Under consideration for publication in Knowledge and Information Systems

Efficient Algorithms for Finding Optimal Binary Features in Numeric and Nominal Labeled Data

Michael Mampaey[1], Siegfried Nijssen[2,3], Ad Feelders[4], Rob Konijn[2,5], and Arno Knobbe[2]

[1]University of Bonn, Germany
[2]Leiden University, The Netherlands
[3]KU Leuven, Belgium
[4]Utrecht University, The Netherlands
[5]VU University Amsterdam, The Netherlands

Abstract. An important subproblem in supervised tasks such as decision tree induction and subgroup discovery is finding an interesting binary feature (such as a node split or a subgroup refinement) based on a numeric or nominal attribute, with respect to some discrete or continuous target variable. Often one is faced with a trade-off between the expressiveness of such features on the one hand, and the ability to efficiently traverse the feature search space on the other hand. In this article, we present efficient algorithms to mine binary features that optimize a given convex quality measure. For numeric attributes we propose an algorithm that finds an optimal interval, whereas for nominal attributes we give an algorithm that finds an optimal value set. By restricting the search to features that lie on a convex hull in a coverage space, we can significantly reduce computation time. We present some general theoretical results on the cardinality of convex hulls in coverage spaces of arbitrary dimensions, and perform a complexity analysis of our algorithms. In the important case of a binary target, we show that these algorithms have linear runtime in the number of examples. We further provide algorithms for additive quality measures, which have linear runtime regardless of the target type. Additive measures are particularly relevant to feature discovery in subgroup discovery. Our algorithms are shown to perform well through experimentation, and furthermore provide additional expressive power leading to higher-quality results.

Keywords: Binary features, decision trees, subgroup discovery, numeric data, nominal data, labeled data, convex functions, convex hulls, coverage space, ROC analysis

Received Mar 09, 2013
Revised Aug 16, 2013
Accepted Sep 07, 2013

1. Introduction

Many tasks in data mining are supervised in nature: one wishes to predict or characterize a given target attribute. Important methods that can be used to solve such tasks are those that find *symbolic* models or descriptions of data. Well-known examples of symbolic predictive models are decision trees, regression trees, and rule-based models. Subgroup discovery (SD) is an important example of a descriptive data mining task [Herrera et al., 2010, Klösgen, 2002, Wrobel, 1997]. A characterizing feature of these symbolic models and patterns is that they are built from *conditions* on attributes: in decision trees, internal nodes consist of attribute-value conditions; in rule-based models and subgroup discovery, rules consist of conjunctions of conditions on attributes.

As an example, consider the problem of subgroup discovery in the context of fraud detection in a medical insurance dataset. The aim is to find subsets of the data where the distribution of a (binary) target variable is substantially different from its distribution in the whole data, assessed by a quality measure [Klösgen, 2002, Wrobel, 1997]. The data contains records about all claims (treatments) that were made over a certain period. We are investigating a specific medical practitioner, say a dentist, and define a target concept to reflect this: positive cases are claimed by the dentist under investigation, negative cases are claimed by one of the other dentists. We look for conditions on the attributes, such that claims satisfying the conditions (members of the *subgroup*) are from the dentist in question significantly more than when taking a random sample. As such, the attribute conditions featuring in the subgroup description provide information as to the value of the target. In this dataset, subgroup descriptions could consist of (conjunctions of) conditions such as

$$Age \geq 67 \qquad \text{and} \qquad Age < 15 \wedge Treatment = \textit{E13: root canal treatment} \ .$$

Similarly, when learning decision trees on this data, we could look for attributes of the claim (treatment codes, costs, patient details, ...) on which we can perform tests that 'predict' the dentist of interest.

Clearly, when finding such symbolic models or subgroup descriptions, an important choice to make is which conditions to consider. In most symbolic models, these are equalities for nominal attributes (such as $Treatment = \textit{E13: root canal treatment}$) and inequalities for numeric attributes (such as $Age \geq 67$). Important reasons for choosing these conditions are that they are easy to interpret, and that they can be found reasonably efficiently.

Straightforward algorithms for finding good combinations of conditions can be computationally demanding. For instance, discovering a combination of constraints on a patient's age, such as $18 \leq Age \leq 67$, would involve a quadratic number of calculations (with respect to the number of threshold values considered) to find the optimum.

