seem to lead to an unfair comparison between the algorithms. As such, simply performing the post-processing

GRADCAT-EMM	
Results Generated	50
Min Records in Leaf for Split	50
Min Records in Child for Split	10
Min iterations	10
Max iterations	350
Gradient Ascent Step Size	0.1, 10
Gradient Convergence Threshold	0.1
Pruning α	0.01
Max Runtime per Dataset (Minutes)	10
BEAM SEARCH	
Search Depth	Num Descriptive
Minimum Support	10
Beam Width	50
Num Numeric Split Bins	20
BEAM SEARCH WITH POST-PROCESSING	
Search Depth	6, 7, 8 or 6[1]
Minimum Support	10
Beam Width	50
Num Numeric Split Bins	20
Pruning α	0.01

Table 6.2: Parameter settings of the different algorithms.
1: For the experiments with numeric descriptive attributes, a single search depth of 6 was used.

on the training data seems to both be the fairest, and most straightforward approach to use.

6.1.3 *Performance Measures*

Performance of the algorithms was measured in two ways. The primary measure makes use of the fact that, due to the synthetic construction of the data, we can evaluate the problem as being supervised, that is, we can make use of the fact that we know the 'correct' answers. The second measure, on the other hand, only makes use of the quality $\varphi_{\mathcal{D}}(\cdot)$ of the algorithms' returned results. Thus, we may regard this second measure

as being an unsupervised evaluation method. The following two sections discuss these evaluation measures in detail.

Supervised Evaluation

Our method for evaluating the algorithms' performance in a supervised fashion works in the following way. For each algorithm, we looked at the set of results returned, and compared those descriptions against the descriptions that were used to construct the subgroups in the dataset. If a returned description matched any of these subgroup descriptions, we considered that result 'correct', and otherwise we considered it 'incorrect', regardless of the actual quality of that result in terms of the quality measure $\varphi_{\mathcal{D}}(\cdot)$. Only for the experiments where numeric descriptive attributes were used, were we slightly less precise in what we counted as 'correct' answers. Here, the actual subgroup descriptions were formulated using pairs of lower and upper bounds on the same attribute, e.g., they might be defined as $(a_3 > 0.7) \wedge (a_3 \leq 0.9)$, where 0.7 and 0.9 constitute the lower and upper bound, respectively. To evaluate whether a result correctly identified such a subgroup, we counted the returned threshold values as being correct so long as they were not lower or higher than the actual lower or upper bounds, respectively. Thus, for example, we would also consider a result such as $(a_3 > 0.75) \wedge (a_3 \leq 0.85)$ to be correct. This was mostly done to offset the different ways the algorithms handle the splitting on numeric attributes, so that we do not penalize one method more than the other.

By only counting exact subgroup descriptions as correct results, we were able to quantify the performance of the algorithms in a number of different ways. Firstly, we could look at the algorithms' *precision*, i.e., the fraction of correct results in the entire set of returned results. Secondly, we measured the algorithms' *recall* on each dataset, that is, the fraction of correct subgroups found, out of the total number of subgroups that were used in the generation of the data.

These measures both quantify different aspects of an algorithm that may be considered important in the field of data mining; a high recall but low precision means that most of the 'correct' answers are returned, but they may be buried in a set of 'uninteresting' results. Clearly, this would be inconvenient for an end-user who wishes to use these results to analyze the data. On the other hand, a high precision but low recall indicates that although most of the returned results are 'correct', only a small number of the 'correct' results have been found. Thus, the user would miss interesting results, which is also obviously undesirable.

The F_1 measure, or more generally the F_β measure, is a scoring function that captures both of these aspects in a single quantity. Essentially, it is the weighted harmonic mean of the precision and the recall, weighted such that it reflects the proper trade-off for a user that considers the re-

call to be β times more important than the precision [31]. Formally, the F_β measure is given by

$$F_\beta(\text{precision}, \text{recall}) = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}},$$

for $\beta \in \mathbb{R}$ and $\beta > 0$. For our purposes, we only consider the F_1 measure, and use it to summarize each algorithm's performance on the various datasets.

Furthermore, we recorded the percentage of datasets on which the algorithms were able to find at least one of the 'correct' subgroups. Broadly, this may give us some idea of how often an algorithm may fail completely to find any interesting results on a given dataset.

We summarize the relative performance of the various algorithms on this performance measure by performing a Friedman test [9] over all datasets generated in a single experiment, using the algorithms' F_1 score as the statistic under consideration. This test may tell us if there is a statistically significant difference in the algorithms' F_1 scores. If such a difference is detected, we follow up with a Nemenyi *post-hoc* test [9] to determine a performance ranking of the various algorithms.

Unsupervised Evaluation

The second method of evaluating the algorithms' performance only makes use of the quality of the returned results, that is, the quality quantified by the function $\varphi_{\mathcal{D}}(\cdot)$ on the held out validation dataset. We consider this useful for the following reasons. When we will consider real world data in Section 6.2, we clearly can no longer use the F_1 measure for evaluation, because there, we do not have access to the 'correct' answers. Hence, there, we will need a different method of evaluation, and by also applying that method here, we may compare how well both performance measures agree on a ranking of the algorithms. In particular, if one accepts that the F_1 measure is a plausible method to compare the algorithms' performance, we may evaluate the second method's merit by comparing the ranking obtained by that method with the one obtained using the F_1 score.

The method we propose to use is to construct a ranking based on two different measures. The first of these measures works as follows. When an algorithm returns a set of results on a given dataset, we look at the top k *unique* results returned, and take the average quality $\varphi_{\mathcal{D}}(\cdot)$ of these results. This averaged top-k quality is then the first quantity on which we will compare the various algorithms.

There is, however, an issue that needs addressing here; what to do if an algorithm returns less than k unique results? The method we propose to solve this would be to penalize the algorithm for having a low variety of results, i.e., by 'padding' the remainder of the unique results with

zero-quality dummy results. That is, if the algorithm returned $m < k$ unique results, we would count the remaining $k - m$ results as being of quality zero.

Interestingly, we may expect a ranking based on this method to correlate quite strongly with a ranking based on recall. The easiest way to see this is to consider an algorithm with perfect precision. Then, the number m of unique returned results is equal to the algorithm's recall, up to a constant inverse scaling factor of the number of subgroups in the data. If, for the moment, we make the (quite unreasonable) assumptions that all 'correct' subgroups have an equal score on the quality measure, and that furthermore all algorithms have perfect precision, then a ranking based on this method would be equivalent to a ranking based on the algorithms' recall. Now, under the much more reasonable assumption that a 'correct' subgroup has a much higher quality than an 'incorrect' subgroup, this would still roughly hold regardless of the algorithms' precision. That is, intuitively, adding a (relatively very) low quality result would not be that different from adding a zero-quality dummy result. If, finally, we drop the assumption that all 'correct' subgroups have equal quality, the resulting ranking becomes only slightly less intuitive. In particular, algorithms would be ranked in order of recall, but for algorithms with equal recall, there now arises a sub-ordering based on the quality of their corresponding 'correct' results. Hence, to conclude, we would expect a ranking of algorithms based on this 'padding with zero quality results' method to correlate quite strongly with a ranking of the algorithms based on their recall.

Contrariwise, we would not really expect a ranking based on this method to correlate with a ranking based on the algorithms' precision, although the specific resulting ranking would depend on the exact value of k. Intuitively, this can be seen by again initially considering an algorithm with perfect precision, and a fixed number m of unique results. Then, as k grows larger, the average quality (over the m 'correct' results' qualities and an increasing number of zeroes) will decrease. On the other hand, using the same argument as before, this would not be very different from including an increasing number of (very) low quality results in the algorithm's average. Hence, even if that algorithm would have a low precision, the average quality remains roughly on par with the case where the precision was perfect. Thus, for fixed k and m, this method does not really yield information about the algorithm's precision.

The second measure that we use is to simply take the average quality $\varphi_{\mathcal{D}}(\cdot)$ over *all* returned results. It should be relatively obvious how this measure *does* reflect an algorithm's precision. On the other hand, this measure, in turn, does not really give us information about the algorithm's recall. This can be seen by again starting from the assumption that all 'correct' subgroups have an equal quality. Then, a ranking based

on this method would only reflect the number of 'correct' subgroups in the algorithms' returned results, that is, the precision. As before, by dropping this assumption, the ranking becomes slightly less intuitive, now also reflecting the relative quality of the returned subgroups' qualities. Nevertheless, we would still expect a ranking based on this method to correlate quite strongly with a ranking based on the algorithms' precision.

As should be obvious from our use of the F_1 measure earlier, what we would ideally prefer is a single measure reflecting the performance on *both* of these measures in aggregate. To construct this measure, we take the same approach used by the F_1 score, that is, we construct it by taking the harmonic mean of the average top-k quality ('padded' with zeroes), and the average quality over the entire set of returned results. There are several reasons why we use the harmonic mean, instead of, say, the arithmetic mean. Firstly, because the measures that we combine are expected to reflect the same orderings as those of the recall and the precision, it makes sense to take this analogue to its final conclusion in the construction of the aggregate measure. Secondly, and somewhat more theoretically founded, the harmonic mean ensures that a relatively high score on only one of the measures will not dominate the aggregate score too much. Finally, somewhat related to that, because the two measures are expected to reflect rather different aspects of an algorithm's results, it is not obvious that the arithmetic mean would have a more meaningful interpretation than the harmonic mean.

To quantify the merits of these three measures, which we will refer to as '$\varphi_{\mathcal{D}}(\cdot)$ top-k', '$\varphi_{\mathcal{D}}(\cdot)$ all', and '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean', respectively, we perform several tests. For each experiment and dataset constructed in that experiment, we construct a ranking based on the algorithms' precision, recall, and F_1 score. We then compare these with the rankings obtained using the quality-based measures. Agreement is quantified by computing Kendall's τ rank correlation coefficient [16] between the two rankings obtained on each dataset, and taking the average of that quantity over all datasets in the experiment.

Briefly, Kendall's τ may be defined as follows. Suppose we compare the results of n algorithms $a_1, \ldots, a_n$ using two different quality measures $q_1(\cdot)$ and $q_2(\cdot)$. Clearly, both quality measures impose a bucket ordering on the algorithms, which we may define with the operators $\preceq_{q_1}$ and $\preceq_{q_2}$ for measure q_1 and q_2, respectively. Unfortunately, although τ is readily adapted to handle bucket orderings, the exposition is somewhat involved. The 'simple' version of τ assumes a complete ordering on the algorithms, and is much easier to explain. Hence, we will assume here that a complete ordering is defined by $\prec_{q_1}$ and $\prec_{q_2}$, and ignore the case where a bucket ordering exists.

Given two such orderings, a *concordance* function $C(\cdot)$ is defined over a pair of algorithms $\langle a_i, a_j \rangle$, $1 \leq i, j \leq n, i \neq j$, such that

$$C(a_i, a_j) \equiv \begin{cases} 1 & \text{if } (a_i \prec_{q_1} a_j) \wedge (a_i \prec_{q_2} a_j) \text{ or} \\ & \quad (a_j \prec_{q_1} a_i) \wedge (a_j \prec_{q_2} a_i), \\ -1 & \text{if } (a_i \prec_{q_1} a_j) \wedge (a_j \prec_{q_2} a_i) \text{ or} \\ & \quad (a_j \prec_{q_1} a_i) \wedge (a_i \prec_{q_2} a_j). \end{cases}$$

Thus, $C(a_i, a_j) = 1$ if $q_1(\cdot)$ and $q_2(\cdot)$ agree on the pairwise ordering of a_i and a_j, and $C(a_i, a_j) = -1$ if they disagree.

Kendall's τ rank correlation coefficient is now defined as

$$\tau(a_1, \ldots, a_n) = 2 \cdot \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} C(a_i, a_j)}{n(n-1)}.$$

Here, the numerator counts, over all distinct pairs of algorithms, the number of pairs on which the orderings agree, and subtracts the number of pairs on which the orderings disagree. The denominator then divides this by the total number of pairs, $\frac{1}{2}n(n-1)$, to normalize the quantity to the range $[-1, 1]$.

Qualitatively, Kendall's τ may now be interpreted as a normal correlation coefficient; a high value on this measure will indicate that the two ranking-methods generally agree on how to rank the algorithms, whereas either a low or negative value indicates that they do not correlate, or completely disagree, respectively.

In our evaluation of the different quality measures, a more intuitive evaluation is also performed by looking at the average ranks they assign to each algorithm. Regarding the choice k of the number of unique subgroups taken into account by the '$\varphi_{\mathcal{D}}(\cdot)$ top-k' measure, we will take this in the experiments on the synthetic data to be equal to the actual number of 'correct' subgroups in the datasets. For our later experiments on real world datasets, we will take this quantity to be $k = 5$. Finally, we again perform a Friedman test with a Nemenyi *post-hoc* test, using the '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' measure as the statistic under consideration, to quantify the algorithms' relative performance.

6.1.4 *Results*

In this section, we discuss the results of the experiments performed on the synthetic datasets. We begin by discussing the results of the set of experiments in which the number of 'correct' subgroups in the data was varied.

Varying Subgroup Counts

We here discuss the results of the set of experiments on synthetic data where the number of 'correct' subgroups in the data was varied. A summary of the results, using the supervised evaluation measures, is given in Figure 6.2. This figure shows the average values of these performance measures, over all 12 parameter settings and 50 datasets per setting.

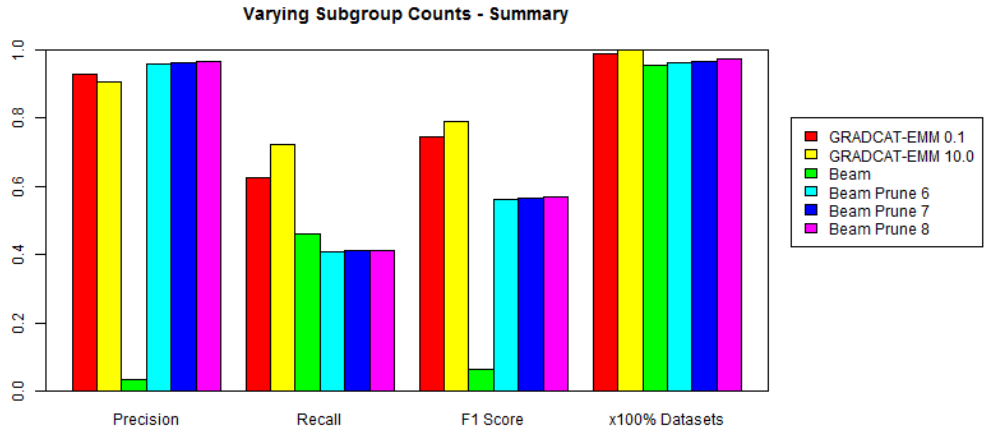

Figure 6.2: Summary of the supervised evaluation measures over all datasets where the number of correct subgroups was varied.

Here, in the legend, 'GRADCAT-EMM 0.1' and 'GRADCAT-EMM 10.0' refer to the GRADCAT-EMM algorithm with gradient ascent step size equal to 0.1 and 10.0, respectively. The entry 'Beam' indicates the normal BS algorithm, and 'Beam Prune' the BSP algorithm. For the latter, the search depth that was used is specified after the algorithm's identifier.

Considering first the algorithms' precision, we see that BS in particular performs a lot worse than the other algorithms. This may in part be explained by the fact that the BS algorithm, as specified, always returns a set of completely unique subgroup descriptions. That is, it is not possible for the algorithm to only return 'correct' subgroups, assuming that the beam width is larger than the number of correct subgroups in the data. On the one hand, this may thus seem like an unfair comparison between the algorithms. However, due to this property of the algorithm, an end-user may still run into the problem that is indicated by a low precision. That is, the 'correct' answers may still be buried in a set of relatively uninteresting results.

We also see that the BSP algorithm, by searching slightly deeper than BS and then pruning the resulting subgroups, manages to obtain a much higher precision than BS. This precision is also slightly higher than that of the GRADCAT-EMM algorithm.

Considering next the algorithms' recall, we see that, in comparison to BS, this quantity is negatively influenced by the BSP procedure. That is, by searching deeper than the depth at which all 'correct' subgroups may be found, it is possible that some of these results are removed from the algorithm's beam before the post-processing is performed. We also find that GRADCAT-EMM here outperforms both BS and BSP.

When we look at the aggregate score of these quantities, which is expressed by the F_1 score, we see that GRADCAT-EMM again outperforms both other algorithms. We also see that the bad performance of BS on the precision measure is here reflected in the algorithm's F_1 score. Further, although BSP scores slightly higher on precision than GRADCAT-EMM, we see that this is here compensated by the algorithm's lower recall.

Finally, the last measure displayed in Figure 6.2 expresses the percentage of datasets on which the algorithms managed to find at least one of the 'correct' subgroups. On average, we do not here observe a large difference between the various algorithms.

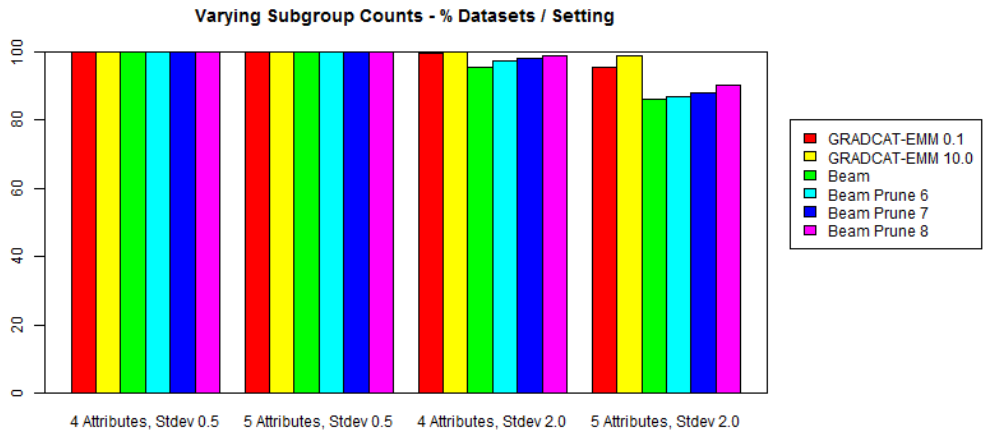

Figure 6.3: Percentage of datasets on which at least one correct subgroup was found, by dataset-setting.

This last measure is shown expanded over the different dataset-settings and the number of subgroups in the data, in Figure 6.3 and Figure 6.4, respectively. Considering first the data shown in Figure 6.3, we observe no real difference in the algorithms' performance on the datasets where the error variance of the global model was low. When this error variance is increased, we do observe a difference between the algorithms. Intuitively, increasing this variance corresponds to the 'correct' subgroups' data-points being 'buried' in the noise of the global model, making it harder to identify these individual subgroups. We also see that both BS and BSP appear to be more strongly influenced by this than GRADCAT-EMM. Further, we see that this effect is slightly stronger when the sub-

groups' descriptions are also more complex, that is, when the subgroups are defined on more attributes.

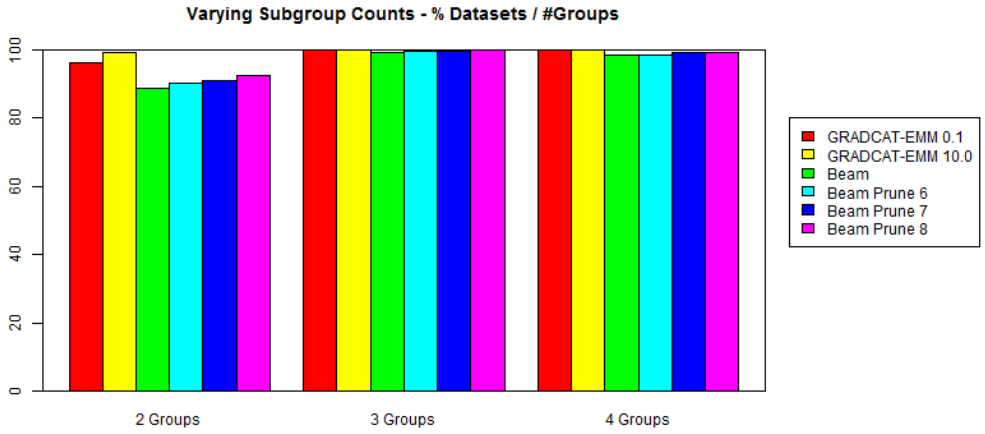

Figure 6.4: Percentage of datasets on which at least one correct subgroup was found, by number of subgroups in the data.

When we consider this measure expanded over the different number of subgroups in the data, shown in Figure 6.4, we further see that this difference mostly occurs when there are few 'correct' subgroups in the data. Colloquially, this may be explained as being due to the fact that, because there are less subgroups to be found, it must also be harder to find a single one such.

We show the algorithms' precision, recall, and F_1 score, expanded over the different dataset-settings, in Figure 6.5. We largely observe the same relative performance as already captured by the summary shown in Figure 6.2. Overall, we may also conclude that a dataset's 'difficulty' appears to correlate roughly with the left-to-right ordering of the dataset-parameters shown here.

We show the same measures, here expanded over the number of subgroups in the data, in Figure 6.6. We observe that the algorithms' precision appears to correlate with the number of subgroups in the data, which conforms to our earlier observation about the percentage of datasets on which at least one correct subgroup was found.