To avoid this complexity, most well-known methods apply a greedy search strategy that finds thresholds while searching for models: a number of potential cut-points of a given numeric attribute is computed on the fly, by considering the subset of data currently under investigation. In decision trees, this subset concerns the examples that fall in the current branch to be split, whereas in SD, it concerns the examples in the current candidate subgroup that needs refining. Only the most promising splits are considered further. Whereas this makes the search feasible, it also limits the models or SD descriptions that can be found.

In this article we propose an improvement of this state of the art of symbolic modeling and subgroup discovery, based on the observation that a larger space of conditions allows for finding models and patterns of higher quality. The improvement consists of efficient algorithms for finding two fundamental types of conditions: intervals for continuous

attributes, and value sets for nominal attributes. We prove that in the binary case, under reasonable assumptions, the best scoring such conditions can be found without increasing the computational complexity of symbolic modeling or subgroup discovery systems.

Interval conditions are of the form $Age \in]18, 35]$. They allow us to focus directly on ranges of values of interest. Such conditions are hard to find by traditional greedy algorithms, as they would need to find the conditions $Age > 18$ and $Age \leq 35$ in two stages. A straightforward algorithm would involve $O(N + t^2)$ calculations, where N is the number of examples in the current node or subgroup, and t is the number of possible thresholds. Instead, we propose a new efficient algorithm that can directly mine the best scoring specialization of a rule or model over all possible intervals. We show that in the important binary case, an optimum can be found in $O(N+t)$ time, for any convex quality measure; hence, finding an interval is not more expensive than finding an inequality such as $Age > 18$ and is feasible even on large datasets.

Set-valued conditions are conditions of the form $Treatment \in \{$*M55: dental cleaning*, *V21: polishing a filling*, *V11: one-sided filling*$\}$. It is likely that considering an individual treatment does not result in a subgroup indicative of fraud by a specific dentist, while an unusual combination does (certain combinations of treatments are actually not permitted). As such, if we only consider single values of nominal attributes, we miss out on some potentially useful subgroups. To some extent this can be alleviated by imposing a hierarchy on the attribute values, but such a hierarchy would have to be specified by the user and still considerably restricts the number of possible groupings. We can avoid such problems by directly searching for an optimal subset of attribute values. We present an algorithm that finds an optimal set in polynomial time, and prove that in the important binary case its runtime is $O\left(N + d\right)$, where d is the size of the attribute domain. We further prove that with a straightforward optimization, the algorithm presented by Breiman et al. [1984] for binary targets can be implemented to run in linear time with respect to the number of examples, rather than linearithmic time. To the best of our knowledge, this is a novel result.

From a theoretical perspective, the above algorithms can easily be added to greedy symbolic modeling systems and subgroup discovery systems. In the very common binary case, this can even be done without affecting their asymptotic runtime complexity. However, as the algorithms are conceptually more involved, it may be harder to implement them efficiently. We address this issue by showing that simpler algorithms can be used for additive quality measures such as Weighted Relative Accuracy (WRAcc), which are particularly relevant in subgroup discovery.

Note that every condition can be seen as a new binary feature: instead of including the test $Age \geq 67$ in a rule, we could also have included an additional column in our data with boolean values representing the truth of $Age \geq 67$. A rule or tree could then be represented by conditions on these boolean attributes; hence, finding conditions also amounts to identifying interesting boolean features for an originally non-boolean dataset.

In the remainder of this article, we mostly use subgroup discovery as an example for the application of our algorithms; however, it should be stressed that the algorithms can also be used in other tasks, such as decision tree learning, regression tree learning, and the induction of rule-based predictive models, which are algorithmically closely related. This is despite the fact that the goals of subgroup discovery and, e.g., classification are distinct: a classifier is a global predictive model, whereas a subgroup is a local descriptive pattern. A good subgroup is therefore not necessarily also a good classification rule or vice versa. Many subgroup discovery quality measures are asymmetric, considering only the subgroup itself, whereas a node split in a decision tree takes into account the impurity of all its child nodes. Moreover, the impurity of a split in a decision tree is measured against the class distribution in the parent node, whereas a subgroup (refinement)'s quality is

compared against the global target distribution. That being said, from an algorithmic perspective the techniques presented here are readily applicable in all these areas.

This article is structured as follows. First, we discuss related work in Section 2. In Section 3, the necessary preliminaries and notation are introduced. Section 4 discusses convex hulls and coverage spaces, and some general theoretical results. Then, in Sections 5 and 6 we present two algorithms to efficiently find optimal features in the form of intervals and value sets, for numeric and nominal attributes, respectively. In Section 7 we experimentally evaluate these algorithms on synthetic and benchmark data, as well as on a real-world health insurance dataset. Finally, we end with conclusions in Section 8.

2. Related Work

The discovery of conditions is required in several important supervised mining tasks, including predictive modeling tasks and subgroup discovery or pattern mining tasks.

Subgroup discovery and pattern mining In subgroup discovery and supervised pattern mining, the goal is to find symbolic descriptions for subgroups of the data in which the distribution of a target attribute is significantly different from the distribution for the whole dataset; hence, a fixed target attribute is assumed, in contrast to what is common in association rule discovery [Agrawal et al., 1993, Hämäläinen, 2012].

Usually, the symbolic descriptions consist of conjunctions of conditions. The simplest case is the one in which the data is binary; conditions in this case only need to test the truth value of the features. Subgroup discovery, and more generally pattern mining, is dominated by algorithms that are restricted to such discrete data.

The algorithms that find subgroup descriptions or patterns can be categorized into two classes: those that perform an exhaustive search under constraints, and those that are greedy. In particular exhaustive search algorithms that operate directly on numerical data face large computational challenges if no strong restrictions are applied.

Grosskreutz and Rüping [2009] propose an exhaustive algorithm for mining top-k subgroups in numeric data. Their approach returns the k globally best subgroups using interval descriptions, for a family of quality measures that includes WRAcc. The method is defined purely for numeric attributes, rather than heterogeneous data. Search space pruning is achieved by using a data structure that maintains optimistic estimates for attributes; only the size of this data structure is already quadratic in the number of split points in the worst case.

Fukuda et al. [1999] present two algorithms for mining optimal association rules. The search is made feasible by limiting the search to rules of the form $A \in [l, r] \Rightarrow C$, for a fixed numeric attribute A and (binary) consequent C, in the well-known support-confidence framework, optimizing for one measure whilst using a threshold for the other. The threshold is necessary in order to avoid discovering trivial rules. These algorithms have linear runtime, but are specific to the support and confidence quality measures, and work in a support-positive support space rather than in ROC space.

To avoid a large computational complexity, and to be able to find conjunctions of conditions, most subgroup discovery systems that operate on numerical data either perform pre-processing of numeric data or avoid exhaustive search altogether. In both cases, there is the risk of losing information [Atzmüller and Puppe, 2006, Kavšek et al., 2003]; however, the advantage of systems that do not rely on exhaustive search is that it more feasible to perform discretization during the search.

A good example of a system that performs on-the-fly discretization is Cortana [Meeng and Knobbe, 2011]. While searching for subgroup descriptions, it searches for inequality

conditions for numerical attributes, and allows for on-the-fly binning; its search is greedy as the search only continues for the best conditions found.

The algorithms studied in this work are specifically aimed for systems of this kind, which can be characterized by these two properties:

- they use heuristics to search through a space of conjunctions;
- during the search they repeatedly perform discretization for subsets of the same data.

The latter property is important, since it means that it may pay off to perform calculations (such as sorting the data) before the search is started. The first property is important, since it means that these systems may not be able to find good conjunctions of conditions through search. Only by a sufficiently broad set of conditions can the discovery of certain subgroups be ensured.

To find good conditions in a greedy search, subgroup discovery algorithms often use similar algorithms as decision tree learning algorithms, as discussed next.

Decision Trees In decision trees, an important challenge is how to find the conditions that are put into the internal nodes of the trees. Most algorithms operate in a top-down greedy fashion, and hence also perform a repeated search for conditions in subsets of the data [Breiman et al., 1984, Rzepakowski and Jaroszewicz, 2012].

A numeric attribute at a given node in a decision tree is typically split into k child nodes, using $k-1$ threshold values (often $k=2$). How to find a good split point has been the topic of several past studies.

Fayyad and Irani [1993] proved that for a convex impurity measure, an optimal split point must always be a boundary point, that is, a point for which the two adjacent values on either side of the split point have distinct labels (assuming no duplicate values occur). This was generalized by Elomaa and Rousu [2004], who showed that a threshold in an optimal k-way split never separates two adjacent base intervals with an equal class distribution. The number of potential split points can thus be reduced considerably, in linear time with respect to the number of data points. For $k=2$, finding the optimal split clearly takes linear time; for $k \geq 3$, the optimal k-way split can be found in quadratic time with respect to the number of split points, using dynamic programming.