We next quantify the difference in the algorithms' F_1 score by performing a Friedman test on this measure, the results of which are shown in Table 6.3. We observe that the Friedman test detects a difference in the algorithms' F_1 score with $p \approx 0$. That is to say, the statistical software package, R [29], that we used for performing this test did not have the numerical precision to express the actual p-value. In any case, we may consider this difference to be statistically significant at any commonly used confidence level, and proceed to perform a Nemenyi *post-hoc* test to compare the performance of the individual algorithms. The results of

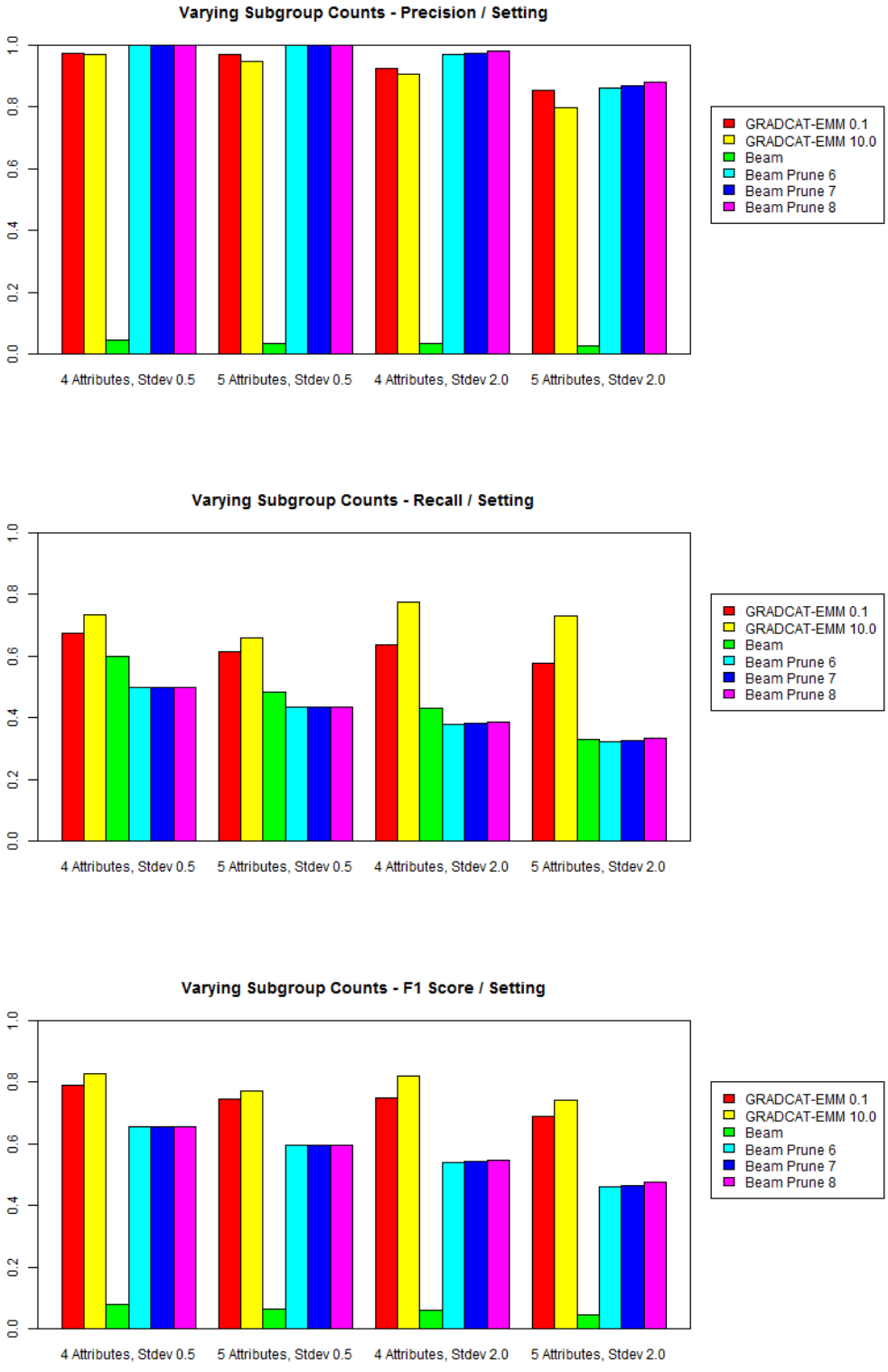

Figure 6.5: The algorithms' precision, recall, and F_1 score, by dataset-setting.

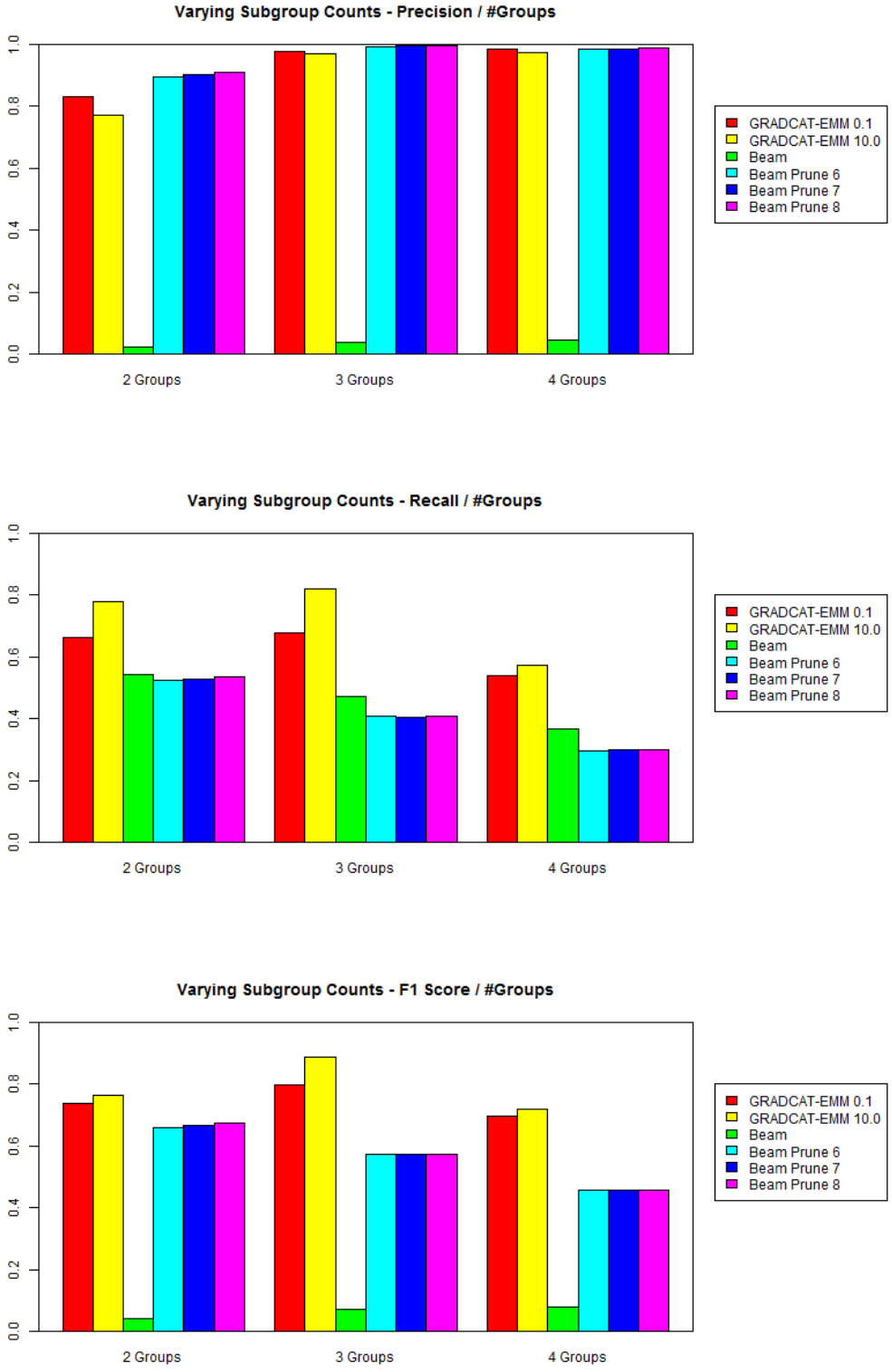

Figure 6.6: The algorithms' precision, recall, and F_1 score, shown by number of correct subgroups in the data.

BEST F_1 SCORES	VARYING SUBGROUP COUNTS					
FRIEDMAN TEST	$p \approx 0$					
ALGORITHM	AVERAGE RANK					
GRADCAT-EMM 10	2.00					
GRADCAT-EMM 0.1	2.50					
Beam Prune Depth 8	3.61					
Beam Prune Depth 7	3.63					
Beam Prune Depth 6	3.66					
Beam Search	5.60					
Pairwise p-values	GC 0.1	GC 10	BS	BS P6	BS P7	BS P8
GC 0.1	-	6×10^{-5}	0	5×10^{-14}	5×10^{-14}	6×10^{-14}
GC 10	6×10^{-5}	-	0	0	0	0
BS	0	0	-	0	0	0
BS P6	5×10^{-14}	0	0	-	0.99	0.99
BS P7	5×10^{-14}	0	0	0.99	-	0.99
BS P8	6×10^{-14}	0	0	0.99	0.99	-

Table 6.3: Results of a Friedman test on the algorithms' F_1 score. We detect a significant difference between the algorithms' performance on this measure with $p \approx 0$.

this test, as well as the average ranks assigned to the algorithms, are also shown in Table 6.3.

We find that GRADCAT-EMM significantly outperforms both BS and BSP at all commonly used confidence levels. Furthermore, the performance obtained by using a gradient ascent step size of 10.0 was significantly better than that obtained with a step size of 0.1. We also see that BSP significantly outperformed BS, although no significant difference was detected between the different search-depths that BSP used.

We will now move on to assessing the performance using the unsupervised evaluation measures. We first try to quantify the merit of these measures, by comparing the resulting rankings with the rankings obtained with the supervised measures. We do this by computing the average Kendall's τ between a pair of such performance measures, where τ is computed between the two rankings on a given dataset, and the average is taken over all datasets in the experiment. The results of this test are shown in Table 6.4.

We observe that the hypothesized correlations that we mentioned in Section 6.1.3 largely bear out. That is, we indeed find a fairly weak, and

CORRELATION OF RANKS VARYING SUBGROUP COUNTS

Supervised:	Precision		Recall		F_1 Score		
Unsupervised:	Top-k	All	Top-k	All	Top-k	All	H. Mean
Avg Kendall's τ	-0.389	0.738	0.867	0.124	0.231	0.386	0.822

Table 6.4: Average Kendall's τ rank correlation coefficient between pairings of sets of rankings by supervised and unsupervised evaluation measures.

in fact negative, correlation between rankings by precision and by the '$\varphi_{\mathcal{D}}(\cdot)$ top-k' measure. Contrariwise, we observe a fairly strong correlation between the precision and the '$\varphi_{\mathcal{D}}(\cdot)$ all' measure. Looking at the correlation between these unsupervised measures and rankings by recall, we see the hypothesized opposite effect. We finally compare rankings by both of these unsupervised measures, and by their harmonic mean, with rankings by the F_1 score. Here, we see that neither '$\varphi_{\mathcal{D}}(\cdot)$ top-k' or '$\varphi_{\mathcal{D}}(\cdot)$ all' correlates strongly with the F_1-based rankings. However, the aggregate measure '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' has a fairly strong correlation with the rankings by F_1 score.

When we consider the average ranks assigned to the algorithms by the various unsupervised evaluation methods, shown in Table 6.5, we observe somewhat less intuitive results. That is, referring back to the average precision and recall shown in Figure 6.2, we see that the average ranks assigned by '$\varphi_{\mathcal{D}}(\cdot)$ top-k' and '$\varphi_{\mathcal{D}}(\cdot)$ all' do not correspond with the overall ranking of the corresponding supervised measures. However, we see that the average ranks assigned by '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' and by the F_1 score, which for ease of reference are also shown in Table 6.5, do correspond.

In summary, we may from these observations so far cautiously conclude that in particular the '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' score appears to have some merit as an aggregate unsupervised evaluation measure.

We conclude by performing a Friedman test on the algorithms' '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' score, the results of which are shown in Table 6.6. We find a significant difference between the algorithms' performance on this measure, with $p \approx 0$. We proceed with the corresponding *post-hoc* test, which reveals that GRADCAT-EMM outperforms both BS and BSP at common confidence levels. Further, using a step size of 10.0 gives significantly better results than a step size of 0.1, with $p \approx 0.007$. BSP again significantly outperforms BS, but we fail to detect a significant difference in performance between the different search depths used by BSP.

RANKING BY METHOD	VARYING SUBGROUP COUNTS			
	$\varphi_{\mathcal{D}}$ TOP-k	$\varphi_{\mathcal{D}}$ ALL	$\varphi_{\mathcal{D}}$ H. MEAN	F_1 SCORE
GRADCAT-EMM 0.1	3.543 (5)	3.622 (5)	2.606 (2)	2.497 (2)
GRADCAT-EMM 10	3.282 (4)	3.616 (4)	2.233 (1)	1.999 (1)
Beam Search	5.501 (6)	5.879 (6)	5.872 (6)	5.600 (6)
Beam Prune Depth 6	2.908 (3)	2.658 (3)	3.463 (5)	3.664 (5)
Beam Prune Depth 7	2.878 (1)	2.621 (2)	3.428 (4)	3.630 (4)
Beam Prune Depth 8	2.888 (2)	2.604 (1)	3.399 (3)	3.610 (3)

Table 6.5: Average ranks assigned to the algorithms by various evaluation methods. For ease of reference, the corresponding re-ranking is shown in parentheses.

BEST $\varphi_{\mathcal{D}}$ SCORES	VARYING SUBGROUP COUNTS					
FRIEDMAN TEST	$p \approx 0$					
ALGORITHM	AVERAGE RANK					
GRADCAT-EMM 10	2.23					
GRADCAT-EMM 0.1	2.61					
Beam Prune Depth 8	3.40					
Beam Prune Depth 7	3.43					
Beam Prune Depth 6	3.46					
Beam Search	5.87					
Pairwise p-values	GC 0.1	GC 10	BS	BS P6	BS P7	BS P8
GC 0.1	-	7×10^{-3}	0	1×10^{-13}	5×10^{-13}	3×10^{-12}
GC 10	7×10^{-3}	-	0	0	3×10^{-14}	6×10^{-14}
BS	0	0	-	0	0	0
BS P6	1×10^{-13}	0	0	-	0.99	0.99
BS P7	5×10^{-13}	3×10^{-14}	0	0.99	-	0.99
BS P8	3×10^{-12}	6×10^{-14}	0	0.99	0.99	-

Table 6.6: Results of a Friedman test on the algorithms' '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' score. We detect a significant difference between the algorithms' performance on this measure with $p \approx 0$.

Varying Explanatory Variable Counts

In this section, we will discuss the results on the synthetic datasets, for the experiments in which the number of explanatory variables in the regression model was varied. A summary of these results, using the supervised evaluation measures, is shown in Figure 6.7. This figure shows the average values of the supervised measures over all 12×50 datasets used in these experiments.

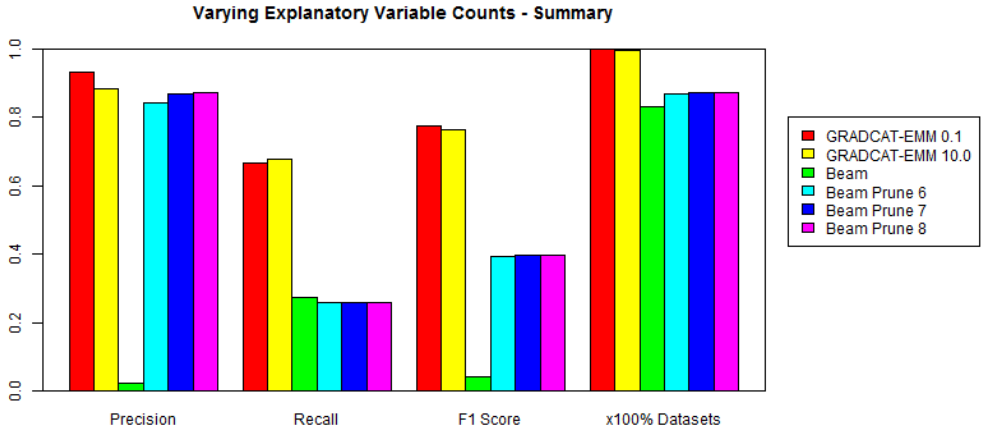

Figure 6.7: Summary of the supervised evaluation measures over all datasets where the number of explanatory variables was varied.

Here, we largely find the same relative performance of the algorithms as previously observed in the experiments where the number of subgroups in the data was varied. However, here, we do see a noticeable difference in performance between GRADCAT-EMM and both BS and BSP, in the percentage of datasets on which at least one correct subgroup was found.

We again show this measure expanded over the different dataset-settings and number of explanatory variables in Figure 6.8 and Figure 6.9, respectively.

Looking first at the difference in performance over the various dataset-settings, we see that GRADCAT-EMM's performance on this measure is not very sensitive to the 'difficulty' of the datasets, as we consider it here. This conforms to the observation from the experiments where the number of subgroups was varied. We also observe that both BS and BSP here perform worse than on these previous experiments.

When we look at the performance on this measure expanded over the number of coefficients in the regression model, we again see very little sensitivity in GRADCAT-EMM's performance. However, both BS and BSP appear to be rather sensitive to this variable. In fact, we observe that, when more than two explanatory variables are included in the model,

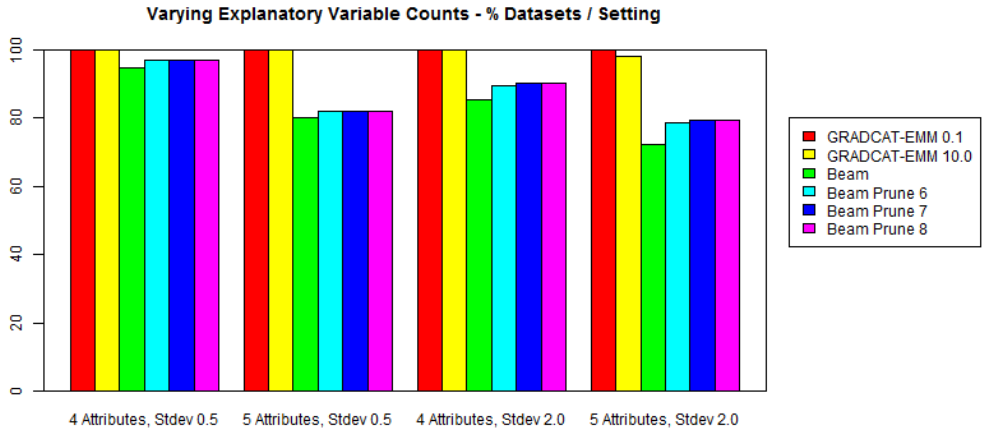

Figure 6.8: Percentage of datasets on which at least one correct subgroup was found, by dataset-setting.

both BS and BSP are able to find at least one correct subgroup on only about 80% of the datasets.

Performance on precision, recall, and F_1 score is again shown by dataset-setting, and number of explanatory variables, in Figure 6.10 and Figure 6.11, respectively. These results again largely conform to the summary shown in Figure 6.7. We do observe that GRADCAT-EMM does not appear to be very sensitive, on any of the performance measures used, to the number of coefficients in the regression model.

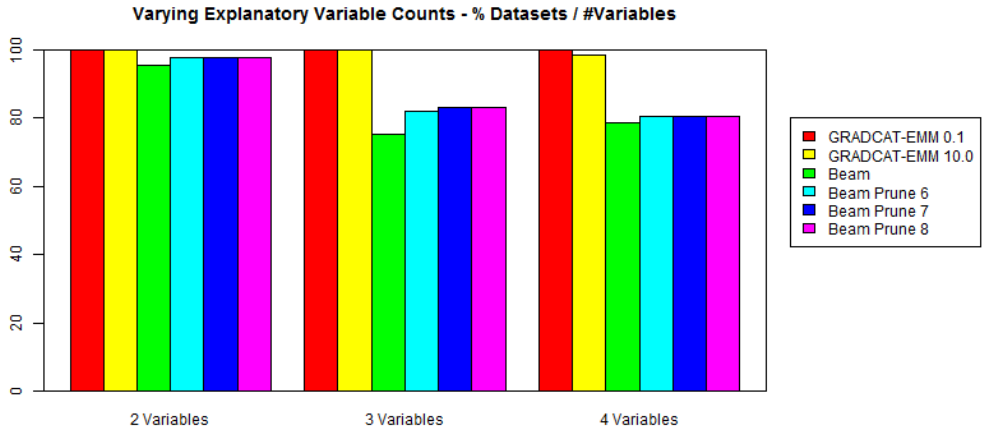

Figure 6.9: Percentage of datasets on which at least one correct subgroup was found, by number of explanatory variables in the regression model.