The algorithms that we will propose in this article are related to these earlier algorithms; however, in contrast to these earlier algorithms, we use two thresholds (the interval endpoints) to define a binary split, i.e., we separate data points within the interval from those outside of it.

As conditions for nominal attributes, decision trees can either perform a d-way split on all possible values (where d is the domain size of the attribute), or can search for the optimal subset of values (and its complement), to split the node into two child nodes. Such a node splitting algorithm is used by CART [Breiman et al., 1984], which works with a binary target.

A contribution of this work is that we will show that a straightforward optimization lowers the complexity of this algorithm from $O\left(N + d \log d\right)$ to $O\left(N + d\right)$.

Chou [1991] presented an approximate technique to find a good split of a nominal attribute. This algorithm uses a k-means-like approach, which is justified by the observation that a globally optimum solution must satisfy a nearest neighbor criterion. For binary splits specifically (as considered in this article), the complexity is $O\left(dc\right)$ per iteration, where d is the size of the domain and c is the number of classes of the target. The number of iterations required until convergence, however, is not known beforehand, and moreover, the result is only an approximation of the optimal solution.

3. Preliminaries and Notation

We assume that we are given a dataset D containing examples $\vec{x} \in D$ of the form $\vec{x} = (a_1, \ldots, a_k, c)$, where k is a positive integer. Each a_i is the value of a corresponding *attribute* A_i, and we write $\vec{a} = (a_1, \ldots, a_k)$. We call c the *class label* or *target value* of $\vec{x}$. Each attribute value a_i is taken from a domain $dom\,(A_i)$. We consider binary, nominal, and numeric attributes, that is, attributes for which, $dom\,(A) = \{0, 1\}$, $|dom\,(A)| \in \mathbb{N}_0$, and $dom\,(A) = \mathbb{R}$, respectively. The target is also allowed to be binary, nominal, or numeric; we focus mostly on the common case of binary targets. The size of the dataset is denoted by $N = |D|$. If the target is nominal, the subset of the data for which the class is equal to i is written as $D_i = \{\vec{x} \in D \mid c = i\}$, and its size as $N_i = |D_i|$. For binary targets, these are written as D_+ and D_- to denote the positive and negative part of the data, respectively. If the target is continuous, we consider the mean μ_0 and standard deviation σ_0 of the target.

Definition 1. A *feature* on an attribute A is a binary function $f : dom\,(A) \to \{0, 1\}$. We consider features that are described as one of four types of basic conditions:

1. equalities: $A = a$, where $a \in dom\,(A)$,
2. inequalities: $A \leq a$, where $a \in dom\,(A)$,
3. value sets: $A \in V$, where $V \subseteq dom\,(A)$, and
4. intervals: $A \in]l, r]$, where $l < r \in dom\,(A)$.

Conditions 1 and 3 are applicable to nominal (including binary) attributes, whereas conditions 2 and 4 are applicable to numeric (and ordinal) attributes. Note that 1 and 2 are in fact special cases of 3 and 4, respectively.

Conditions can be used in patterns, subgroup descriptions and predictive models such as decision trees.

Definition 2. A *pattern* is a binary function $p : \prod_{i=1}^{k} dom\,(A_i) \to \{0, 1\}$. We assume that a pattern is described by a conjunction of features, that is, a conjunction of conditions on the attributes. A pattern p is said to *cover* a data point $\vec{x} = (\vec{a}, c)$ if and only if $p\,(\vec{a}) = 1$. A *subgroup* corresponding to a pattern p is the bag of data points $G_p \subseteq D$ that are covered by p:

$$G_p = \{\vec{x} \in D \mid p(\vec{a}) = 1\} \ .$$

If no confusion can arise, we omit the subscript p. We write $n = |G|$ for the size of G. If the target is nominal, n_i denotes the number of examples of class i in G (hence $\sum_i n_i = n$), whereas if the target is continuous, write μ and σ for the mean and standard deviation of the target within G. The complement of a subgroup is written as $\overline{G} = D \setminus G$, and its size as $\overline{n} = \left|\overline{G}\right|$.

Similarly, we can use conditions to construct binary decision trees.

Definition 3. A binary *decision tree* is a binary tree in which every internal node corresponds to one of the basic conditions, and every leaf is labeled with a class label.

Despite the differences between trees and subgroup descriptions, the search for both tasks proceeds in a similar way. The search starts from an empty pattern or tree, and iteratively conditions are added to the pattern or the tree.

The space of candidate patterns in subgroup discovery can be organized in a lattice; by iteratively adding conditions, subgroup discovery algorithms traverse this lattice in a top-down, general-to-specific fashion. Which conditions are added is determined by a

refinement operator $\rho : \mathcal{P} \to 2^{\mathcal{P}}$, a syntactic operation which determines how simple patterns can be extended into more complex ones by atomic additions.

One can distinguish between exhaustive search algorithms, which explore all possible refinements, and greedy algorithms, which only explore the best scoring refinement according to a heuristic.

In decision tree learning, the standard approach is a top-down approach, in which first the root of the tree is searched for using a heuristic; the examples of the data are sorted in the left-hand or the right-hand subtree, based on the condition put in the root of the tree. The search for further refinements proceeds recursively for the left-hand child and the right-hand child of the root.

In both cases, in order to find a good candidate feature, a *quality measure* is used.

For a pattern p in the pattern language $\mathcal{P}$, this is a function that measures how interesting the induced subgroup G_p is.

Definition 4. Given a dataset D, a *quality measure for patterns* is a function $\varphi_D : \mathcal{P} \to \mathbb{R}$ that assigns a numeric value to a pattern p.

For convenience, we also write φ_D as a function of the n_i, the number of examples in class i in G_p, that is, $\varphi_D(p) = \varphi_D(n_1, \ldots, n_c)$ for a nominal target; or as $\varphi_D(p) = \varphi_D(n, \mu, \sigma)$ for a numeric target. We again omit the subscript, whenever the dataset D is clear from the context.

In the case of decision trees, a quality measure is usually evaluated on a single condition only.

Below, we give the definitions of some commonly used quality measures. For more measures, we refer to Fürnkranz and Flach [2005].

Definition 5. We are given a dataset D of size N, with a binary target defining N_+ and N_- as above, and a pattern p with counts n_+ and n_-.
The *Weighted Relative Accuracy* of p is defined as

$$WRAcc_D(n_+, n_-) = \frac{n}{N}\left(\frac{n_+}{n} - \frac{N_+}{N}\right) .$$

The *Binomial test* [Klösgen, 2002] of p is defined as

$$Bin_D(n_+, n_-) = \sqrt{\frac{n}{N}}\left(\frac{n_+}{n} - \frac{N_+}{N}\right) .$$

Weighted Relative Accuracy is one of the most used quality measures in subgroup discovery [Klösgen, 2002, Wrobel, 1997]. It strikes a balance between the target divergence of the subgroup and its size. WRAcc is also known under different names in other contexts in data mining, for instance, leverage in association rule mining, see Novak et al. [2009]. WRAcc is an additive function, for which we provide specialized algorithms in Sections 5 and 6. The Binomial test is similar in form to WRAcc, but is weighted by the square root of the relative support of the subgroup. It can be shown to be order equivalent to the standardized z-score.

As an example of general dimensionality, we consider the *Information Gain* quality measure, which is common in decision tree induction.

Definition 6. The *Information Gain* measure is defined as

$$IG_D(n_1 \ldots, n_c) = h\left(\frac{N_1}{N}, \ldots, \frac{N_c}{N}\right) - \frac{n}{N}h\left(\frac{n_1}{n}, \ldots, \frac{n_c}{n}\right) - \frac{\overline{n}}{N}h\left(\frac{\overline{n}_1}{\overline{n}}, \ldots, \frac{\overline{n_c}}{\overline{n}}\right)$$

where h denotes entropy, which is defined as $h(x_1, \ldots, x_c) = -\sum_{i=1}^{c} x_i \log x_i$.

Finally, we consider the following quality measures for numeric target variables.

Definition 7. Let μ_0 and σ_0 be the global target mean and standard deviation. The *standardized z-score* of a pattern p is defined as

$$z(n, \mu) = \frac{\sqrt{n}}{\sigma_0}(\mu - \mu_0) \ .$$

The *t-statistic* of a pattern p is defined as

$$t(n, \mu, \sigma) = \frac{\sqrt{n}}{\sigma}(\mu - \mu_0) \ .$$

As we shall see below, a crucial property for quality measures is *convexity*.