We again conclude the discussion regarding the supervised evaluation measures with a Friedman test on the algorithms' F_1 score. The results

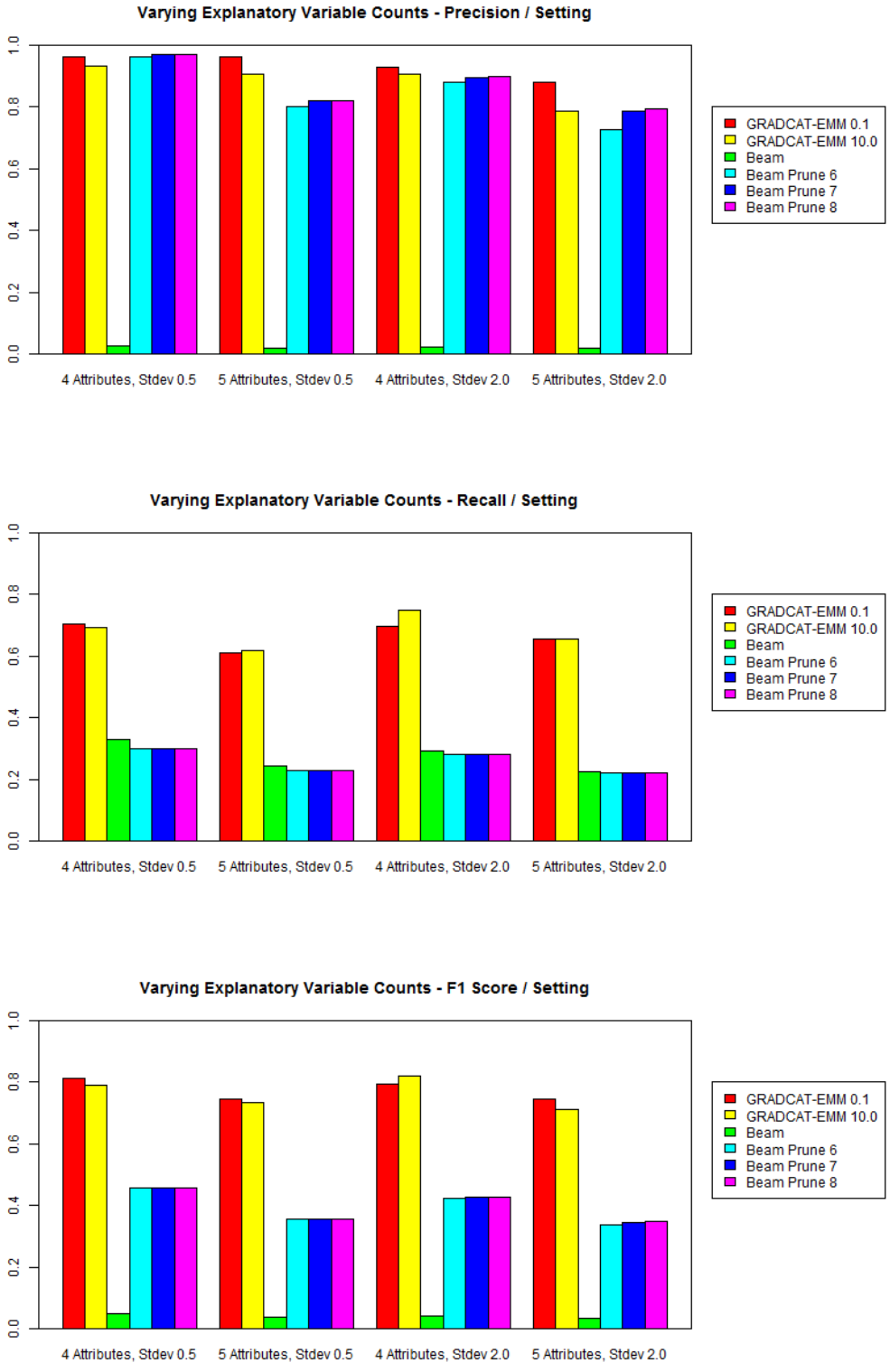

Figure 6.10: The algorithms' precision, recall, and F_1 score, by dataset-setting.

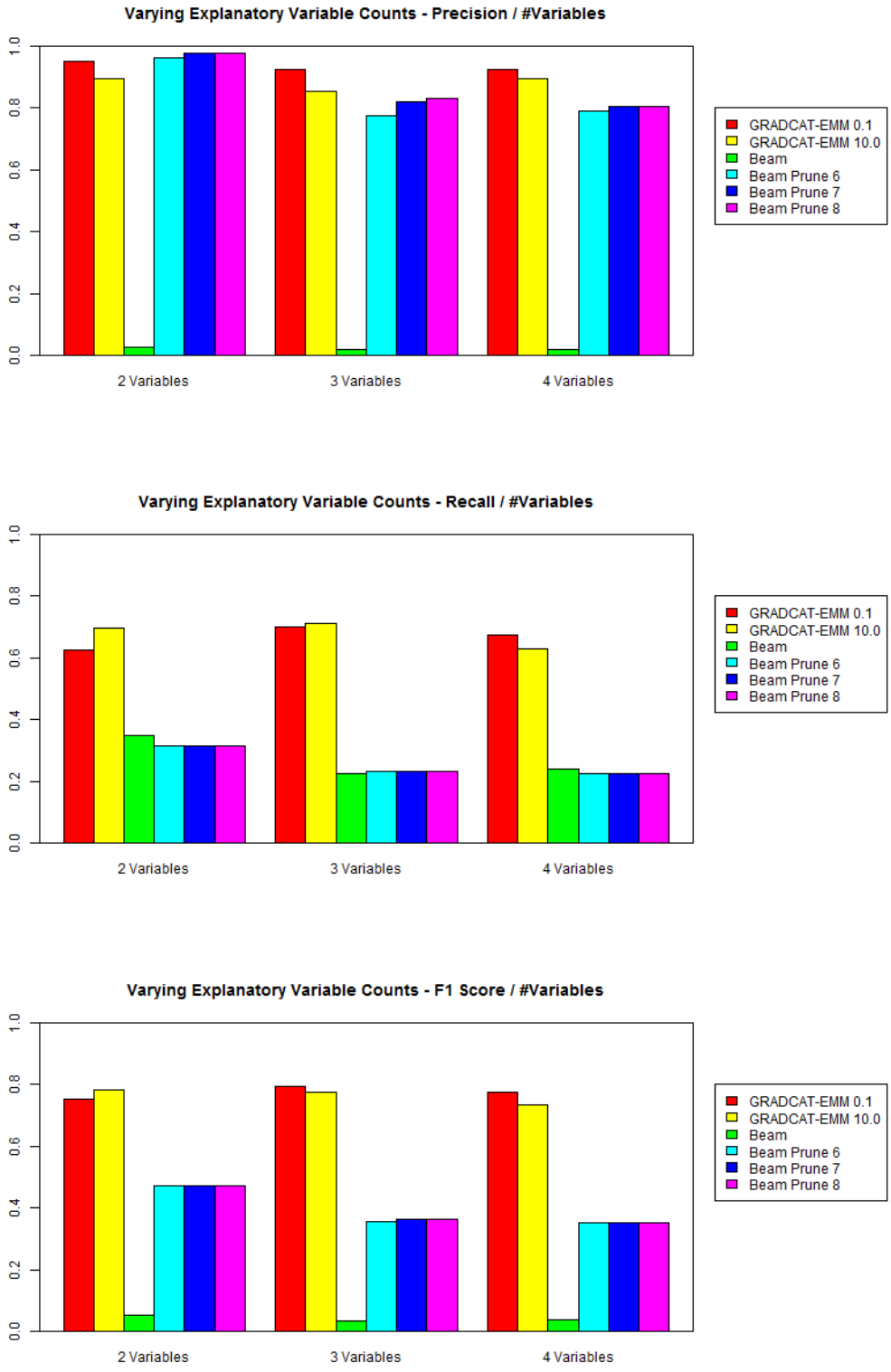

Figure 6.11: The algorithms' precision, recall, and F_1 score, by number of explanatory variables in the regression model.

BEST F_1 SCORES	VARYING β COUNTS
FRIEDMAN TEST	$p \approx 0$

ALGORITHM	AVERAGE RANK
GRADCAT-EMM 0.1	1.529
GRADCAT-EMM 10	1.648
Beam Prune Depth 7	4.361
Beam Prune Depth 8	4.361
Beam Prune Depth 6	4.363
Beam Search	4.738

Pairwise p-values	GC 0.1	GC 10	BS	BS P6	BS P7	BS P8
GC 0.1	-	0.88	0	0	0	0
GC 10	0.88	-	0	0	0	0
BS	0	0	-	6×10^{-3}	6×10^{-3}	6×10^{-3}
BS P6	0	0	6×10^{-3}	-	0.99	0.99
BS P7	0	0	6×10^{-3}	0.99	-	1.00
BS P8	0	0	6×10^{-3}	0.99	1.00	-

Table 6.7: Results of a Friedman test on the algorithms' F_1 score. We detect a significant difference between the algorithms' performance on this measure with $p \approx 0$.

of this test are shown in Table 6.7. We observe a significant difference in the algorithms' F_1 scores with $p \approx 0$.

Further, we find from the Nemeyi *post-hoc* test that GRADCAT-EMM significantly outperforms both BS and BSP, although we fail to detect a difference based on the different step sizes used. Furthermore, BSP scores significantly higher on the F_1 measure than BS, on any search depth used, with $p \approx 0.006$. However, we fail to detect a difference in performance between the different search depths used by BSP.

We again begin our discussion regarding the unsupervised measures by comparing the rankings obtained using these methods, with those computed from the supervised measures. The average Kendall's τ values between the sets of rankings obtained with the different measures are shown in Table 6.8.

Here, we again largely observe a conformance to the expected pairwise correlations. It is, however, of note that the '$\varphi_{\mathcal{D}}(\cdot)$ top-k' measure here also appears to correlate quite strongly with the rankings based on the F_1 score.

CORRELATION OF RANKS VARYING β COUNTS

Supervised:	Precision		Recall		F_1 Score		
Unsupervised:	Top-k	All	Top-k	All	Top-k	All	H. Mean
Avg Kendall's τ	-0.318	0.662	0.860	0.239	0.652	0.262	0.751

Table 6.8: Average Kendall's τ rank correlation coefficient between pairings of sets of rankings by supervised and unsupervised evaluation measures.

RANKING BY METHOD VARYING β COUNTS

	$\varphi_{\mathcal{D}}$ TOP-k	$\varphi_{\mathcal{D}}$ ALL	$\varphi_{\mathcal{D}}$ H. MEAN	F_1 SCORE
GRADCAT-EMM 0.1	1.712 (1)	2.753 (1)	1.653 (1)	1.529 (1)
GRADCAT-EMM 10	1.719 (2)	3.316 (5)	1.951 (2)	1.648 (2)
Beam Search	2.975 (3)	5.836 (6)	5.803 (6)	4.738 (6)
Beam Prune Depth 6	4.851 (4)	3.072 (4)	3.895 (5)	4.363 (5)
Beam Prune Depth 7	4.873 (6)	3.010 (2)	3.848 (3)	4.361 (3.5)
Beam Prune Depth 8	4.871 (5)	3.013 (3)	3.850 (4)	4.361 (3.5)

Table 6.9: Average ranks assigned to the algorithms by various evaluation methods. For ease of reference, the corresponding re-ranking is shown in parentheses.

Looking at the average ranks assigned by the different evaluation methods, shown in Table 6.9, we again find that the aggregate measure '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' conforms quite nicely to the ranking based on the F_1 score. We conclude that these observations strengthen our confidence in the merit of this unsupervised composite measure.

We finish our discussion of the results on these datasets with a Friedman test on the algorithms' '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' score, the results of which are shown in Table 6.10.

We again observe a significant difference between the algorithms' performance with $p \approx 0$. Subsequent pairwise testing of the algorithms reveals that both BS and BSP are statistically significantly outperformed by GRADCAT-EMM. Further, we find no significant difference, at the $\alpha = 0.05$ level, based on the different step sizes used. We also again fail to detect a difference between the various search depths used by BSP, at any reasonable choice of confidence level. However, BSP does significantly outperform BS under any choice of search depth.

BEST $\varphi_{\mathcal{D}}$ SCORES	VARYING β COUNTS					
FRIEDMAN TEST	$p \approx 0$					
ALGORITHM	AVERAGE RANK					
GRADCAT-EMM 0.1	1.653					
GRADCAT-EMM 10	1.951					
Beam Prune Depth 7	3.848					
Beam Prune Depth 8	3.850					
Beam Prune Depth 6	3.895					
Beam Search	5.803					
Pairwise p-values	GC 0.1	GC 10	BS	BS P6	BS P7	BS P8
GC 0.1	-	0.06	0	0	0	0
GC 10	0.06	-	0	0	0	0
BS	0	0	-	0	0	0
BS P6	0	0	0	-	0.99	0.99
BS P7	0	0	0	0.99	-	1.00
BS P8	0	0	0	0.99	1.00	-

Table 6.10: Results of a Friedman test on the algorithms' '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' score. We detect a significant difference between the algorithms' performance on this measure with $p \approx 0$.

Numeric Descriptive Attributes

This section describes the final set of experiments that we have performed on the synthetic datasets. Here, the subgroups' descriptive attributes were chosen to be numeric, as opposed to binary, which was the case for the other sets of experiments. The average values of the supervised performance measures, over all 4×50 datasets, are shown in Figure 6.12.

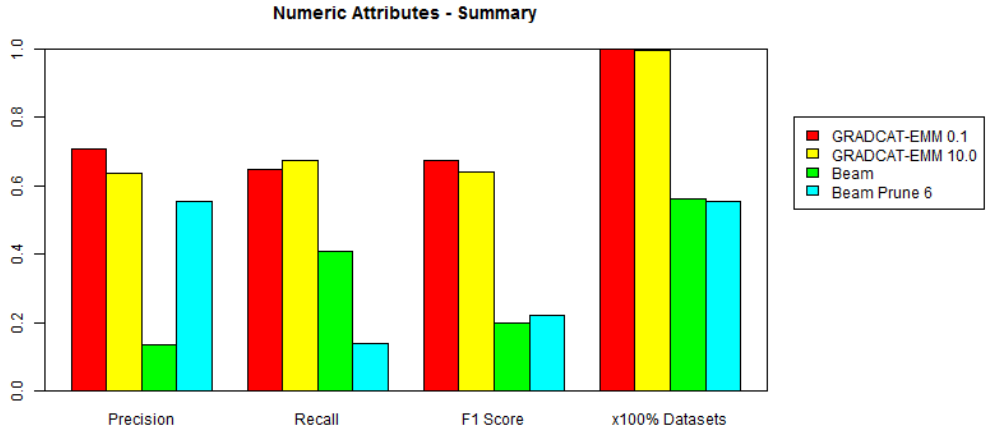

Figure 6.12: Summary of the supervised evaluation measures over all datasets where numeric descriptive attributes were used.

We see that the relative performance of the algorithms on the precision, recall, and F_1 scores largely conforms to the earlier observations on the other synthetic data experiments. However, we do see that all algorithms here perform worse on these measures, in absolute terms, than we have observed previously. Furthermore, although GRADCAT-EMM does not appear to be influenced by this very much, we see that both BS and BSP here have fairly bad performance in terms of the percentage of datasets on which at least one correct subgroup was found.

Expanding this latter performance measure over the various dataset-settings, shown in Figure 6.13, we see that none of the algorithms have much trouble when subgroups are defined with only two descriptive conditions. However, the performance of both BS and BSP shows a steep decline when this complexity is increased to four. We again find that GRADCAT-EMM does not appear to be very sensitive to this increase in complexity.

Moving on to a discussion of the other supervised evaluation measures, we show these results expanded over the different dataset-settings in Figure 6.14. Here, we observe similar behavior of the BS and BSP algorithms on all performance measures. That is, we find that their performance drops off steeply when the complexity of the subgroup descrip-

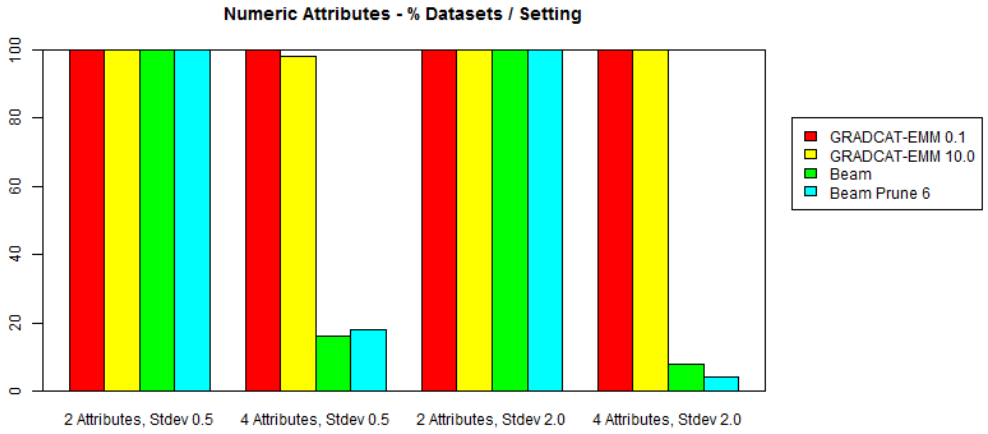

Figure 6.13: Percentage of datasets on which at least one correct subgroup was found, by dataset-setting.

tions is increased. In contrast to the percentage of datasets on which GRADCAT-EMM was able to find at least one correct subgroup, we do here observe a distinct sensitivity to the 'difficulty' of the dataset, albeit not as much as that displayed by BS and BSP.

A Friedman test on the F_1 score, the results of which are shown in Table 6.11, reveals a significant difference in the algorithms' performance on the measure, with $p \approx 3 \times 10^{-187}$. The *post-hoc* test finds that GRADCAT-EMM is significantly better than BS and BSP, regardless of the value of the step size used. It furthermore finds that a step size of 0.1 is here significantly better than a step size of 10.0. The test however fails to detect any difference in performance between BS and BSP.

We next show the rank correlations between the supervised and unsupervised performance measures in Table 6.12. We observe the expected effect on the pairings with precision and recall. However, we only observe a moderate average τ rank correlation coefficient between the ranking by the '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' measure, and by the F_1 score.

Regardless, looking at the average ranks assigned by both measures, shown in Table 6.13, we find that the two aggregate measures do conform quite closely.

Finishing our discussion of the experiments on the synthetic datasets by performing a Friedman test on the algorithms' performance on the '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' measure, we find a significant difference with $p \approx 4 \times 10^{-131}$. The results of this test are shown in Table 6.14. The Nemenyi *post-hoc* test reveals that GRADCAT-EMM significantly outperforms both BS and BSP. Furthermore, it finds that a step size of 0.1 is significantly better than one of 10.0, with $p = 0.002$. The pairwise test fails to detect a significant difference between BS and BSP.

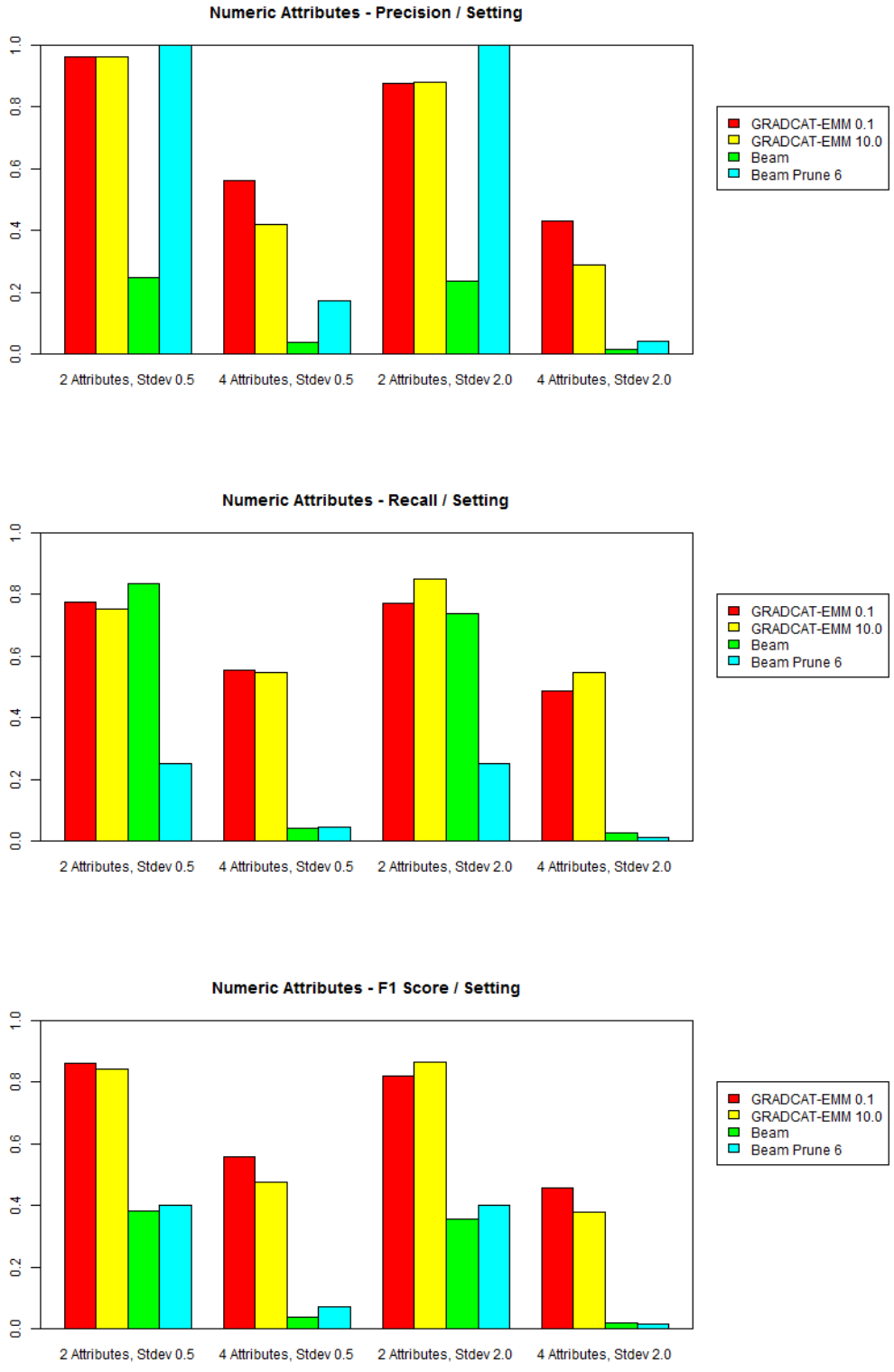

Figure 6.14: The algorithms' precision, recall, and F_1 score, by dataset-setting.

BEST F_1 SCORES	NUMERIC ATTRIBUTES
FRIEDMAN TEST	$p \approx 3 \times 10^{-187}$

ALGORITHM	AVERAGE RANK
GRADCAT-EMM 0.1	1.13
GRADCAT-EMM 10	2.60
Beam Prune Depth 6	3.11
Beam Search	3.16

Pairwise p-values	GC 0.1	GC 10	BS	BS P6
GC 0.1	-	0	0	0
GC 10	0	-	8×10^{-5}	5×10^{-4}
BS	0	8×10^{-5}	-	0.98
BS P6	0	5×10^{-4}	0.98	-

Table 6.11: Results of a Friedman test on the algorithms' F_1 score. We detect a significant difference between the algorithms' performance on this measure with $p \approx 3 \times 10^{-187}$.

CORRELATION OF RANKS	NUMERIC ATTRIBUTES						
Pearson's r	Precision		Recall		F_1 Score		
	Top-k	All	Top-k	All	Top-k	All	H. Mean
Avg Kendall's τ	0.030	0.686	0.780	-0.103	0.322	0.227	0.565

Table 6.12: Average Kendall's τ rank correlation coefficient between pairings of sets of rankings by supervised and unsupervised evaluation measures.

RANKING BY METHOD	NUMERIC ATTRIBUTES			
	$\varphi_{\mathcal{D}}$ TOP-k	$\varphi_{\mathcal{D}}$ ALL	$\varphi_{\mathcal{D}}$ H. MEAN	F_1 SCORE
GRADCAT-EMM 0.1	1.835 (1)	1.985 (2)	1.395 (1)	1.125 (1)
GRADCAT-EMM 10	1.915 (2)	2.565 (3)	1.850 (2)	2.603 (2)
Beam Search	2.303 (3)	3.723 (4)	3.425 (4)	3.163 (4)
Beam Prune Depth 6	3.948 (4)	1.728 (1)	3.330 (3)	3.110 (3)

Table 6.13: Average ranks assigned to the algorithms by various evaluation methods. For ease of reference, the corresponding re-ranking is shown in parentheses.

BEST $\varphi_{\mathcal{D}}$ SCORES	NUMERIC ATTRIBUTES
FRIEDMAN TEST	$p \approx 4 \times 10^{-131}$

ALGORITHM	AVERAGE RANK
GRADCAT-EMM 0.1	1.40
GRADCAT-EMM 10	1.85
Beam Prune Depth 6	3.33
Beam Search	3.43

Pairwise p-values	GC 0.1	GC 10	BS	BS P6
GC 0.1	-	2×10^{-3}	0	0
GC 10	2×10^{-3}	-	0	0
BS	0	0	-	0.88
BS P6	0	0	0.88	-

Table 6.14: Results of a Friedman test on the algorithms' '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' score. We detect a significant difference between the algorithms' performance on this measure with $p \approx 4 \times 10^{-131}$.

6.2 REAL WORLD DATA

As we mentioned at the beginning of this chapter, the second part of the experiments was performed on real world datasets. In the following, Section 6.2.1 describes the various datasets that we have used, as well as the models that were fitted on these datasets. In Section 6.2.2, we describe the experimental setup that we have used, and Section 6.2.3 discusses the results of these experiments in detail.

6.2.1 *Datasets*

The real world datasets that we have used are shown in Table 6.15. On each dataset, we have performed experiments with at least two different target models. The models that were induced on the corresponding datasets are given by Table 6.16. Each dataset is here identified both by a name relating it to the domain that the data originates from, and a number to identify the different models that were induced on the dataset.

When a dataset is identified in Table 6.15 as having model type 'Linear Regression', this means that the target variable was numeric, and at least one explanatory variable was included in the model. Models whose type is identified as 'Linear Probability' were induced on a binary target variable, and here we again used at least one explanatory variable. When a model is identified as 'Mean', this essentially indicates that it may be

DATASET	MODEL TYPE	n	k	p
EAEF 1 [10]	Linear Regression	2485	39	3
Wine 1 [8]	Linear Regression	9600	6	4
Extra Marital 1 [14]	Linear Regression	6366	5	4
Extra Marital 2 [14]	Linear Regression	6366	7	2
Contraceptives 1 [25, 1]	Linear Regression	1473	8	2
Adult 1 [20, 1]	Linear Probability	30162	5	4
Credit 1 [14]	Linear Probability	13390	5	3
Credit 2 [14]	Linear Probability	10461	5	5
Contraceptives 2 [25, 1]	Linear Probability	1473	7	3
EAEF 2 [10]	Mean	2485	41	1
Wine 2 [8]	Mean	9600	7	1
Extra Marital 3 [14]	Mean	6366	8	1
Adult 2 [20, 1]	Mean	30162	5	1
Credit 3 [14]	Mean	10461	7	1
Contraceptives 3 [25, 1]	Mean	1473	8	1

Table 6.15: Information about the datasets used. The first column gives the dataset's identifier that we will use throughout this work, as well as the source from which the data was obtained. The second column identifies the model type that was induced on each dataset. The column denoted with n shows the number of records in the dataset. The columns marked k and p give the number of descriptive attributes, and coefficients included in the model, respectively. Do note that the number of descriptive attributes k should not be confused with the number of unique subgroups taken into account by the '$\varphi_{\mathcal{D}}(\cdot)$ top-k' measure, which in these experiments uses a constant choice $k = 5$.

regarded as a Subgroup Discovery problem. That is, in this case the target variable may have been either numeric or binary, and no explanatory variables were included in the model.

6.2.2 *Experimental Setup*

To quantify the performance of the algorithms on these real world datasets, the following approach was used. For each algorithm, a set of parameters was created on which they could be varied. Noting that real world datasets' characteristics may be much more disparate than the relatively controlled datasets used in Section 6.1, the sets of values that these parameters could take were constructed to be somewhat larger than

DATASET	MODEL
EAEF 1	Earnings $= \beta_0 + \beta_1 \times$ YearsSchool $+ \beta_1 \times$ YearsExperience
Wine 1	Price $= \beta_0 + \beta_1 \times$ Cases $+ \beta_2 \times$ Score $+ \beta_3 \times$ Age
Extra Marital 1	YRB $= \beta_0 + \beta_1 \times$ Age $+ \beta_2 \times$ YearsMarried $+ \beta_3 \times$ #Children
Extra Marital 2	YRB $= \beta_0 + \beta_1 \times$ MarriageRating
Contraceptives 1	#Children $= \beta_0 + \beta_1 \times$ Age
Adult 1	($\geq$50K) $= \beta_0 + \beta_1 \times$ Age $+ \beta_2 \times$ fnlwgt $+ \beta_3 \times$ HoursWeek
Credit 1	IsCardHolder $= \beta_0 + \beta_1 \times$ MajorDrg $+ \beta_2 \times$ MinorDrg
Credit 2	Default $= \beta_0 + \beta_1 \times$ Income $+ \beta_2 \times$ IncPer $+ \beta_3 \times$ Spending $+$ $\beta_4 \times$ MinorDrg
Contraceptives 2	UsesContraceptive $= \beta_0 + \beta_1 \times$ Age $+ \beta_2 \times$ #Children
EAEF 2	Earnings $= \beta_0$
Wine 2	Price $= \beta_0$
Extra Marital 3	YRB $= \beta_0$
Adult 2	($\geq$50K) $= \beta_0$
Credit 3	Default $= \beta_0$
Contraceptives 3	UsesContraceptive $= \beta_0$

Table 6.16: Models fitted on the various datasets.

GRADCAT-EMM	
Results Generated	50
Min Records in Leaf for Split	125
Min Records in Child for Split	50
Min iterations	10
Max iterations	350
Gradient Ascent Step Size	0.1, 10
Gradient Convergence Threshold	0.1
Pruning α	0.01, 0.05, 0.10
Max Runtime per Run (Minutes)	10
BEAM SEARCH	
Search Depth	1, 2, 4
Minimum Support	50
Beam Width	50
Num Numeric Split Bins	12
BEAM SEARCH WITH POST-PROCESSING	
Search Depth	2, 4
Minimum Support	50
Beam Width	50
Num Numeric Split Bins	12
Pruning α	0.01, 0.05, 0.10

Table 6.17: Parameter settings of the different algorithms.

those previously considered. The various settings used are shown in Table 6.17.

For each dataset and algorithm, then, the algorithm's parameters were optimized and performance evaluated using repeated randomly sub-sampled 5-fold cross-validation with a 4-fold inner-cross-validation parameter optimization loop. In particular, each dataset was uniformly sub-sampled into 5 folds of (roughly) equal size, such that the same folds were used for each algorithm. Furthermore, these folds were constructed such that they contained at most 1000 records each. Then, from these 5 folds, one was chosen as the evaluation data, one as parameter optimization data, and the remaining three as training data. Then, for each setting of the algorithm, a set of subgroups was retrieved from the training data, and the quality of this set of results evaluated on the optimization data using the measure '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' outlined in

Section 6.1.3. The various parameter settings were then ranked using this quality measure. This process was repeated so that, for the same choice of evaluation fold, each of the other four folds was chosen exactly once as the optimization fold. Then, over these 4 runs, the parameter setting that on average ranked highest was chosen as the best setting. The algorithm was then run, using this setting, on all four folds, and the performance finally evaluated on the evaluation fold. This process was then repeated such that every fold was chosen exactly once as the evaluation fold, yielding a total of 5 quality scores for each algorithm. This entire process was repeated a total of 10 times, using a different sub-sampling of the dataset into 5 folds each time.

Thus, in total, we obtained 50 quality scores for each dataset and algorithm pair. For each dataset, all algorithms' quality scores were paired by the evaluation folds, and the algorithms ranked in order of quality. Average ranks of the algorithms were computed over each set of 5 evaluation folds, in order to aggregate the cross-validation results. The algorithms were subsequently re-ranked to obtain a ranking for each sub-sampling of the data. Finally, then, we obtained 10 rankings of the algorithms for each dataset, over which average ranks were computed as the final performance measure on each dataset. A Friedman test with corresponding Nemenyi *post-hoc* test was then performed, over all datasets, using these average ranks as the statistic under consideration, to evaluate the algorithms' relative performance.

6.2.3 *Quantitative Results*

The average ranks obtained by the algorithms, on the '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' measure, are shown for each dataset in Table 6.18. Each row here corresponds to the average ranks on the indicated dataset, where the average was taken over the rankings obtained on the 10 sub-samplings. For each dataset, a corresponding re-ranking is shown in parentheses. The row toward the bottom of the table, denoted 'Overall', indicates the average of these re-rankings, which is also the quantity used for the Friedman test.

We observe that the BS algorithm on average performed best, followed by GRADCAT-EMM, with BSP performing the worst on average. A Friedman test reveals that there is a significant difference between the algorithms' performance, under most commonly used confidence levels, with $p = 0.0031$. We follow up with a pairwise comparison of the algorithms by performing a Nemenyi *post-hoc* test, which reveals that BS outperforms GRADCAT-EMM and BSP with $p = 0.03$ and $p = 0.009$, respectively. No statistically significant difference is observed between GRADCAT-EMM and BSP.

AVERAGE RANKS	$\varphi_{\mathcal{D}}$ H. MEAN		
	GC	BS	BSP
EAEF 1	1.10 (1)	2.00 (2)	2.90 (3)
Wine 1	2.85 (3)	1.35 (1)	1.80 (2)
ExtraMarital 1	2.00 (2)	1.00 (1)	3.00 (3)
ExtraMarital 2	2.45 (2)	1.00 (1)	2.55 (3)
Contraceptives 1	1.90 (2)	1.15 (1)	2.95 (3)
Adult 1	3.00 (3)	1.10 (1)	1.90 (2)
Credit 1	2.10 (2)	1.60 (1)	2.30 (3)
Credit 2	1.00 (1)	2.00 (2)	3.00 (3)
Contraceptives 2	3.00 (3)	1.00 (1)	2.00 (2)
EAEF 2	2.05 (2)	1.25 (1)	2.70 (3)
Wine 2	3.00 (3)	1.75 (2)	1.25 (1)
ExtraMarital 3	1.80 (1)	2.25 (3)	1.95 (2)
Adult 2	2.90 (3)	1.05 (1)	2.05 (2)
Credit 3	3.00 (3)	1.00 (1)	2.00 (2)
Contraceptives 3	2.90 (3)	1.00 (1)	2.10 (2)
Overall	2.27 (2)	1.33 (1)	2.40 (3)
Friedman	$p = 0.0031$		
Pairwise	GC	BS	BSP
GC	-	0.03	0.93
BS	0.03	-	9×10^{-3}
BSP	0.93	9×10^{-3}	-

Table 6.18: Average ranks of the algorithms on the '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' measure, by dataset. A Friedman test shows that there is a significant difference in the algorithms' performance with $p = 0.0031$.

Interestingly, this result is in stark contrast with all the results observed on the experiments with synthetic datasets. An obvious question at this point would therefore be how we can explain this difference in results. It seems likely that this effect may in part be explained by a general difference in the characteristics of the datasets involved in these different sets of experiments. We will return to this point later.

There is, however, at least one other property that is distinctly different in the experiments that we consider here. Specifically, in the experiments with synthetic data, the BS algorithm was told in advance the search depth at which the 'correct' subgroups could be found. Contrariwise, we here choose that parameter by optimizing on the quality measure under consideration.

We will in the following argue that this distorts which characteristics of the results are quantified by the quality measure '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean'. That is, it causes the quality measure to rank certain sets of results higher than we would subjectively desire, given these results' characteristics. We will perform further analysis of the data obtained, leading us to conclude that this effect is due to a characteristic of the results that is not properly accounted for by the used quality measure, which BS can exploit by explicitly optimizing on the search depth. We will then therefore suggest an adaptation of the quality measure that we have used here.

The reasoning is as follows. As was noted by van Leeuwen *et al.* [23], the BS algorithm has a tendency to return result sets containing subgroups whose descriptions are largely the same, differing mostly in 'noise' conditions in these descriptions. Some consideration will reveal that this effect is due to searching too deep, that is, by a 'wrong' specification of the algorithm's search depth. In particular, say that some hypothetical 'correct' subgroup G is found on, say, depth d. Now, if the algorithm were to search up to depth $d + 1$, it is very likely that the algorithm's beam will be saturated with minor variations of G.

To see why this may be an issue, consider that it will (subjectively 'falsely') lead to the quality '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' being higher than that of a set of results containing only the 'underlying', 'correct', subgroups, that is, the ones without the 'noise' conditions. In particular, this happens because of two factors. First, and most troublesome, it hides the fact that the algorithm's returned results are much less diverse than would be obvious on first sight. This is problematic in particular because part of our quality measure is based on the top-k *unique* results. That is, the noise conditions allow the 'same' subgroup to be counted multiple times, obviously offsetting our stipulation of the uniqueness of the results used for this measure's computation. The other factor is the beam's saturation with minor variations of only the 'correct' subgroups, which will quite easily raise the value of the '$\varphi_{\mathcal{D}}(\cdot)$ all' measure to around the

same level as that of the best 'correct' subgroups at depth $d-1$. This latter property is one that the other two algorithms also have, which is to say, both the GRADCAT-EMM and Beam Search Prune algorithms' results also tend to contain multiple copies of the same subgroup. However, there, such copies tend to be strictly the same, which does not interfere with the uniqueness-based measure.

We again note that none of this was an issue in our earlier experiments on synthetic data, because there, we explicitly told the beam search algorithm the search depth at which the 'correct' answers were to be found. Hence, there, the algorithm was never allowed to search deep enough to start adding such noise conditions to its results. On the other hand, had we there instead also optimized the BS algorithm's search depth on the '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' measure, we hypothesize that the algorithm's precision and recall would both drop to near zero. Clearly, doing so would then there lead to a rather unfair evaluation of the algorithm's performance, so we consider the method of telling it the correct search depth to be a better approach.

Returning to the experiments considered here, and as a start to quantifying this attribution of the results to an interference with the uniqueness-based measure, we show the rankings obtained on the '$\varphi_{\mathcal{D}}(\cdot)$ top-k' and '$\varphi_{\mathcal{D}}(\cdot)$ all' measures separately in Table 6.19. These rankings were generated from the same data as was used in Table 6.18, which is to say, they were computed from the same result sets used to construct the rankings shown previously. Furthermore, the algorithms were ranked and the results aggregated identically to the method outlined in Section 6.2.2.

Looking first at the rankings obtained on the '$\varphi_{\mathcal{D}}(\cdot)$ top-k' measure, we observe that BS performs best on this measure on all but one of the datasets. We also see that GRADCAT-EMM is on average ranked slightly better than BSP. A Friedman test show that this difference is significant with $p \approx 2 \times 10^{-7}$. Subsequent pairwise comparison of the algorithms shows that BS performs significantly better than GRADCAT-EMM and BSP, with respectively $p \approx 0.0005$ and $p \approx 0.0003$. We find no significant difference between GRADCAT-EMM and BSP.

When we consider the rankings obtained on the '$\varphi_{\mathcal{D}}(\cdot)$ all' measure, shown in the right-hand column of Table 6.19, we see that the algorithms perform on average roughly on par. This is confirmed by a Friedman test which fails to detect a significant difference in these results at any plausible confidence level. As such, we do not follow up with the corresponding *post-hoc* test.

These results appear to confirm that the results obtained on the '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' measure may largely be attributed to BS's better performance on the top-k based measure. However, by itself this does not show whether there is anything 'wrong' with BS's scoring on this measure. That is, it is in principle completely possible that BS legitimately

manages to consistently find a few subgroups with much higher quality than those found by either of the other algorithms.

To investigate whether the top-k based measure here properly captures the characteristics that we intended, we again first note that it was constructed to, intuitively, resemble a quantification similar to an algorithm's recall. As such, it depended on the uniqueness of the results used for the computation of this scoring function, so as not to conflate this quantification of a result set's quality with its precision. We will next attempt to quantify the uniqueness, or rather, the *lack of diversity*, of the subgroups used in the computation of the different quality measures.

One way to quantify this lack of diversity was proposed by van Leeuwen *et al.* [23], and they refer to this measure as the *cover redundancy* of a set of subgroups. Intuitively, it measures the average deviation of the number of subgroup descriptions that cover each record, from what would be expected under a uniform distribution of all subgroups over all records. As is mentioned in their work, such an assumption of a uniform distribution is clearly unlikely given the nature of the algorithms involved in solving the EMM problem, but it does allow one to compare, on the same dataset, the diversity of the sets of results returned by different algorithms.

The cover redundancy (CR) measure may be defined as follows. Let Ω denote a set of returned subgroups, and let $\iota(\cdot)$ denote the indicator function. The *cover count* of the i-th record is written c_i, and defined as $c_i \equiv \sum_{G \in \Omega} \iota(r_i \in G)$. That is, c_i simply counts the number of subgroups in Ω in which the i-th record is included. The *expected cover count*, $\hat{c}$, is then defined as $\hat{c} \equiv \frac{1}{n} \cdot \sum_{i=1}^{n} c_i$. Finally, the cover redundancy $\mathrm{CR}(\cdot)$ of a set of subgroups Ω is defined as

$$\mathrm{CR}(\Omega) \equiv \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{|c_i - \hat{c}|}{\hat{c}},$$

where $|\cdot|$ denotes the absolute value. A lower CR then indicates a lower deviation from the uniform, viz., maximally diverse, distribution of the records over the subgroups. That is, a low CR is desirable because it indicates that the subgroups are all relatively distinct.

We next computed the CR of the sets of subgroups used in the computation of the '$\varphi_{\mathcal{D}}(\cdot)$ top-k' and '$\varphi_{\mathcal{D}}(\cdot)$ all' measures. We then ranked the algorithms by their performance on this measure, and aggregated the ranks over the cross-validation folds and sub-samplings in the aforementioned fashion. The average ranks so obtained are shown for each dataset in Table 6.20.