Definition 8. A function $f : \mathbb{R}^k \to \mathbb{R}$ is called convex if for any $\mathbf{a}, \mathbf{b} \in \mathbb{R}^k$ and $\lambda \in [0, 1]$ it holds that

$$f\left(\lambda\mathbf{a} + (1 - \lambda)\mathbf{b}\right) \leq \lambda f\left(\mathbf{a}\right) + (1 - \lambda) f\left(\mathbf{b}\right) \ .$$

If $-f$ is convex, f is called concave.

We assume that the quality measures we use are convex, which is true for most common quality measures, including WRAcc and Information Gain. This can be seen by verifying that their second derivatives are positive everywhere. Unfortunately, some frequently used measures, such as the Binomial test, the standardized z-score, or the t-statistics are not. However, this need not always be problematic. For instance, the Binomial test is convex if $Bin(n_+, n_-) \geq 0$, or equivalently, if $n_+/n \geq N_+/N$. Typically, subgroups for which $Bin(n_+, n_-) < 0$ are not of interest (otherwise we can, for instance, invert the target). Similarly, the standardized z-score and the t-statistic are convex on the domain where $z(n, \mu) \geq 0$ and $t(n, \mu, \sigma) \geq 0$, that is, where $\mu \geq \mu_0$. For our intents and purposes, we can assume that all the quality measures we use are convex.

Finally, we also assume that the arguments of a quality measure are *monotonic* with respect to subgroup inclusion. That is, for two subgroups $G \subseteq G'$ with corresponding argument values x and x' of some (convex) quality measure φ, it must hold that $x \leq x'$. If the arguments of φ are target counts $(n_1, \ldots, n_c)$, it can easily be seen that this holds. In some cases, however, it may be necessary to rewrite φ; for instance, we can rewrite the t-statistic as a function of a subgroup's size, target sum, and target squared sum, which is a set of sufficient statistics for its size, target mean, and target standard deviation.

4. Convex Hulls in Coverage Space

To get some insight into the quality of features, we reason with them in *coverage space*. First, let us assume that the target attribute is nominal with c classes, and that we use a quality measure φ. Typically, the quality φ of a feature is a function of its counts n_i for each class $i = 1, \ldots, c$. Then the associated coverage space is the c-dimensional space $[0, N_1] \times \cdots \times [0, N_c]$. For a binary labeled dataset D with N_+ positive and N_- negative examples, the corresponding coverage space is the subset $[0, N_-] \times [0, N_+]$ of the plane—by convention the negative counts are on the x-axis, as shown in Figure 1. In this case coverage space is directly related to ROC space, since the latter is simply a normalized version of the former, rescaled to the unit square [Fürnkranz and Flach, 2005]. A feature p with counts n_i for each class i is represented by the *stamp point* $(n_1, \ldots, n_c)$. An important observation to make is that if the target is nominal, then all stamp points in coverage space have integer coordinates.

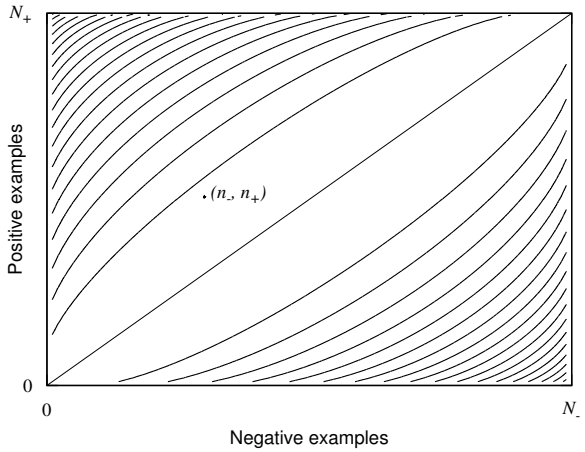


Figure 1. Example of a coverage space of size $N_- \times N_+$, a stamp point with coordinates (n_-, n_+), and the isometrics of the Information Gain quality measure.

Analogously, if the target is continuous, the coverage space is defined by the domains of the arguments of the quality measure φ. For example, if we use the t-statistic, whose arguments are the feature size, mean, and variance, the corresponding coverage space is $[0, N] \times \mathbb{R}^2$. We will see that sometimes it might be convenient to rewrite a quality measure using other sufficient statistics as its arguments, for instance, writing the t-statistic as a function of feature size, target sum, and target squared sum. In the following, we identify features with their corresponding stamp points in coverage space.

For almost any sensible quality measure, it holds that features that are close to the diagonal connecting $(0, \ldots, 0)$ and $(N_1, \ldots, N_c)$, are considered uninteresting, since they have approximately the same target distribution as the whole dataset or the parent node; on the other hand, features closer to any of the c corners $(0, \ldots, N_i, \ldots, 0)$ are reasonably assumed to have high quality because their target distribution is divergent and pure. For a given quality measure, we can plot its isometrics in coverage space connecting points of equal quality, as shown in Figure 1 for Information Gain in two dimensions.

In our hunt for an optimal feature, let us consider the search space. For instance, for an interval feature on a numeric attribute, the search space consists of all possible intervals whose endpoints are taken from some predefined set, e.g., all distinct values occurring in the data. The stamp points corresponding to the search space are contained in a *convex hull* in coverage space.

Definition 9. Given a set of points S in $\mathbb{R}^k$, its convex hull $CH(S)$ is defined as the minimal superset of S for which it holds that if $x, y \in CH(S)$ and $\lambda \in [0, 1]$, then

$$\lambda x + (1 - \lambda)y \in CH(S) \ .$$

If S is finite, it can be seen that $CH(S)$ forms a convex polytope, and thus can be uniquely identified by its non-degenerate vertices, which we shall denote $H(S)$. In other words, $H(S)$ consists of all points lying on the 'edge' of S. It follows that $H(S) \subseteq S$, and in practice it often may hold that $|H(S)| \ll |S|$. In the remainder, we will refer to both $CH(S)$ and $H(S)$ the convex hull of S; from the context it should be clear which one is meant.

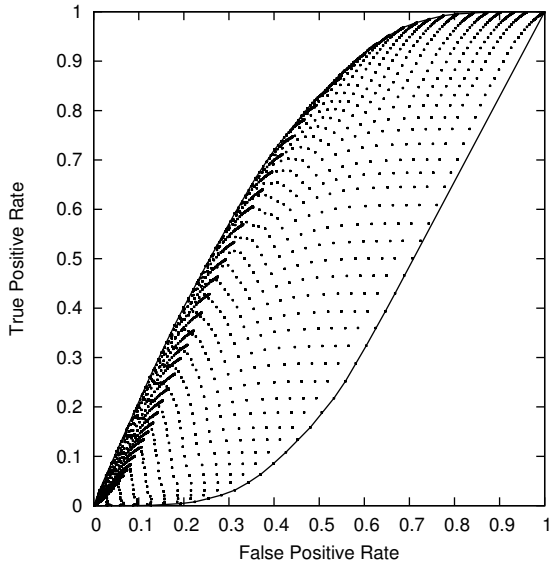


Figure 2. The convex hull of a set of points in ROC space. Only the subsets of points that lie on the hull can maximize a convex quality function.

Property 1. Let S be a set of points in coverage space, and let φ be a convex quality measure, then it holds that

$$\arg\max_{s \in S} \varphi(s) = \arg\max_{s \in H(S)} \varphi(s) \ .$$

Hence, in order to find an optimal point in the search space S, it suffices to only consider the members of the convex hull $H(S)$, rather than the entire set. This fact can be utilized to significantly reduce computation.

An example is given in Figure 2. We selected the *Age* attribute from the *Adult* dataset, which has a binary target attribute [Frank and Asuncion, 2010], and computed the stamp point of every interval feature $Age \in]a, b]$, with a and b taken from the set of all ages occurring in the data. We now want to find a (not necessarily unique) interval maximizing a given convex quality measure φ. Since the attribute has 74 distinct values in the data, there are 2 775 possible intervals to consider. However, since only points that lie on the convex hull can maximize φ, we only need to consider a small fraction of all intervals. In this example, the convex hull consists of a mere 56 points. If we could directly find those intervals whose stamp points lie on the convex hull, we could save ourselves a lot of computational effort.

Many algorithms that calculate the convex hull of a set of points are to be found in the literature. The Graham scan algorithm is one of several well-known planar convex hull algorithms [Graham, 1972]. Given a set of points S in the plane, it computes the convex hull $H(S)$ in $O\left(|S| \log |S|\right)$ time, or in $O\left(|S|\right)$ time if the points are given in some convenient form (e.g., sorted by their polar- or x-coordinates). In three dimensions, $O\left(|S| \log |S|\right)$ algorithms exist as well, whereas in the general c-dimensional case, the complexity of computing $H(S)$ quickly increases to $O\left(|S|^{\lfloor d/2 \rfloor}\right)$ [Preparata and Shamos, 1985].