Considering first the ranking by CR on the top-k results, we find that BS is here the worst performing algorithm. We also observe a small difference in the average ranks of GRADCAT-EMM and BSP, with the latter scoring slightly better. However, a Friedman test fails to detect a signif-

AVERAGE RANKS	$\varphi_{\mathcal{D}}$ TOP-k			$\varphi_{\mathcal{D}}$ ALL		
	GC	BS	BSP	GC	BS	BSP
EAEF 1	1.95 (2)	1.05 (1)	3.00 (3)	1.00 (1)	2.15 (2)	2.85 (3)
Wine 1	2.40 (2.5)	1.20 (1)	2.40 (2.5)	2.85 (3)	1.35 (1)	1.80 (2)
ExtraMarital 1	2.00 (2)	1.00 (1)	3.00 (3)	1.90 (2)	3.00 (3)	1.10 (1)
ExtraMarital 2	2.05 (2)	1.00 (1)	2.95 (3)	2.45 (3)	2.25 (2)	1.30 (1)
Contraceptives 1	2.00 (2)	1.00 (1)	3.00 (3)	2.10 (2)	2.30 (3)	1.60 (1)
Adult 1	3.00 (3)	1.00 (1)	2.00 (2)	3.00 (3)	1.70 (2)	1.30 (1)
Credit 1	2.30 (2)	1.00 (1)	2.70 (3)	1.90 (1)	2.10 (3)	2.00 (2)
Credit 2	1.35 (1)	1.65 (2)	3.00 (3)	1.00 (1)	2.10 (2)	2.90 (3)
Contraceptives 2	3.00 (3)	1.00 (1)	2.00 (2)	3.00 (3)	1.10 (1)	1.90 (2)
EAEF 2	2.85 (3)	1.00 (1)	2.15 (2)	1.70 (1)	1.85 (2)	2.45 (3)
Wine 2	3.00 (3)	1.30 (1)	1.70 (2)	3.00 (3)	1.85 (2)	1.15 (1)
ExtraMarital 3	3.00 (3)	1.00 (1)	2.00 (2)	1.15 (1)	2.85 (3)	2.00 (2)
Adult 2	2.20 (2)	1.00 (1)	2.80 (3)	3.00 (3)	1.25 (1)	1.75 (2)
Credit 3	3.00 (3)	1.00 (1)	2.00 (2)	2.30 (3)	2.10 (2)	1.60 (1)
Contraceptives 3	2.95 (3)	1.05 (1)	2.00 (2)	2.70 (3)	1.20 (1)	2.10 (2)
Overall	2.43 (2)	1.07 (1)	2.50 (3)	2.20 (3)	2.00 (2)	1.80 (1)
Friedman	$p \approx 2 \times 10^{-7}$			$p = 0.56$		
Pairwise	GC	BS	BSP			
GC	-	5×10^{-4}	0.98			
BS	5×10^{-4}	-	3×10^{-4}			
BSP	0.98	3×10^{-4}	-			

Table 6.19: Average ranks of the algorithms on the '$\varphi_{\mathcal{D}}(\cdot)$ top-k' and '$\varphi_{\mathcal{D}}(\cdot)$ all' measures, by dataset. Two Friedman tests show that there is a significant difference in the algorithms' performance on the top-k based score, with $p \approx 2 \times 10^{-7}$, but no statistically significant difference is detected on the other measure.

icant difference in these results at the $\alpha = 0.05$ level. At $\alpha = 0.10$, the difference is considered significant, but subsequent pairwise testing fails to detect to which algorithm this difference should then be attributed.

When we look at the CR of the algorithms' entire sets of results, shown in the right-hand column of Table 6.20, we find that BSP is again ranked highest, with BS here also being the worst on average. We could consider this difference to be significant with $p = 0.0321$. The *post-hoc* test then finds, at $\alpha = 0.05$, that BSP performs better than BS with $p = 0.046$ (p-value shown rounded to 0.05 in the table). No pairwise significant difference is detected between the other algorithms.

How, then, should we interpret these results? It seems clear that BS's result-sets on average have a higher CR than those of the other algorithms. Further, we find that it ranks, on average, substantially lower than the other two. However, we cannot really consider this difference to be significant at any plausible confidence level. Before moving on to some concluding remarks about the results observed so far, we will first finally look at another measure of the results' quality. Here, we will try to quantify the amount of 'noise' that is present in the descriptions of a set of returned subgroups.

To motivate the use of this measure, consider a set of subgroups largely consisting of minor variations of some relatively small set of high quality subgroups. That is, let the majority of this set consist of subgroups whose description is obtained by adding a single, marginally influential, condition to the description of any of the subgroups in the small set of high quality subgroups. Assume furthermore that these 'non-refined' other subgroups generalize well, which is to say, that they have a high quality on both the training data, and on the held-out evaluation data.

We argue that a substantial number of the subgroups constructed in this way will still have relatively high quality when evaluated on the evaluation data. Specifically, we argue that these additional conditions may be considered 'noise', in that they are largely non-informational, but that this will not be offset by evaluating these subgroups on a separate dataset.

To see this, assume that one is given a high quality subgroup description, i.e., a description that covers, on both the training and the evaluation data, a subgroup with high quality. Now consider adding a condition to this description that only marginally changes the subgroups that would be covered by the original description. Taking the limit where the subgroup is only changed infinitesimally, we see that the 'refined' subgroup must necessarily also have high quality on both datasets. Relaxing this limit somewhat, we see that this holds for any condition that does not change the subgroups covered by the original description too much.

AVERAGE RANKS	CR TOP-k			CR ALL		
	GC	BS	BSP	GC	BS	BSP
EAEF 1	2.25 (3)	1.85 (1)	1.90 (2)	2.45 (2)	1.00 (1)	2.55 (3)
Wine 1	1.20 (1)	2.15 (2)	2.65 (3)	1.00 (1)	2.70 (3)	2.30 (2)
ExtraMarital 1	1.10 (1)	3.00 (3)	1.90 (2)	1.00 (1)	2.60 (3)	2.40 (2)
ExtraMarital 2	1.60 (2)	3.00 (3)	1.40 (1)	1.00 (1)	3.00 (3)	2.00 (2)
Contraceptives 1	1.00 (1)	3.00 (3)	2.00 (2)	1.10 (1)	2.95 (3)	1.95 (2)
Adult 1	2.95 (3)	2.05 (2)	1.00 (1)	2.30 (2)	2.65 (3)	1.05 (1)
Credit 1	2.20 (2)	2.70 (3)	1.10 (1)	2.00 (2)	2.40 (3)	1.60 (1)
Credit 2	1.30 (1)	2.05 (2)	2.65 (3)	1.35 (1)	1.65 (2)	3.00 (3)
Contraceptives 2	2.00 (2)	3.00 (3)	1.00 (1)	2.65 (3)	2.35 (2)	1.00 (1)
EAEF 2	1.00 (1)	3.00 (3)	2.00 (2)	2.20 (2)	2.55 (3)	1.25 (1)
Wine 2	2.80 (3)	1.25 (1)	1.95 (2)	1.00 (1)	3.00 (3)	2.00 (2)
ExtraMarital 3	1.55 (1)	2.15 (2)	2.30 (3)	2.25 (3)	1.60 (1)	2.15 (2)
Adult 2	1.55 (2)	2.95 (3)	1.50 (1)	2.35 (2)	2.55 (3)	1.10 (1)
Credit 3	2.30 (2)	2.60 (3)	1.10 (1)	2.80 (3)	2.20 (2)	1.00 (1)
Contraceptives 3	1.80 (2)	3.00 (3)	1.20 (1)	2.25 (2)	2.75 (3)	1.00 (1)
Overall	1.80 (2)	2.47 (3)	1.73 (1)	1.80 (2)	2.53 (3)	1.67 (1)
Friedman	$p = 0.0808$			$p = 0.0321$		
Pairwise	GC	BS	BSP	GC	BS	BSP
GC	-	0.16	0.98	-	0.11	0.93
BS	0.16	-	0.11	0.11	-	0.05
BSP	0.98	0.11	-	0.92	0.05	-

Table 6.20: Average ranks by dataset, on the cover redundancy of the algorithms' top-k results, shown on the left, and over the entire sets of results, on the right. Two Friedman tests show that the difference in performance on the top-k based measure is significant at the $\alpha = 0.10$ level, and at the $\alpha = 0.05$ level for the entire sets of results.

Furthermore, such a small, but measurable, change in the subgroup being covered would likely introduce some minor variation in the subgroups' quality, when the description is evaluated on either the training or the evaluation data. Consider, then, the behavior of a set of such minor variations of a small number of high quality subgroup descriptions. Although all these subgroups may be expected to have a relatively high quality on both datasets, the descriptions' *ordering* is expected to have a high variance when evaluated on different datasets. That is, some of the subgroups may have a higher quality than others when evaluated on the training data, but the converse may hold on the evaluation data.

Consider also that this variability of the ordering is expected to be much lower for descriptions that do not have the additional, superfluous, conditions added. Using this observation, we propose to compute Kendall's τ rank correlation coefficient between the ordering of a set of results evaluated on the training data, and the ordering of those same results on the evaluation fold. A high value on this measure then indicates that the results' relative ordering is fairly 'stable', that is, that it generalizes well between datasets. Contrariwise, a low value on this measure indicates, regardless of the actual absolute quality of the results, that it is unclear which of the returned results is 'better'. This, in turn, indicates that there must be some 'noise' in the subgroups' descriptions that interferes with one's ability to properly evaluate the results' relative quality.

Alternatively, of course, this variability of the ordering may be attributed merely to variance in the underlying data, in which case we would not expect to observe a significant difference in this measure when comparing multiple algorithms that, over their entire set of results, have a similar performance in terms of the quality of the subgroups they return.

Computing Kendall's τ in this way for each of the cross-validation folds, ordering the algorithms by their performance on this measure, and aggregating the results in the familiar fashion, we show the average ranks obtained in Table 6.21.

We observe that BS is, on average, ranked lower than both GRADCAT-EMM and BSP. Furthermore, BSP performs slightly better than GRADCAT-EMM. A Friedman test shows that this difference in ranks is significant with $p \approx 3 \times 10^{-6}$. Following up with a *post-hoc* test, we find that BS performs worse than GRADCAT-EMM and BSP with $p \approx 0.002$ and $p \approx 0.0005$, respectively. No statistically significant difference is detected between the other two algorithms.

We thus find that the ordering of the results returned by BS is indeed significantly less 'stable' than that of the other algorithms. Combined with this algorithm performing the worst, albeit not significantly so, on both of the CR measures, we conclude that the BS algorithm very likely

AVERAGE RANKS	KENDALL'S τ, TRAIN VS. TEST DATA		
	GC	BS	BSP
EAEF 1	2.10 (2)	2.90 (3)	1.00 (1)
Wine 1	1.35 (1)	2.90 (3)	1.75 (2)
ExtraMarital 1	2.25 (2)	2.50 (3)	1.25 (1)
ExtraMarital 2	1.60 (1)	2.70 (3)	1.70 (2)
Contraceptives 1	1.10 (1)	2.45 (2.5)	2.45 (2.5)
Adult 1	1.15 (1)	2.85 (3)	2.00 (2)
Credit 1	1.00 (1)	2.50 (2.5)	2.50 (2.5)
Credit 2	2.40 (2)	2.55 (3)	1.05 (1)
Contraceptives 2	2.45 (2)	2.55 (3)	1.00 (1)
EAEF 2	1.60 (1.5)	2.80 (3)	1.60 (1.5)
Wine 2	1.00 (1)	2.90 (3)	2.10 (2)
ExtraMarital 3	2.85 (3)	1.95 (2)	1.20 (1)
Adult 2	2.20 (2)	2.80 (3)	1.00 (1)
Credit 3	1.90 (2)	2.80 (3)	1.30 (1)
Contraceptives 3	1.60 (2)	2.90 (3)	1.50 (1)
Overall	1.63 (2)	2.87 (3)	1.50 (1)
Friedman	$p \approx 3 \times 10^{-6}$		
Pairwise	GC	BS	BSP
GC	-	2×10^{-3}	0.93
BS	2×10^{-3}	-	5×10^{-4}
BSP	0.93	5×10^{-4}	-

Table 6.21: Average ranks of the algorithms on the τ rank correlation coefficient, computed between the algorithms' subgroup sets' rankings on the training and evaluation data. A Friedman test shows that there is a significant difference in the algorithms' performance with $p \approx 3 \times 10^{-6}$.

indeed adds superfluous conditions to its results' descriptions. This, in turn, would interfere with the '$\varphi_{\mathcal{D}}(\cdot)$ top-k' measure's ability to ensure the top-k results' uniqueness. From here we can also explain BS's high scores on this quality measure, because the 'same', highest quality, subgroup would be counted multiple times in its computation. Together with the fact that the different algorithms do not, over the entire set of results, differ significantly in the quality of their results, we may finally attribute BS's high quality on the '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' measure to this tendency to saturate its beam with multiple minor variations of its highest quality subgroups.

Ideally, then, of course, we would like to refine our experimental setup to compensate for this behavior, which as stated we consider undesirable. Clearly, there is a large variety of ways in which one might go about doing this, but we here take a fairly straightforward approach.

Essentially, what we do is the following. We re-run all the experiments as before, but slightly adapt the quality measure used for evaluation. Specifically, in the evaluation of the individual results' qualities, we divide the original score $\varphi_{\mathcal{D}}(\cdot)$ by the number of conditions in the result's description. In this way, we seek to penalize additional conditions that do not substantially change the quality of the subgroup being covered. Furthermore, by only performing this penalization during evaluation, it does not interfere with the algorithms' search strategies. On the other hand, since we still optimize the algorithms' parameters during the cross-validation, and because this penalization is taken into account here, the algorithms can still adapt to how this modification changes the final evaluation function.

The results of these experiments, on the compensated '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' measure, are shown in Table 6.22. Here, we see that BSP is ranked the best, with GRADCAT-EMM performing slightly worse, and BS ranking worst out of the different algorithms. However, we find that this difference is only considered significant at the $\alpha = 0.10$ level. We show the corresponding pairwise p-values for reference, but the Nemenyi *post-hoc* test fails to attribute this difference in performance to any of the algorithms.

When we consider the rankings obtained on the two individual $\varphi_{\mathcal{D}}(\cdot)$-based measures, we do find a significant difference in performance between the algorithms. These results are shown in Table 6.23. Considering first the top-k based measure, we find that BS still ranks highest, with GRADCAT-EMM being on average ranked in second place, but only marginally better than BSP. We observe that this difference is significant with $p = 0.0119$. The *post-hoc* test shows BS to outperform GRADCAT-EMM and BSP with $p = 0.046$ and $p = 0.028$, respectively. No significant difference is detected between the performance of GRADCAT-EMM and BSP.

AVERAGE RANKS	$\varphi_{\mathcal{D}}$ H. MEAN		
	GC	BS	BSP
EAEF 1	1.00 (1)	2.00 (2)	3.00 (3)
Wine 1	2.55 (3)	1.75 (2)	1.70 (1)
ExtraMarital 1	1.00 (1)	2.00 (2)	3.00 (3)
ExtraMarital 2	1.35 (1)	3.00 (3)	1.65 (2)
Contraceptives 1	1.00 (1)	3.00 (3)	2.00 (2)
Adult 1	3.00 (3)	2.00 (2)	1.00 (1)
Credit 1	1.50 (1)	2.90 (3)	1.60 (2)
Credit 2	1.00 (1)	2.30 (2)	2.70 (3)
Contraceptives 2	2.00 (2)	3.00 (3)	1.00 (1)
EAEF 2	1.40 (1)	3.00 (3)	1.60 (2)
Wine 2	3.00 (3)	1.75 (2)	1.25 (1)
ExtraMarital 3	1.95 (2)	3.00 (3)	1.05 (1)
Adult 2	3.00 (3)	1.10 (1)	1.90 (2)
Credit 3	2.00 (2)	3.00 (3)	1.00 (1)
Contraceptives 3	2.00 (2)	3.00 (3)	1.00 (1)
Overall	1.80 (2)	2.47 (3)	1.73 (1)
Friedman	$p = 0.0808$		
Pairwise	GC	BS	BSP
GC	-	0.16	0.98
BS	0.16	-	0.11
BSP	0.98	0.11	-

Table 6.22: Average ranks of the algorithms on the '$\varphi_{\mathcal{D}}(\cdot)$ harmonic mean' measure, by dataset. Here, the cross-validation was re-run, and the individual subgroups' quality divided by their description length before this aggregate measure was computed, in order to penalize superfluous complexity. A Friedman test shows that there is a significant difference in the algorithms' performance at the $\alpha = 0.10$ level.

AVERAGE RANKS	$\varphi_{\mathcal{D}}$ TOP-k			$\varphi_{\mathcal{D}}$ ALL		
	GC	BS	BSP	GC	BS	BSP
EAEF 1	1.90 (2)	1.10 (1)	3.00 (3)	1.00 (1)	2.10 (2)	2.90 (3)
Wine 1	2.20 (2)	1.55 (1)	2.25 (3)	2.80 (3)	1.40 (1)	1.80 (2)
ExtraMarital 1	2.00 (2)	1.00 (1)	3.00 (3)	1.85 (2)	3.00 (3)	1.15 (1)
ExtraMarital 2	1.65 (2)	1.35 (1)	3.00 (3)	2.00 (2)	3.00 (3)	1.00 (1)
Contraceptives 1	1.20 (1)	1.90 (2)	2.90 (3)	1.00 (1)	3.00 (3)	2.00 (2)
Adult 1	3.00 (3)	1.00 (1)	2.00 (2)	3.00 (3)	2.00 (2)	1.00 (1)
Credit 1	2.20 (3)	1.80 (1)	2.00 (2)	1.55 (1.5)	2.90 (3)	1.55 (1.5)
Credit 2	1.00 (1)	2.00 (2)	3.00 (3)	1.00 (1)	2.45 (2)	2.55 (3)
Contraceptives 2	3.00 (3)	1.30 (1)	1.70 (2)	2.00 (2)	3.00 (3)	1.00 (1)
EAEF 2	1.65 (2)	3.00 (3)	1.35 (1)	1.10 (1)	3.00 (3)	1.90 (2)
Wine 2	1.80 (2)	2.55 (3)	1.65 (1)	3.00 (3)	1.75 (2)	1.25 (1)
ExtraMarital 3	2.85 (3)	1.15 (1)	2.00 (2)	1.95 (2)	3.00 (3)	1.05 (1)
Adult 2	2.30 (2)	1.10 (1)	2.60 (3)	3.00 (3)	2.00 (2)	1.00 (1)
Credit 3	2.90 (3)	1.25 (1)	1.85 (2)	1.60 (2)	3.00 (3)	1.40 (1)
Contraceptives 3	3.00 (3)	1.20 (1)	1.80 (2)	2.00 (2)	3.00 (3)	1.00 (1)
Overall	2.27 (2)	1.40 (1)	2.33 (3)	1.97 (2)	2.53 (3)	1.50 (1)
Friedman	$p = 0.0119$			$p = 0.0117$		
Pairwise	GC	BS	BSP	GC	BS	BSP
GC	-	0.05	0.98	-	0.27	0.41
BS	0.05	-	0.03	0.27	-	0.01
BSP	0.98	0.03	-	0.41	0.01	-

Table 6.23: Average ranks of the algorithms on the '$\varphi_{\mathcal{D}}(\cdot)$ top-k' and '$\varphi_{\mathcal{D}}(\cdot)$ all' measures, by dataset. Here, the cross-validation was re-run, and the individual subgroups' quality divided by their description length before this aggregate measure was computed, in order to penalize superfluous complexity. Two Friedman tests show that there is a significant difference in the algorithms' performance on the top-k based score, with $p = 0.0119$, as well as on the average over all results, with $p = 0.0117$.

Looking at the average qualities over the entire sets of results, the corresponding rankings of which are shown in the right-hand column of Table 6.23, we see that BSP is here the best performing algorithm. BS here performs worst, with GRADCAT-EMM ranking second on average. This difference is significant with $p = 0.0117$, and BSP outperforms BS with $p = 0.01$. However, no significant difference is detected for the remaining pairings of the algorithms.

In closing, it is interesting to observe that BS still performs significantly better on the top-k results, when using this adapted measure. On first sight, this may seem to indicate that this adaptation of the measure failed to limit BS's tendency to add 'noise' conditions to its descriptions. However, this appears to not be the case when we consider the two CR measures, or the performance on Kendall's τ. The corresponding rankings are shown in Table 6.24 and Table 6.25, respectively. Here, we see that all algorithms on average perform roughly on par, and we can no longer detect a significant difference in the algorithms' performance on these measures.

AVERAGE RANKS	CR TOP-k			CR ALL		
	GC	BS	BSP	GC	BS	BSP
EAEF 1	2.35 (3)	1.45 (1)	2.20 (2)	2.45 (2)	1.05 (1)	2.50 (3)
Wine 1	1.00 (1)	2.20 (2)	2.80 (3)	1.00 (1)	2.15 (2)	2.85 (3)
ExtraMarital 1	1.10 (1)	2.85 (3)	2.05 (2)	1.00 (1)	2.50 (2.5)	2.50 (2.5)
ExtraMarital 2	1.50 (1)	2.90 (3)	1.60 (2)	1.00 (1)	2.80 (3)	2.20 (2)
Contraceptives 1	1.85 (2)	2.80 (3)	1.35 (1)	2.40 (3)	2.30 (2)	1.30 (1)
Adult 1	2.10 (3)	1.95 (1.5)	1.95 (1.5)	2.60 (3)	1.00 (1)	2.40 (2)
Credit 1	2.60 (3)	2.00 (2)	1.40 (1)	2.30 (3)	1.80 (1)	1.90 (3)
Credit 2	1.70 (2)	1.40 (1)	2.90 (3)	1.75 (2)	1.25 (1)	3.00 (3)
Contraceptives 2	2.25 (2)	2.75 (3)	1.00 (1)	3.00 (3)	2.00 (2)	1.00 (1)
EAEF 2	2.40 (3)	2.20 (2)	1.40 (1)	3.00 (3)	1.00 (1)	2.00 (2)
Wine 2	1.00 (1)	2.80 (3)	2.20 (2)	1.00 (1)	2.75 (3)	2.25 (2)
ExtraMarital 3	2.15 (2.5)	2.15 (2.5)	1.70 (1)	2.15 (2)	1.55 (1)	2.30 (3)
Adult 2	1.85 (2)	1.30 (1)	2.85 (3)	2.90 (3)	1.00 (1)	2.10 (2)
Credit 3	2.60 (3)	2.40 (2)	1.00 (1)	2.95 (3)	2.00 (2)	1.05 (1)
Contraceptives 3	2.50 (2.5)	2.50 (2.5)	1.00 (1)	3.00 (3)	1.90 (2)	1.10 (1)
Overall	2.13 (2)	2.17 (3)	1.70 (1)	2.27 (3)	1.70 (1)	2.03 (2)
Friedman	$p = 0.3548$			$p = 0.2996$		

Table 6.24: Average ranks by dataset, on the cover redundancy of the algorithms' top-k results, shown on the left, and over the entire sets of results, on the right. Here, the cross-validation was re-run, and the individual subgroups' quality divided by their description length during the optimization of the algorithms' parameters, in order to penalize superfluous complexity. Performing a Friedman test on each of these results fails to detect a significant difference on either.

AVERAGE RANKS	AVG KENDALL'S τ, TRAIN VS. TEST DATA		
	GC	BS	BSP
EAEF 1	1.60 (1)	2.30 (3)	2.10 (2)
Wine 1	2.10 (2)	1.35 (1)	2.55 (3)
ExtraMarital 1	2.20 (2)	2.45 (3)	1.35 (1)
ExtraMarital 2	1.80 (1)	2.15 (3)	2.05 (2)
Contraceptives 1	1.40 (1)	1.80 (2)	2.80 (3)
Adult 1	1.60 (1)	1.95 (2)	2.45 (3)
Credit 1	1.30 (1)	2.00 (2)	2.70 (3)
Credit 2	1.85 (2)	1.50 (1)	2.65 (3)
Contraceptives 2	2.70 (3)	2.30 (2)	1.00 (1)
EAEF 2	2.05 (2)	1.00 (1)	2.95 (3)
Wine 2	2.60 (3)	1.20 (1)	2.20 (2)
ExtraMarital 3	2.40 (2)	1.00 (1)	2.60 (3)
Adult 2	3.00 (3)	2.00 (2)	1.00 (1)
Credit 3	2.10 (2)	2.60 (3)	1.30 (1)
Contraceptives 3	2.60 (3)	2.30 (2)	1.10 (1)
Overall	1.93 (1.5)	1.93 (1.5)	2.13 (3)
Friedman	$p = 0.8287$		

Table 6.25: Average ranks of the algorithms on the τ rank correlation coefficient, computed between the algorithms' subgroup sets' rankings on the training and evaluation data. Here, the cross-validation was re-run, and the individual subgroups' quality divided by their description length during the optimization of the algorithms' parameters, in order to penalize superfluous complexity. Performing a Friedman test on these results fails to detect a significant difference between the performance of the algorithms.

6.3 ILLUSTRATIVE RESULTS

In this section, we will show some illustrative results obtained with GRADCAT-EMM on the real world datasets used in Section 6.2.

To obtain these results on a given dataset, the complete dataset was split into a training and evaluation dataset of equal size. The algorithm was then run, and 100 results were obtained on the training data. These results were subsequently evaluated and ranked on the evaluation dataset. In this section, then, we discuss some illustrative subgroups from among these results.

6.3.1 *EAEF Data*

The Educational Attainment and Earnings Functions (EAEF) dataset consists of a subset of the data from the U.S. National Longitudinal Survey of Youth 1979. It contains survey data from a nationally representative sample of the population, on such variables as hourly earnings, education, and socioeconomic background. A detailed description is given in Appendix B of [10]. This dataset was also used in the experiments of EMM with linear regression models performed by Duivesteijn *et al.* [12]. In our experiments, we have used the same model as described in their work, which we here refer to as the 'EAEF 1' dataset.

The corresponding model induced on the global population is

$$\text{Earnings} = -30.01 + 2.93 \times \text{YearsSchool} + 0.57 \times \text{YearsExperience}.$$

Do note that these exact values of the coefficients were obtained on a random sub-sampling of the data, so there may be some minor differences with work described elsewhere. Regardless, preliminary analysis on the entire dataset found that all these explanatory variables are significant at the $\alpha = 0.001$ confidence level, and that the model's $R^2 = 0.21$. Furthermore, the coefficients' signs appear to intuitively make sense.

In the experiments performed to find these results, the second highest quality subgroup was described by $(\text{Female} = 1)$, and the subgroup's model given by

$$\text{Earnings} = -17.10 + 1.91 \times \text{YearsSchool} + 0.46 \times \text{YearsExperience}.$$

This subgroup contained 623 records on the evaluation dataset. The change in the subgroup's coefficients indicate that females are expected to only receive an additional \$1.91 in hourly wages for each additional year of schooling, compared to the global population, where an additional \$2.93 is expected per year. We see a similar effect in the years of work experience, for which this group receives \$0.46, compared to \$0.57 in the entire population. However, do note that the intercept is substantially higher for this subgroup, which may compensate for these effects somewhat.

6.3.2 *Extramarital Affair Data*

The 'Extramarital' dataset was obtained from [14]. It contains survey data from 1977 pertaining to extramarital affairs. In our experiments described in Section 6.2, we used as a target variable a constructed measure in this dataset that quantifies the survey's participants' involvement in extramarital affairs. However, other models may clearly also be induced on these data. Here, we consider the model that on the global population is given by

$$\text{RatingOfMarriage} = 4.25 - 0.09 \times \text{NumberOfChildren}.$$

The variable RatingOfMarriage is the survey participant's self-reported rating of marriage happiness. This variable takes values from 1 to 5, with higher scores being better. Analysis on the entire dataset shows that the coefficients are significant at the $\alpha = 0.001$ level, and the model's $R^2 \approx 0.02$. We leave it to the reader to interpret whether the coefficients' signs make intuitive sense.

The highest quality subgroup that we found on this dataset contained 406 records, and was described by

$$\begin{aligned}&\big(\text{YearsMarried} \leq 4.25\big) \wedge \\ &\big(\text{HusbandOccupation} \in \{\text{SemiSkilled}, \text{Unskilled}, \text{WhiteCollar}\}\big).\end{aligned}$$

The corresponding coefficients of the model are

$$\text{RatingOfMarriage} = 4.35 - 0.37 \times \text{NumberOfChildren}.$$

This result would seem to indicate that, for recently wed couples in relatively low to average socioeconomic positions, marriage happiness is more strongly negatively influenced by how many children they have, as compared to the entire population. However, we do not propose a reason for why this effect may be observed.

6.3.3 *Credit Scoring Data*

The 'Credit' dataset was obtained from [14], and it was analyzed in [13]. It contains processed information about loan defaulting behavior of the clients of an anonymous major credit card company. The default task is to predict whether or not a client will default on a loan.

Here, we fitted the global model

$$\text{Default} = 0.11 - 0.63 \times \text{Spending}.$$

The variable Spending is the average monthly credit card expenditure divided by 10,000. The analysis was performed only on records corresponding to credit card holders. Analysis on the main dataset shows that the coefficients are significant at $\alpha = 0.001$. The model's $R^2 \approx 0.002$.

The highest quality subgroup that we found on this dataset contained 223 records, and it was described by $(\text{Age} \leq 21.5)$. The corresponding model was

$$\text{Default} = 0.19 + 0.59 \times \text{Spending}.$$

Note the sign-change of the coefficient for this subgroup. In an attempt to explain this effect, we note that younger people might typically have a rather lower income than people who have had more time to establish a professional career. Furthermore, they might also typically have access to less savings. Intuitively, then, this would seem to imply that an increase in credit-based spending may indeed raise the probability of having to default on the loan.

7

CONCLUSIONS

In this work, we have considered how to improve on the performance of the beam search algorithm as a search heuristic for solving the Exceptional Model Mining (EMM) problem for linear regression models.

We have proposed a novel algorithm that exploits information about individual records' influence on the subgroup's model, and hence on the subgroup's exceptionality, in order to obtain such improved performance. The method is based on the idea of generalizing, during the search process, the subgroup being optimized to a *soft* subgroup. By then rewriting the quality of a subgroup as a differentiable objective function, we could compute such an individual record's influence on the subgroup's quality by differentiating the objective function with respect to that record's degree of inclusion in the subgroup.

The search strategy that we use, is a novel combination of classification trees and numerical optimization, which may be interpreted as a form of constrained gradient ascent. Under this interpretation, the algorithm tries to find the *weakest* possible constraint on the objective function, that still allows it to express the gradient ascent's optimum in a rule-based format. We have referred to this search strategy as GRADCAT, for Gradient Ascent with Differentiable ClAssition Trees.

By some qualitative comparisons, and a numeric example, we have shown that there exist problems that this algorithm can solve, and for which neither beam search nor a more 'traditional' tree-based approach is able to find the optimal solution.

The algorithm furthermore incorporates a post-processing procedure to ensure that the resulting subgroup description is as succinct as possible. This is done both as an application of Occam's razor, and to make it easier to interpret the results for an end-user. We have also shown that this post-processing procedure is directly applicable to the results returned by beam search. This may help with the latter algorithm's tendency to include a large number of minor variations of a small set of high quality subgroups in its set of results.

We have referred to the novel algorithm proper as GRADCAT-EMM, indicating the use of the GRADCAT search strategy and the domain-specific auxiliary restart and post-processing procedures.

By showing that the EMM problem with linear regression models is a strict generalization of traditional Subgroup Discovery (SD), i.e., that the latter is a special case of the former, we have shown that the newly

proposed algorithm is at least also immediately applicable to the general problem of SD.

7.1 EXPERIMENTS

Experiments were performed to compare the difference in performance between the GRADCAT-EMM algorithm and beam search (BS). A composite algorithm applying the post-processing procedure to the BS algorithm's results was also included in this comparison, which we referred to as 'beam search with pruning' (BSP). Experiments were performed both on synthetic data, and on real worlds datasets taken from a selection of different domains.

On the synthetic data, we found that GRADCAT-EMM consistently statistically significantly outperformed BS and BSP on both supervised and unsupervised evaluation measures. Furthermore, the BSP algorithm also significantly outperformed BS on the majority of these experiments.

On randomly repeated and cross-validated experiments on real world datasets, we found the opposite effect. That is, here BS statistically significantly outperformed both GRADCAT-EMM and BSP on the previously used unsupervised evaluation measure. Here, we found also that there was no statistically significant difference in the performance of GRADCAT-EMM and BSP. After further analysis of these results, we concluded that BS was likely to score best on these experiments, because it was able to exploit a property of the evaluation measure that caused it to assign a high quality to sets of results that we would subjectively find undesirable.

Particularly, we concluded that this was an effect of the BS algorithm's tendency to saturate its beam with minor variations of what effectively amount to the same high quality subgroup. Somewhat more specifically, that the algorithm does this by appending largely non-informational conditions to its subgroups' descriptions. This effect was not observed in our earlier experiments on synthetic data, where we assessed the evaluation method's merit. This was due to the fact that, there, BS was explicitly prevented from adding such additional non-informational conditions, by letting it search exactly to the depth at which the correct subgroups in the data were to be found.

We therefore proposed an adaption to the evaluation measure used, that was intended to penalize adding conditions to a subgroup description without substantially changing the subgroup being covered. Repeating the cross-validated experiments using this adapted measure, we found that BSP generally ranked highest on average on most of the various methods of assessment that we used. However, in this collection of experiments, we largely failed to detect statistically significant differences between the algorithms' performance. Of note is that we also no

longer found a difference in performance on the measures that we used to detect this 'adding of non-informational conditions'.

It is furthermore interesting to observe the stark difference in the relative performance of the algorithms, between the experiments on synthetic and real world datasets. We may use the observation that BSP is essentially a composite algorithm combining BS's method of finding descriptions, with GRADCAT-EMM's method of post-processing these descriptions, to make some conclusions about when to use which algorithm.

Specifically, we failed to detect a significant difference in performance between BSP and GRADCAT-EMM in any of the experiments on the real world datasets. Here, on the first set of experiments, BS outperformed the other two, but we concluded that this was mostly due to it being able to construct superfluously complex subgroup descriptions. Note that this is *precisely* what the post-processing in BSP is trying to remedy. When we used an adapted evaluation measure to penalize this behavior, we no longer found any significant difference between the three algorithms, in the extent to which they displayed this behavior.

We may conclude, then, that on the real world datasets, after controlling for the algorithms' different ways of limiting the complexity of their results, we find no significant difference in performance between the beam search and GRADCAT strategies of searching for (not yet post-processed) subgroups.

Contrariwise, in the experiments with synthetic data, the different methods of limiting the results' complexity were already taken into account by the experimental setup used. Furthermore, we there found that the GRADCAT strategy of searching for subgroups outperformed the beam search strategy, by virtue of these different strategies being the only difference between GRADCAT-EMM and BSP.

Therefore, in summary, we may conclude that the synthetic datasets have certain characteristics on which GRADCAT-EMM may be expected to perform better than BS-based methods. On the other hand, on datasets whose properties are more closely reflected by the real world datasets that we used, we do not conclude on a performance-wise preference for either GRADCAT-EMM or BSP. For the user familiar with BS, however, BSP may then be the more natural choice. In any case, it seems clear that either would be preferred over the 'standard' BS algorithm, given that the latter both does not control for unneeded complexity in its results, and is not as of yet expected to outperform either based on a dataset's specific characteristics.

In an informal attempt to intuitively characterize the difference between the types of datasets, we first note that the synthetic datasets contained subgroups which were known to have very high quality. That is, these subgroups' models deviated more from the global model, than

anything we found on the real world data. On the one hand, this would not seem like a desirable prerequisite to have for recommending the GRADCAT-EMM algorithm. However, it is of note that the BS-based algorithms specifically had difficulties with these data. That is, if the subgroups are known to have such high quality, it would be nice to be able to retrieve them consistently.

The most obvious other difference between the characteristics of the datasets pertains to the descriptive attributes used for describing subgroups. That is, the synthetic data had a relatively large number of these attributes, and furthermore, most of these attributes contained only 'noise'. That is, they were sampled from a uniform distribution, and by construction therefore did not carry information about the objective function. Contrariwise, the real world datasets had in comparison relatively few descriptive attributes. Furthermore, we note that all of these datasets were obtained from sources that, in general, had specifically preprocessed the datasets for statistical analysis. In other words, we may expect the few attributes that *were* included, to actually carry useful information about the objective function.

From the above, we hypothesize that GRADCAT-EMM is expected to perform particularly well, in comparison to BS-based methods, on datasets where high quality subgroups can be found in comparatively large search spaces that have a very low signal-to-noise ratio.

7.2 FUTURE WORK

In future work, it may be interesting to find datasets that have a large number of descriptive attributes with a very low signal-to-noise ratio, to try to confirm the hypothesis that GRADCAT-EMM may be expected to perform particularly well on such datasets.

Furthermore, we note that the gradient ascent algorithm used in the GRADCAT strategy is a fairly naive method of numerical optimization. It would thus seem interesting to try to establish whether performance might be improved by using a more sophisticated approach. An obvious first step might be to extend the algorithm with a dynamic step-size, for example using a line-search algorithm.

Finally, we note that, although GRADCAT-EMM was developed for use with linear regression models, and is as such already directly applicable to the general SD problem, it may be interesting to see if we can use it for different model types. We note that both the GRADCAT strategy and the post-processing procedure are largely agnostic to the specific type of model that is used. That is, the latter necessitates that one can formulate a confidence interval for the model, whereas the former only requires that the estimation of the model's parameters can be made differentiable by stipulating the use of soft subgroups.

A

REDUCTION OF GRADCAT TO CART

In this section, we show how the GRADCAT algorithm described in Section 4.3.1 can, under some mild assumptions, be reduced to the CART algorithm for solving the binary classification problem. Specifically, we will show that the GRADCAT algorithm can be made to approximate the output of the CART algorithm to an arbitrary precision.

With respect to the assumptions made, we will assume that the step size η of the gradient ascent algorithm is small enough to guarantee convergence, and similarly, that the maximum number of iterations $\tau_{\max}$ is large enough to ensure that the algorithm does not stop prematurely. Also, we enforce that the minimum threshold of the gradient $\nabla_{\min}$, which is used for checking convergence, is small enough to ensure that we only stop the algorithm after it has converged on the output of the CART algorithm to the desired precision.

As per the specification of the binary classification problem discussed in Section 2.4.1, we assume that we are given a dataset containing n records r_i, $1 \leq i \leq n$, of the form

$$r_i \equiv \langle a_1^i, \ldots, a_k^i, t^i \rangle,$$

where we refer to the various a_j^i as the record's attributes, and $t^i \in \{0, 1\}$ is referred to as the record's target.

The objective function that we use for the GRADCAT algorithm is the log-likelihood function $\mathcal{L} : [0, 1]^n \to \mathbb{R}$, given by

$$\mathcal{L}(o_1, \ldots, o_n) \equiv \sum_{i=1}^{n} t^i \ln o_i + (1 - t^i) \ln(1 - o_i).$$

To show the reduction to CART, we need to prove two facts; firstly, that the splits performed by the GRADCAT algorithm are the same as those performed by the CART algorithm, and secondly, that the predictions for all records converge to the same values. We start by showing that both algorithms always perform the same splits.

Consider that for both algorithms, the quality of a split in a given leaf strongly depends on which records are assigned to that leaf. Thus, to show equivalence of the performed splits, we need to show that the same records are always assigned to the same leafs. The proof for this follows directly from a trivial induction argument; initially, all records are always assigned to the root of the tree; if, in the root, the same split is performed in both algorithms, the same records are assigned to the resulting children, and so forth. Thus, we only need to show that, if

the same records are assigned to a leaf, both algorithms will perform the same split in that leaf. Sequential splitting of the resulting leafs will then ensure that the records always continue to end up in the same leafs of the tree.

Thus, let R_{V_j} denote the set of records assigned to a leaf V_j. For the CART algorithm, the impurity $\phi^{\text{CART}}(\cdot)$ of V_j is computed using the Gini-index, which is to say, as

$$\phi^{\text{CART}}(V_j) = \bar{t}_j(1 - \bar{t}_j),$$

where $\bar{t}_j$ is the mean of the target values of the records assigned to V_j.

The GRADCAT algorithm, on the other hand, uses the signs of the derivatives for the records assigned to V_j. For all $i \in R_{V_j}$, we find after simplification

$$\frac{\partial \mathcal{L}(\cdot)}{\partial o_i} = \frac{t^i}{o_i} - \frac{1 - t^i}{1 - o_i}.$$

We consider when these derivatives have which signs. Assuming that the partial derivative is positive, we have after simplification

$$\begin{aligned} \frac{t^i}{o_i} - \frac{1 - t^i}{1 - o_i} &> 0 \\ t^i &> o_i. \end{aligned}$$

From $t^i > o_i$, $t^i \in \{0, 1\}$, and $o_i \in [0, 1]$, we find $t^i = 1$ and $o_i < 1$. In other words, if the partial derivative for the i-th record is positive, we know $t^i = 1$. Furthermore, note that this result holds for any choice $o_i < 1$, meaning that this holds irrespective of the current output of the GRADCAT model, and hence, irrespective of the current iteration of the gradient ascent algorithm. Similarly, if we assume that the derivative is negative, we have

$$\begin{aligned} \frac{t^i}{o_i} - \frac{1 - t^i}{1 - o_i} &< 0 \\ t^i &< o_i, \end{aligned}$$

and using the same argument, we know $t^i = 0$ and $o_i > 0$. Finally, if we assume that the derivative is zero, we have

$$\begin{aligned} \frac{t^i}{o_i} - \frac{1 - t^i}{1 - o_i} &= 0 \\ t^i &= o_i, \end{aligned}$$

but from the fact that $t^i \in \{0, 1\}$ and $o_i = \sigma(v^j)$, we know that this cannot happen. That is, $\sigma(v^j) \in \{0, 1\}$ if and only if $v^j \in \{-\infty, \infty\}$, which is impossible from $v^j \in \mathbb{R}$. Thus, by contradiction, $\frac{\partial \mathcal{L}(\cdot)}{\partial o_i} \neq 0$.

To compute the impurity of V_j, the GRADCAT algorithm uses the relative frequency of the signs of the partial derivatives of the records assigned to V_j. For the records with a positive derivative, this relative frequency is denoted $\varrho^+_{V_j,\tau}$, and given by

$$\varrho^+_{V_j,\tau} = \frac{|R^+_{V_j,\tau}|}{|R_{V_j}|}.$$

Here, $R^+_{V_j,\tau}$ denotes the set of indices of records assigned to V_j, which have a positive derivative at time τ.

From our previous results, we have that the i-th record has a positive derivative if and only if $t^i = 1$. It follows that

$$|R^+_{V_j,\tau}| = \sum_{i \in R_{V_j}} t^i,$$

from which we have

$$\varrho^+_{V_j,\tau} = \bar{t}_j.$$

Using the same argument, we find for the relative frequency of records with negative derivative

$$\varrho^-_{V_j,\tau} = 1 - \bar{t}_j.$$

Finally, we know

$$\varrho^0_{V_j,\tau} = 0,$$

from the fact that no record can have a partial derivative of zero.

The impurity measure used by the GRADCAT algorithm is the multi-class generalization of the Gini-index, given by

$$\begin{aligned}\phi^{\text{GRADCAT}}(V_j,\tau) \equiv\ & \varrho^+_{V_j,\tau-1} \cdot \left(1 - \varrho^+_{V_j,\tau-1}\right) \\ & + \varrho^-_{V_j,\tau-1} \cdot \left(1 - \varrho^-_{V_j,\tau-1}\right) \\ & + \varrho^0_{V_j,\tau-1} \cdot \left(1 - \varrho^0_{V_j,\tau-1}\right).\end{aligned}$$

Substituting in our earlier results, we have, for any choice of τ,

$$\begin{aligned}\phi^{\text{GRADCAT}}(V_j,\tau) &= \bar{t}_j(1-\bar{t}_j) + (1-\bar{t}_j)(1-(1-\bar{t}_j)) + 0 \cdot (1-0) \\ &= \bar{t}_j(1-\bar{t}_j) + (1-\bar{t}_j)\bar{t}_j \\ &= 2 \cdot \bar{t}_j(1-\bar{t}_j).\end{aligned}$$

Thus, we find, for any choice of τ,

$$\phi^{\text{GRADCAT}}(V_j,\tau) = 2 \cdot \phi^{\text{CART}}(V_j).$$

Finally, consider that both the CART and GRADCAT algorithm use the same form of split quality function. Denote these functions as $\Delta_\phi^{\text{CART}}(\cdot)$ and $\Delta_\phi^{\text{GRADCAT}}(\cdot)$, respectively. From our earlier results, it follows that

$$\Delta_\phi^{\text{GRADCAT}}(V_j, d^j(\cdot), \tau) = 2 \cdot \Delta_\phi^{\text{CART}}(V_j, d^j(\cdot)),$$

for any choice of τ and split function $d^j(\cdot) \in \mathcal{S}$. It follows that the optimal split function is the same for both algorithms, that is

$$\arg\max_{d(\cdot)\in\mathcal{S}} \Delta_\phi^{\text{GRADCAT}}(V_j, d^j(\cdot), \tau) = \arg\max_{d(\cdot)\in\mathcal{S}} \Delta_\phi^{\text{CART}}(V_j, d^j(\cdot)),$$

for any choice of τ. This concludes the proof that both algorithms will always perform the same split for each leaf.

Now, assume that a given leaf V_j can no longer be split, either because it is fully pure, or because there are no valid splits to perform. Using the CART algorithm, the output of this leaf is given by

$$o^j = \bar{t}_j.$$

The GRADCAT algorithm, on the other hand, will update the output $o^j = \sigma(v^j)$ of V_j until convergence. Clearly, this happens when the partial derivative of the objective function with respect to v^j goes to zero. The partial derivative is given by

$$\begin{aligned}
\frac{\partial \mathcal{L}(\cdot)}{\partial v^j} &= \frac{\partial \mathcal{L}}{\partial \sigma(v^j)} \frac{\partial \sigma(v^j)}{\partial v^j} \\
&= \sum_{i \in R_{V_j}} \frac{\partial \mathcal{L}}{\partial o_i} \frac{\partial \sigma(v^j)}{\partial v^j} \\
&= \sum_{i \in R_{V_j}} \left(t^i - \sigma(v^j) \right).
\end{aligned}$$

As this derivative goes to zero, we have

$$\begin{aligned}
\frac{\partial \mathcal{L}(\cdot)}{\partial v^j} &\to 0 \\
\sum_{i \in R_{V_j}} \left(t^i - \sigma(v^j) \right) &\to 0 \\
\sum_{i \in R_{V_j}} \sigma(v^j) &\to \sum_{i \in R_{V_j}} t^i \\
\sigma(v^j) &\to \frac{1}{|R_{V_j}|} \cdot \sum_{i \in R_{V_j}} t^i \\
\sigma(v^j) &\to \bar{t}_j.
\end{aligned}$$

Thus, the output of a leaf in the GRADCAT algorithm will converge to the output of a leaf in the CART algorithm.

Hence, by the fact that the same splits are performed during the growing of the tree, the outputs of the GRADCAT model will converge, for all records, to the outputs of the CART model.

■

B

DERIVATIVE OF OBJECTIVE FUNCTION

In this section, we give the derivation of the derivative of the objective function $O : [0,1]^n \to \mathbb{R}$ given by

$$O(w_1, \ldots, w_n) \equiv \frac{\mathrm{Tr}(\mathbf{W})}{n} \cdot \frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2s^2},$$

where for all $1 \leq i \leq n$, the i-th diagonal element of the diagonal matrix $\mathbf{W}$ is given by w_i.

We show the derivation of the partial derivative of $O(\cdot)$ with respect to w_i. We first note that by application of the product rule for partial derivatives, and the equality $\frac{\partial \mathrm{Tr}(\mathbf{W})}{\partial w_i} = 1$, we have

$$\frac{\partial O(\cdot)}{\partial w_i} = \frac{\partial}{\partial w_i}\left[\frac{\mathrm{Tr}(\mathbf{W})}{n} \cdot \frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2s^2}\right] \tag{B.1}$$

$$= \frac{\partial \mathrm{Tr}(\mathbf{W})}{\partial w_i} \cdot \frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2ns^2} + \frac{\mathrm{Tr}(\mathbf{W})}{ns^2} \cdot \frac{\partial}{\partial w_i}\left[\frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2}\right] \tag{B.2}$$

$$= \frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2ns^2} + \frac{\mathrm{Tr}(\mathbf{W})}{ns^2} \cdot \frac{\partial}{\partial w_i}\left[\frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2}\right]. \tag{B.3}$$

We expand the remaining partial derivative in Equation B.3 separately. Noting first that this derivative depends only on w_i through $\hat{\beta}_G$, we obtain

$$\frac{\partial}{\partial w_i}\left[\frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2}\right] = \frac{\partial}{\partial w_i}\left[\frac{(\hat{\mathbf{y}}_G - \hat{\mathbf{y}})^\top (\hat{\mathbf{y}}_G - \hat{\mathbf{y}})}{2}\right] \tag{B.4}$$

$$= \frac{\partial}{\partial w_i}\left[\frac{(\hat{\beta}_G - \hat{\beta})^\top \mathbf{X}^\top \mathbf{X} (\hat{\beta}_G - \hat{\beta})}{2}\right] \tag{B.5}$$

$$= (\hat{\beta}_G - \hat{\beta})^\top \mathbf{X}^\top \mathbf{X} \frac{\partial \hat{\beta}_G}{\partial w_i}. \tag{B.6}$$

Again expanding the term $\frac{\partial \hat{\beta}_G}{\partial w_i}$ separately, we find

$$\frac{\partial \hat{\beta}_G}{\partial w_i} = \frac{\partial}{\partial w_i}\left[(\mathbf{X}^\top \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{W} \mathbf{y}\right] \tag{B.7}$$

$$= \frac{\partial}{\partial w_i}\left[(\mathbf{X}^\top \mathbf{W} \mathbf{X})^{-1}\right] \mathbf{X}^\top \mathbf{W} \mathbf{y} + (\mathbf{X}^\top \mathbf{W} \mathbf{X})^{-1} \frac{\partial}{\partial w_i}\left[\mathbf{X}^\top \mathbf{W} \mathbf{y}\right], \tag{B.8}$$

by the definition of $\hat{\beta}_G$ and application of the product rule for partial derivatives. For notational convenience, we define an auxiliary $p \times n$ matrix $\mathbf{A}$ such that

$$\mathbf{A} \equiv (\mathbf{X}^\top \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^\top.$$

Making use of the definition of $\mathbf{A}$ and the equality [27] $\frac{\partial \mathbf{M}^{-1}}{\partial x} = -\mathbf{M}^{-1}\frac{\partial \mathbf{M}}{\partial x}\mathbf{M}^{-1}$ for any matrix $\mathbf{M}$, we can simplify Equation B.8 as

$$\begin{aligned}
\frac{\partial \hat{\beta}_G}{\partial w_i} &= \frac{\partial}{\partial w_i}\left[(\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\right]\mathbf{X}^\top \mathbf{W}\mathbf{y} + (\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\frac{\partial}{\partial w_i}\left[\mathbf{X}^\top \mathbf{W}\mathbf{y}\right] \\
&= \frac{\partial}{\partial w_i}\left[(\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\right]\mathbf{X}^\top \mathbf{W}\mathbf{y} + (\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top \frac{\partial \mathbf{W}}{\partial w_i}\mathbf{y} \\
&= \frac{\partial}{\partial w_i}\left[(\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\right]\mathbf{X}^\top \mathbf{W}\mathbf{y} + \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\mathbf{y} \\
&= -(\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\frac{\partial}{\partial w_i}\left[\mathbf{X}^\top \mathbf{W}\mathbf{X}\right](\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{W}\mathbf{y} + \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\mathbf{y} \\
&= -(\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\frac{\partial}{\partial w_i}\left[\mathbf{X}^\top \mathbf{W}\mathbf{X}\right]\mathbf{A}\mathbf{W}\mathbf{y} + \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\mathbf{y} \\
&= -(\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top \frac{\partial \mathbf{W}}{\partial w_i}\mathbf{X}\mathbf{A}\mathbf{W}\mathbf{y} + \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\mathbf{y} \\
&= -\mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\mathbf{X}\mathbf{A}\mathbf{W}\mathbf{y} + \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\mathbf{y} \\
&= \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\mathbf{y} - \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\mathbf{X}\mathbf{A}\mathbf{W}\mathbf{y}.
\end{aligned}$$

We note that $\hat{\beta}_G$ is given by

$$\begin{aligned}
\hat{\beta}_G &= (\mathbf{X}^\top \mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{W}\mathbf{y} \\
&= \mathbf{A}\mathbf{W}\mathbf{y}.
\end{aligned}$$

We thus find

$$\begin{aligned}
\frac{\partial \hat{\beta}_G}{\partial w_i} &= \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\mathbf{y} - \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\mathbf{X}\hat{\beta}_G \\
&= \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\mathbf{y} - \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\hat{\mathbf{y}}_G \\
&= \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}(\mathbf{y} - \hat{\mathbf{y}}_G) \\
&= \mathbf{A}\frac{\partial \mathbf{W}}{\partial w_i}\mathbf{e}_G.
\end{aligned}$$

Note that $\frac{\partial \mathbf{W}}{\partial w_i}$ is an $n \times n$ matrix with all elements zero, except for the i-th diagonal element, which equals one. Denote this matrix as $\mathbf{W}'$, and its diagonal elements as w'_j for all $1 \leq j \leq n$. We have

$$\begin{aligned}
\frac{\partial \hat{\beta}_G}{\partial w_i} &= \mathbf{A}\mathbf{W}'\mathbf{e}_G && \text{(B.9)} \\
&= \sum_{1 \leq j \leq n} \mathbf{A}_{.j} w'_j (\mathbf{e}_G)_j && \text{(B.10)} \\
&= \mathbf{A}_{.i} w'_i (\mathbf{e}_G)_i + \sum_{\substack{1 \leq j \leq n \\ j \neq i}} \mathbf{A}_{.j} w'_j (\mathbf{e}_G)_j && \text{(B.11)} \\
&= \mathbf{A}_{.i} (\mathbf{e}_G)_i, && \text{(B.12)}
\end{aligned}$$

where $\mathbf{A}_{.i}$ is the i-th column of $\mathbf{A}$. Substituting Equation B.12 into Equation B.6 and transposing yields

$$\begin{aligned}\frac{\partial}{\partial w_i}\left[\frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2}\right] &= \left(\hat{\beta}_G - \hat{\beta}\right)^\top \mathbf{X}^\top \mathbf{X} \frac{\partial \hat{\beta}_G}{\partial w_i} \\ &= \left(\hat{\beta}_G - \hat{\beta}\right)^\top \mathbf{X}^\top \mathbf{X} \mathbf{A}_{.i} (\mathbf{e}_G)_i \\ &= (\mathbf{e}_G)_i \left(\left(\mathbf{A}_{.i}\right)^\top \mathbf{X}^\top \mathbf{X} \left(\hat{\beta}_G - \hat{\beta}\right) \right) \\ &= (\mathbf{e}_G)_i \left(\left(\mathbf{A}_{.i}\right)^\top \mathbf{X}^\top \left(\mathbf{X}\hat{\beta}_G - \mathbf{X}\hat{\beta}\right) \right).\end{aligned}$$

Noting that we may take the dependency on i outside, and using the definition of $\hat{\mathbf{y}}_G$, we can simplify this as

$$\begin{aligned}\frac{\partial}{\partial w_i}\left[\frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2}\right] &= (\mathbf{e}_G)_i \left(\left(\mathbf{A}_{.i}\right)^\top \mathbf{X}^\top \left(\hat{\mathbf{y}}_G - \mathbf{X}\hat{\beta}\right) \right) \\ &= (\mathbf{e}_G)_i \left(\mathbf{A}^\top \mathbf{X}^\top \left(\hat{\mathbf{y}}_G - \mathbf{X}\hat{\beta}\right) \right)_i.\end{aligned}$$

Temporarily multiplying through the $\mathbf{X}^\top$, expanding the term $\hat{\beta}$, and multiplying by -1 twice, we find

$$\begin{aligned}\frac{\partial}{\partial w_i}\left[\frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2}\right] &= (\mathbf{e}_G)_i \left(\mathbf{A}^\top \left(\mathbf{X}^\top \hat{\mathbf{y}}_G - \mathbf{X}^\top \mathbf{X} \hat{\beta} \right) \right)_i \\ &= (\mathbf{e}_G)_i \left(\mathbf{A}^\top \left(\mathbf{X}^\top \hat{\mathbf{y}}_G - \mathbf{X}^\top \mathbf{X} (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \right) \right)_i \\ &= (\mathbf{e}_G)_i \left(\mathbf{A}^\top \left(\mathbf{X}^\top \hat{\mathbf{y}}_G - \mathbf{X}^\top \mathbf{y} \right) \right)_i \\ &= (\mathbf{e}_G)_i \left(\mathbf{A}^\top \mathbf{X}^\top \left(\hat{\mathbf{y}}_G - \mathbf{y} \right) \right)_i \\ &= -(\mathbf{e}_G)_i \left(\mathbf{A}^\top \mathbf{X}^\top \left(\mathbf{y} - \hat{\mathbf{y}}_G \right) \right)_i \\ &= -(\mathbf{e}_G)_i \left(\mathbf{A}^\top \mathbf{X}^\top \mathbf{e}_G \right)_i.\end{aligned}$$

We observe that we may take the element $(\mathbf{e}_G)_i$ inside, using the diagonal matrix $\text{diag}(\mathbf{e}_G)$. We obtain

$$\frac{\partial}{\partial w_i}\left[\frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2}\right] = -\left(\text{diag}(\mathbf{e}_G) \mathbf{A}^\top \mathbf{X}^\top \mathbf{e}_G \right)_i.$$

Expanding the term $\mathbf{A}^\top$ yields

$$\frac{\partial}{\partial w_i}\left[\frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2}\right] = -\left(\text{diag}(\mathbf{e}_G) \mathbf{X} \left(\mathbf{X}^\top \mathbf{W} \mathbf{X} \right)^{-1} \mathbf{X}^\top \mathbf{e}_G \right)_i. \tag{B.13}$$

Substituting Equation B.13 into Equation B.3, we finally obtain

$$\begin{aligned}\frac{\partial O(\cdot)}{\partial w_i} &= \frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2ns^2} + \frac{\text{Tr}(\mathbf{W})}{ns^2} \cdot \frac{\partial}{\partial w_i}\left[\frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2}\right] \\ &= \frac{\|\hat{\mathbf{y}}_G - \hat{\mathbf{y}}\|^2}{2ns^2} - \frac{\text{Tr}(\mathbf{W})}{ns^2} \cdot \left(\text{diag}(\mathbf{e}_G) \mathbf{X} \left(\mathbf{X}^\top \mathbf{W} \mathbf{X} \right)^{-1} \mathbf{X}^\top \mathbf{e}_G \right)_i.\end{aligned}$$

■

C

CONFIDENCE ELLIPSOID INTERSECTION

In this section, we discuss how to test for the intersection of the $(1-\alpha)\times 100\%$ confidence ellipsoids of two estimated regression coefficient vectors $\hat{\beta}^1$ and $\hat{\beta}^2$. Recall from Section 4.1 that according to normal theory [6], the $(1-\alpha)\times 100\%$ confidence ellipsoid of the estimation $\hat{\beta}$ of a vector of regression coefficients is given by all vectors β^* satisfying

$$\frac{\left(\beta^*-\hat{\beta}\right)^\top \mathbf{X}^\top \mathbf{X}\left(\beta^*-\hat{\beta}\right)}{ps^2} \leq F(p,n-p,1-\alpha),$$

where

$$s^2 = \frac{\mathbf{e}^\top \mathbf{e}}{n-p},$$

and $F(p,n-p,1-\alpha)$ denotes the $1-\alpha$ probability point of the central F-distribution with p and $n-p$ degrees of freedom.

Our procedure for testing this intersection at a specified confidence level is a two-step process. First, we check whether the centers of either of the ellipsoids, i.e., the regression coefficient vectors, lie within the other vector's confidence ellipsoid. If this is the case, the ellipsoids clearly intersect, and the test is completed. If this is not the case, we need to solve a constrained convex optimization problem to test for intersection. This optimization problem would also work without performing the first test, but clearly this would be computationally less efficient.

To formalize the setting of the problem, we assume that we are given a full column-rank $n\times p$ matrix $\mathbf{X}$, and an $n\times 1$ target vector $\mathbf{y}$. Let $\mathbf{W}_1$ and $\mathbf{W}_2$ be two $n\times n$ diagonal zero-one matrices that define which records the vectors $\hat{\beta}^1$ and $\hat{\beta}^2$ are estimated on, respectively. We assume that both $\mathbf{W}_1\mathbf{X}$ and $\mathbf{W}_2\mathbf{X}$ are also full column-rank. The j-th, $j\in\{1,2\}$, coefficient vector is estimated as

$$\hat{\beta}^j = (\mathbf{X}^\top \mathbf{W}_j \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{W}_j \mathbf{y}.$$

We write $\hat{\mathbf{y}}_1$ and $\hat{\mathbf{y}}_2$ for the estimated target vectors, and these are given by

$$\hat{\mathbf{y}}_j \equiv \mathbf{W}_j \mathbf{X}\hat{\beta}^j.$$

The vectors of residuals are denoted $\mathbf{e}_1$ and $\mathbf{e}_2$, and are given by

$$\mathbf{e}_j \equiv \mathbf{W}_j\mathbf{y} - \hat{\mathbf{y}}_j.$$

The unbiased estimators of the error variance s^2 are denoted s_1^2 and s_2^2, and are now given by

$$s_j^2 \equiv \frac{\mathbf{e}_j^\top \mathbf{e}_j}{\mathrm{Tr}(\mathbf{W}_j) - p},$$

where $\mathrm{Tr}(\mathbf{W}_j)$ denotes the trace of $\mathbf{W}_j$, i.e., the number of records that $\hat{\beta}^j$ is estimated on. The $(1-\alpha) \times 100\%$ confidence ellipsoids $E_j(\cdot)$ are given by all vectors β^* satisfying

$$E_j(\beta^*) \equiv \frac{\left(\beta^* - \hat{\beta}^j\right)^\top \mathbf{X}^\top \mathbf{W}_j \mathbf{X} \left(\beta^* - \hat{\beta}^j\right)}{p s_j^2} \leq F(p, \mathrm{Tr}(\mathbf{W}_j) - p, 1-\alpha).$$

These ellipsoids are centered on the coefficient vectors $\hat{\beta}^j$. For the preliminary intersection test, we may now simply test whether

$$E_1(\hat{\beta}^2) \leq F(p, \mathrm{Tr}(\mathbf{W}_1) - p, 1-\alpha)$$

is satisfied, and *mutatis mutandis*, for $\hat{\beta}^1$ and $E_2(\cdot)$. If either of these inequalities holds, the ellipsoids intersect and the test is completed.

Alternatively, if neither of these inequalities is satisfied, we need to perform a somewhat more involved intersection test. This procedure [28] works as follows. We first rewrite the ellipsoid definitions to hold for a unit inequality. Without loss of generality, ellipsoid E_1 is then transformed to the unit ball centered on the origin, and ellipsoid E_2 is transformed accordingly into an ellipsoid E_2'. We then find the point on E_2' that is closest to the origin, i.e., to the center of the transformed ellipsoid E_1'. If the distance of this closest point to the origin is less than one, it clearly lies within the unit ball, and hence, the ellipsoids must intersect. To find this closest point to the origin, E_2' is transformed into the unit ball centered on the origin, and the origin of the first transformed space is transformed accordingly. We then find the point on the unit ball that is closest to the transformed origin. This last step involves solving a specific case of a Quadratically Constrained Quadratic Program (QCQP) that reduces to a convex optimization problem with a single linear inequality constraint [3].

Our first step will be to rewrite the ellipsoid definitions so as not to bloat our notation too much. We write the j-th ellipsoid as all vectors $\mathbf{x}$ satisfying

$$E_j(\mathbf{x}) = (\mathbf{x} - \mathbf{c}_j)^\top \mathbf{M}_j (\mathbf{x} - \mathbf{c}_j) \leq 1, \tag{C.1}$$

where $\mathbf{c}_j \equiv \hat{\beta}^j$ is the center of the j-th ellipsoid, and

$$\mathbf{M}_j \equiv \frac{\mathbf{X}^\top \mathbf{W}_j \mathbf{X}}{p s_j^2 \cdot F(p, \mathrm{Tr}(\mathbf{W}_j) - p, 1-\alpha)}.$$

By the assumption that $\mathbf{W}_j\mathbf{X}$ has full column-rank, we have that $\mathbf{M}_j$ is symmetric positive definite. We may thus use the Cholesky decomposition $\mathbf{M}_j = \mathbf{L}_j\mathbf{L}_j^\top$ to rewrite Equation C.1 as

$$\begin{aligned} E_j(\mathbf{x}) &= (\mathbf{x} - \mathbf{c}_j)^\top \mathbf{L}_j\mathbf{L}_j^\top (\mathbf{x} - \mathbf{c}_j) \\ &= \|\mathbf{L}_j^\top (\mathbf{x} - \mathbf{c}_j)\|^2 \leq 1. \end{aligned} \tag{C.2}$$

We are now in a position to transform one of the ellipsoids into the unit ball centered on the origin. Without loss of generality, we will transform E_1 in this way, and transform E_2 accordingly. We write E_1' and E_2' for the transformed ellipsoids. For a vector $\mathbf{x}$ in the original space, we write this vector in the transformed space as $\mathbf{x}'$, and let the transformation be given by

$$\mathbf{x}' \equiv \mathbf{L}_1^\top (\mathbf{x} - \mathbf{c}_1). \tag{C.3}$$

The transformed ellipsoid E_1' is defined by substitution of Equation C.3 into Equation C.2 so that we have

$$E_1'(\mathbf{x}') \equiv \mathbf{x}'^\top \mathbf{x}' \leq 1. \tag{C.4}$$

This transformation may be reversed through

$$\mathbf{x} = \mathbf{c}_1 + \mathbf{L}_1^{-\top} \mathbf{x}'. \tag{C.5}$$

We may find the transformed ellipsoid E_2' by noting that it is given by all transformed vectors $\mathbf{x}'$ that satisfy Equation C.2. Moving the parameterization to the transformed space by substituting Equation C.5 into Equation C.2, we have

$$\begin{aligned} E_2'(\mathbf{x}') &\equiv \|\mathbf{L}_2^\top \left(\mathbf{c}_1 + \mathbf{L}_1^{-\top}\mathbf{x}' - \mathbf{c}_2\right)\|^2 \\ &= \|\mathbf{L}_2^\top \left(\mathbf{L}_1^{-\top}\mathbf{x}' - (\mathbf{c}_2 - \mathbf{c}_1)\right)\|^2 \\ &= \|\mathbf{L}_2^\top \left(\mathbf{L}_1^{-\top}\mathbf{x}' - \mathbf{L}_1^{-\top}\mathbf{L}_1^\top(\mathbf{c}_2 - \mathbf{c}_1)\right)\|^2 \\ &= \|\mathbf{L}_2^\top \mathbf{L}_1^{-\top} \left(\mathbf{x}' - \mathbf{L}_1^\top(\mathbf{c}_2 - \mathbf{c}_1)\right)\|^2 \\ &= \|\mathbf{L}'^\top_2 \left(\mathbf{x}' - \mathbf{c}_2'\right)\|^2 \leq 1. \end{aligned} \tag{C.6}$$

Here, we write the center $\mathbf{c}_2$ of E_2 in the transformed space as $\mathbf{c}_2'$. Furthermore, the transformed ellipsoid's matrix $\mathbf{M}_2'$ is obtained through the Cholesky decomposition $\mathbf{M}_2' = \mathbf{L}_2'\mathbf{L}'^\top_2$ where $\mathbf{L}_2' \equiv \mathbf{L}_1^{-1}\mathbf{L}_2$.

We can now formulate the intersection test as the question of whether there exists a point $\mathbf{x}'^*$ that satisfies both Equation C.4 and Equation C.6, that is, whether there exists a point $\mathbf{x}'^*$ that is inside both E_1' and E_2'. Because E_1' is the origin-centered unit ball in $\mathbf{x}'$-space, we can answer

this question by finding the point $\mathbf{x}'^*$ inside E_2' that is closest to the origin. Clearly, we may write the origin of $\mathbf{x}'$-space as $\mathbf{c}_1'$. If this closest point satisfies $\mathbf{x}'^{*\top}\mathbf{x}'^* \leq 1$, it must be within E_1', and hence the ellipsoids intersect.

We can find the closest point $\mathbf{x}'^*$, inside E_2', to the origin by solving

$$\begin{aligned} &\text{Minimize} : (\mathbf{x}' - \mathbf{c}_1')^\top (\mathbf{x}' - \mathbf{c}_1') \\ &\text{Subject to} : E_2'(\mathbf{x}') = \|\mathbf{L}'^{\top}_2 (\mathbf{x}' - \mathbf{c}_2')\|^2 \leq 1. \end{aligned} \tag{C.7}$$

The first step in solving this minimization problem is to *again* transform the space, this time so that E_2' becomes the unit ball centered on the origin. We write E_2'' for this transformed ellipsoid, and refer to the transformed space as $\mathbf{x}''$-space. The transformation is given by

$$\mathbf{x}'' \equiv \mathbf{L}'^{\top}_2 (\mathbf{x}' - \mathbf{c}_2'), \tag{C.8}$$

and reversed through

$$\mathbf{x}' = \mathbf{c}_2' + \mathbf{L}'^{-\top}_2 \mathbf{x}''. \tag{C.9}$$

Substituting Equation C.9 into the minimization problem, we have

$$\begin{aligned} \text{Minimize} : &\left(\mathbf{c}_2' + \mathbf{L}'^{-\top}_2 \mathbf{x}'' - \mathbf{c}_1'\right)^\top \left(\mathbf{c}_2' + \mathbf{L}'^{-\top}_2 \mathbf{x}'' - \mathbf{c}_1'\right) \\ &= \mathbf{x}''^\top \mathbf{L}'^{-1}_2 \mathbf{L}'^{-\top}_2 \mathbf{x}'' + 2(\mathbf{c}_2' - \mathbf{c}_1')^\top \mathbf{x}'' + (\mathbf{c}_2' - \mathbf{c}_1')^\top (\mathbf{c}_2' - \mathbf{c}_1') \\ &\propto \mathbf{x}''^\top \mathbf{L}'^{-1}_2 \mathbf{L}'^{-\top}_2 \mathbf{x}'' + 2(\mathbf{c}_2' - \mathbf{c}_1')^\top \mathbf{x}''. \end{aligned}$$

Substituting Equation C.8 into the minimization problem's constraint, the final problem becomes

$$\begin{aligned} &\text{Minimize} : \mathbf{x}''^\top \mathbf{L}'^{-1}_2 \mathbf{L}'^{-\top}_2 \mathbf{x}'' + 2(\mathbf{c}_2' - \mathbf{c}_1')^\top \mathbf{x}'' \\ &\text{Subject to} : \mathbf{x}''^\top \mathbf{x}'' \leq 1. \end{aligned}$$

After we solve this problem, we can find $\mathbf{x}'^*$ by applying Equation C.9 to the solution $\mathbf{x}''^*$, and check whether $\mathbf{x}'^{*\top}\mathbf{x}'^* \leq 1$ to test for intersection. It remains to show how to solve this minimization problem.

We first reparameterize the problem to clean up our notation. We write $\mathbf{z} \equiv \mathbf{x}''$, $\mathbf{A} \equiv \mathbf{L}'^{-1}_2 \mathbf{L}'^{-\top}_2$ and $\mathbf{b} \equiv (\mathbf{c}_2' - \mathbf{c}_1')$. We then formulate the problem as an equivalent QCQP that is given by

$$\begin{aligned} &\text{Minimize} : \frac{1}{2}\mathbf{z}^\top \mathbf{A}\mathbf{z} + \mathbf{b}^\top \mathbf{z} \\ &\text{Subject to} : \frac{1}{2}\mathbf{z}^\top \mathbf{I}_{p\times p}\mathbf{z} - \frac{1}{2} \leq 0. \end{aligned} \tag{C.10}$$

Because $\mathbf{A}$ is symmetric positive definite by the fact that its Cholesky decomposition $\mathbf{A} = \mathbf{L}'^{-1}_2 \mathbf{L}'^{-\top}_2$ exists, we can solve this problem in its Lagrangian dual form [3]. We define the Lagrangian $L(\mathbf{z}, \lambda)$ with Lagrange multiplier λ as

$$L(\mathbf{z}, \lambda) \equiv \frac{1}{2}\mathbf{z}^\top \bar{\mathbf{A}}(\lambda)\mathbf{z} + \mathbf{b}^\top \mathbf{z} - \frac{\lambda}{2},$$

where

$$\bar{\mathbf{A}}(\lambda) \equiv \mathbf{A} + \lambda \mathbf{I}_{p\times p}.$$

The Lagrangian dual $g(\lambda)$ is given by

$$g(\lambda) \equiv \inf_{\mathbf{z}} L(\mathbf{z}, \lambda), \tag{C.11}$$

which can be found by taking partial derivatives of $L(\mathbf{z}, \lambda)$ with respect to $\mathbf{z}$ and solving for $\mathbf{z}$. We have

$$\begin{aligned}
\frac{\partial L(\mathbf{z}, \lambda)}{\partial \mathbf{z}} &= \frac{\partial}{\partial \mathbf{z}}\left[\frac{1}{2}\mathbf{z}^\top \bar{\mathbf{A}}(\lambda)\mathbf{z} + \mathbf{b}^\top \mathbf{z} - \frac{\lambda}{2}\right] \\
&= \frac{\partial}{\partial \mathbf{z}}\left[\frac{1}{2}\mathbf{z}^\top \bar{\mathbf{A}}(\lambda)\mathbf{z}\right] + \frac{\partial}{\partial \mathbf{z}}\left[\mathbf{b}^\top \mathbf{z}\right] \\
&= \bar{\mathbf{A}}(\lambda)\mathbf{z} + \mathbf{b}.
\end{aligned}$$

Setting this derivative to zero, we obtain

$$\begin{aligned}
\bar{\mathbf{A}}(\lambda)\mathbf{z} + \mathbf{b} &= \mathbf{0}_{p\times 1} \\
\mathbf{z} &= -\bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b}.
\end{aligned}$$

Substituting into Equation C.11, we have

$$\begin{aligned}
g(\lambda) = \inf_{\mathbf{z}} L(\mathbf{z}, \lambda) &= \inf_{\mathbf{z}} \left\{\frac{1}{2}\mathbf{z}^\top \bar{\mathbf{A}}(\lambda)\mathbf{z} + \mathbf{b}^\top \mathbf{z} - \frac{\lambda}{2}\right\} \\
&= \frac{1}{2}\left(-\bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b}\right)^\top \bar{\mathbf{A}}(\lambda)\left(-\bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b}\right) + \mathbf{b}^\top\left(-\bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b}\right) - \frac{\lambda}{2} \\
&= \frac{1}{2}\mathbf{b}^\top \bar{\mathbf{A}}(\lambda)^{-\top}\mathbf{b} - \mathbf{b}^\top \bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b} - \frac{\lambda}{2} \\
&= -\frac{1}{2}\mathbf{b}^\top \bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b} - \frac{\lambda}{2}.
\end{aligned}$$

We can now find the optimal value of the Lagrange multiplier λ^* by solving the dual problem

$$\begin{aligned}
&\text{Maximize} : -\frac{1}{2}\mathbf{b}^\top \bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b} - \frac{\lambda}{2} \\
&\text{Subject to} : \lambda \geq 0.
\end{aligned} \tag{C.12}$$

This is a single-variable convex optimization problem with a single linear constraint, and is thus readily solved with any number of numeric optimization routines. In case such numeric optimization routines require the specification of the gradient of $g(\lambda)$, we show

$$\begin{aligned}
\frac{\partial g(\lambda)}{\partial \lambda} &= \frac{\partial}{\partial \lambda}\left[-\frac{1}{2}\mathbf{b}^\top \bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b} - \frac{\lambda}{2}\right] \\
&= -\frac{1}{2}\mathbf{b}^\top \frac{\partial}{\partial \lambda}\left[\bar{\mathbf{A}}(\lambda)^{-1}\right]\mathbf{b} - \frac{\partial}{\partial \lambda}\left[\frac{\lambda}{2}\right] \\
&= -\frac{1}{2}\mathbf{b}^\top \frac{\partial}{\partial \lambda}\left[\bar{\mathbf{A}}(\lambda)^{-1}\right]\mathbf{b} - \frac{1}{2} \\
&= \frac{1}{2}\mathbf{b}^\top \bar{\mathbf{A}}(\lambda)^{-1}\frac{\partial}{\partial \lambda}\left[\bar{\mathbf{A}}(\lambda)\right]\bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b} - \frac{1}{2} \\
&= \frac{1}{2}\mathbf{b}^\top \bar{\mathbf{A}}(\lambda)^{-1}\frac{\partial}{\partial \lambda}\left[\left(\mathbf{A} + \lambda \mathbf{I}_{p\times p}\right)\right]\bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b} - \frac{1}{2} \\
&= \frac{1}{2}\mathbf{b}^\top \bar{\mathbf{A}}(\lambda)^{-1}\mathbf{I}_{p\times p}\bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b} - \frac{1}{2} \\
&= \frac{1}{2}\mathbf{b}^\top \bar{\mathbf{A}}(\lambda)^{-1}\bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b} - \frac{1}{2} \\
&= \frac{1}{2}\mathbf{b}^\top \bar{\mathbf{A}}(\lambda)^{-2}\mathbf{b} - \frac{1}{2}.
\end{aligned}$$

Here, we have made use of the identity [27] $\frac{\partial \mathbf{Y}^{-1}}{\partial x} = -\mathbf{Y}^{-1}\frac{\partial \mathbf{Y}}{\partial x}\mathbf{Y}^{-1}$ for any matrix $\mathbf{Y}$. Using the same equality, we see that the second-order derivative of $g(\lambda)$ is given by

$$\begin{aligned}
\frac{\partial^2 g(\lambda)}{\partial \lambda^2} &= -\mathbf{b}^\top \bar{\mathbf{A}}(\lambda)^{-1}\bar{\mathbf{A}}(\lambda)^{-1}\bar{\mathbf{A}}(\lambda)^{-1}\mathbf{b} \\
&= -\mathbf{b}^\top \bar{\mathbf{A}}(\lambda)^{-3}\mathbf{b}.
\end{aligned}$$

Because $\bar{\mathbf{A}}(\lambda)$ is symmetric positive definite for $\lambda \geq 0$, we have that $\bar{\mathbf{A}}(\lambda)^{-3}$ is also symmetric positive definite for non-negative λ. It follows that for $\lambda \geq 0$, $\frac{\partial^2 g(\lambda)}{\partial \lambda^2}$ is negative everywhere, and $g(\lambda)$ is indeed concave.

In closing, by Slater's condition, strong duality holds [3] between Equation C.12 and Equation C.10. Thus, we can find the closest point $\mathbf{x}''^*$ on E_2'' to $\mathbf{c}_1''$ through

$$\mathbf{x}''^* = \mathbf{z}^* = -\bar{\mathbf{A}}(\lambda^*)^{-1}\mathbf{b}.$$

■

BIBLIOGRAPHY

[1] K. Bache, M. Lichman: *UCI Machine Learning Repository* [http://archive.ics.uci.edu/ml], University of California, Irvine, School of Information and Computer Science, 2013.

[2] C. M. Bishop: *Pattern Recognition and Machine Learning*, Springer, 2006.

[3] S. Boyd, L. Vandenberghe: *Convex Optimization*, Cambridge University Press, 2004.

[4] L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen: *Classification and Regression Trees*, Chapman & Hall/CRC, 1984.

[5] W. W. Cohen: *Fast effective rule induction*, Proceedings of the Twelfth International Conference on Machine Learning, 1995.

[6] R. D. Cook: *Detection of Influential Observation in Linear Regression*, Technometrics, Volume 19, Issue 1, 1977.

[7] R. D. Cook, S. Weisberg: *Residuals and Influence in Regression*, Chapman & Hall, London, 1982.

[8] M. Costanigro, R. C. Mittelhammer, J. J. McCluskey: *Estimating Class-Specific Parametric Models under Class Uncertainty: Local Polynomial Regression Clustering in an Hedonic Analysis of Wine Markets*, Journal of Applied Econometrics 24, 2009.

[9] J. Demšar: *Statistical Comparison of Classifiers over Multiple Data Sets*, Journal of Machine Learning Research, Volume 7, 2006.

[10] C. Dougherty: *Introduction to Econometrics (4th edition)*, Oxford University Press, 2011.

[11] W. Duivesteijn, A. J. Knobbe, A. Feelders, M. van Leeuwen: *Subgroup Discovery meets Bayesian networks - an Exceptional Model Mining approach*, Proceedings of the 10th IEEE International Conference on Data Mining (ICDM'10), 2010.

[12] W. Duivesteijn, A. Feelders, A. Knobbe: *Different Slopes for Different Folks: Mining for Exceptional Regression Models with Cook's Distance*, Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, 2012.

[13] W. H. Greene: *A statistical model for credit scoring*, New York University, Leonard N. Stern School of Business, 1992.

[14] W. H. Greene: *Econometric Analysis (7th Edition)*, Prentice Hall, 2011.

[15] T. Hastie, R. Tibshirani, J. J. H. Friedman: *The elements of statistical learning*, Volume 1, Springer, 2001.

[16] M. G. Kendall: *A New Measure of Rank Correlation*, Biometrika, 1938.

[17] W. Klösgen: *Explora: a multipattern and multistrategy discovery assistant*, Advances in Knowledge Discovery and Data Mining, 1996.

[18] W. Klösgen: *Subgroup Discovery*, Handbook of Data Mining and Knowledge Discovery, Ch. 16.3, Oxford University Press, 2002.

[19] A. Knobbe: *Cortana Open-Source Subgroup Discovery*, [http://datamining.liacs.nl/cortana.html].

[20] R. Kohavi: *Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid*, Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, 1996.

[21] S. Kullback, R. A. Leibler: *On Information and Sufficiency*, The Annals of Mathematical Statistics, Volume 22, Issue 1, 1951.

[22] M. van Leeuwen: *Maximal exceptions with minimal descriptions*, Data Mining and Knowledge Discovery, Volume 21, Issue 2, 2010.

[23] M. van Leeuwen, A. J. Knobbe: *Diverse Subgroup Set Discovery*, Data Mining and Knowledge Discovery, special issue ECML PKDD'11, Volume 25, Issue 2, 2012.

[24] D. Leman, A. Feelders, A. Knobbe: *Exceptional Model Mining*, Proceedings of the ECML/PKDD'08, Volume 5212, 2008.

[25] T. S. Lim, W. Y. Loh, Y. S. Shih: *A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms*, Machine Learning, Issue 40, Volume 3, 2000.

[26] J. Pearl: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988.

[27] K. B. Petersen, M. S. Pedersen: *The Matrix Cookbook*, Technical University of Denmark, 2008.

[28] S. B. Pope: *Algorithms for Ellipsoids*, Sibley School of Mechanical & Aerospace Engineering, Cornell University, Report: FDA-08-01, 2008.

[29] R Core Team: *R: A Language and Environment for Statistical Computing*, [http://www.R-project.org], R Foundation for Statistical Computing, 2013.

[30] J. R. Quinlan: *C4.5: Programs for Machine Learning*, Volume 1, Morgan Kaufmann, 1993.

[31] C. J. van Rijsbergen: *Information Retrieval (2nd Edition)*, Butterworths, 1979.

[32] J. Rissanen: *Modeling by Shortest Data Description*, Automatica, Volume 14, Issue 1, 1978.

[33] A. Siebes, J. Vreeken, M. van Leeuwen: *Item Sets that Compress*, Proceedings of the SDM'06, 2006.

[34] Y. Xu, A. Fern: *On learning linear ranking functions for beam search*, Proceedings ICML 2007, 2007.