We must remark, however, that in our setting we want to directly construct $H(S)$, without having to consider all points in S, all the more since the search space is typically defined implicitly rather than enumerated explicitly. We would therefore like to compute $H(S)$ in time that does not depend too strongly on the search space itself ($|S|$), but instead preferably depends on the size of the convex hull to be computed, $|H(S)|$.

To this end, it is useful to investigate just how large a convex hull $H(S)$ can get. The following properties present worst-case upper bounds on the size of the convex hull of sets of points in a c-dimensional discrete coverage space; they are integral to the complexity results presented below. To this end, we start by introducing *Farey sets*, whose elements will correspond to points (or vectors) in coverage space.

Definition 10. We define a Farey set of order k and dimensionality c as the set of all irreducible c-tuples of non-negative integers, whose sum is no greater than k, that is,

$$F_k^c = \{(n_1, \ldots, n_c) \in \mathbb{N}^c \mid \gcd(n_1, \ldots, n_c) = 1 \text{ and } \textstyle\sum_{i=1}^{c} n_i \leq k\} \ .$$

Note that for $c = 2$ this is equivalent to the standard definition of a Farey set using rational numbers [Conway and Guy, 1996], that is,

$$F_k^2 = \left\{ \frac{a}{b} \mid \gcd(a, b) = 1 \text{ and } 0 \leq a \leq b \leq k \right\} \ .$$

Lemma 1. Let S be a set of non-negative irreducible c-tuples, and choose k such that $|S| > |F_k^c|$. Define $R = \sum_{n \in S} \sum_{i=1}^{c} n_i$ and $Q = \sum_{n \in F_k^c} \sum_{i=1}^{c} n_i$. Then $R > Q$.

Proof. This is a straightforward generalization of Lemma 6.2 in [Calders et al., 2013], a proof has therefore been omitted. □

It immediately follows from Lemma 1 that for a fixed R, the cardinality of S is at most that of a Farey set for which Q is (approximately) equal to R.[1]

Lemma 2. Let F_k^c be a Farey set of order k and dimensionality c, and define $Q = \sum_{n \in F_k^c} \sum_{i=1}^{c} n_i$. Then $|F_k^c|$ is sublinear in Q, specifically,

$$|F_k^c| = O\left(Q^{c/c+1}\right) \ .$$

Proof. The proof generalizes that of Calders et al. [2013] for $c = 2$, in which case the tuples are represented as fractions. Let us denote by Z_k^c the number of irreducible c-tuples of non-negative integers, each no greater than k. It was shown by Nymann [1972] that $Z_k^2 = k^2/\zeta(2) + O(k \log k)$, and that $Z_k^c = k^c/\zeta(c) + O\left(k^{c-1}\right)$ for $c \geq 3$, where ζ is the Riemann-zeta function. In the following, we only consider the case where $c \geq 3$; the case $c = 2$ is similar, whereas the case $c = 1$ is trivially verified. From the inequalities $Z_{k/c}^c \leq |F_k^c| \leq Z_k^c$, we can derive that

$$\frac{k^c}{c^c \zeta(c)} + O\left(k^{c-1}\right) \leq |F_k^c| \leq \frac{k^c}{\zeta(c)} + O\left(k^{c-1}\right) \ , \tag{1}$$

and subsequently that there must be some $c' \in]0, 1]$ depending only on c such that

$$|F_k^c| = \frac{c'}{\zeta(c)} k^c + O\left(k^{c-1}\right) \ . \tag{2}$$

[1] Note that it might not be possible to construct a Farey set for every R, we are just interested in asymptotics.

Let us denote $\bar{F}_k^c = F_k^c \setminus F_{k-1}^c$, that is, $\bar{F}_k^c$ contains those tuples whose sum equals exactly k. It follows that $|F_k^c| = \sum_{l=0}^{k} \left|\bar{F}_l^c\right|$. Now we can write

$$Q = \sum_{n \in F_k^c} \sum_{i=1}^{c} n_i = \sum_{l=0}^{k} l \cdot \left|\bar{F}_l^c\right| = k\,|F_k^c| - \sum_{l=0}^{k-1} |F_l^c| \ . \tag{3}$$

Substituting Equation 2, we can calculate

$$Q = \frac{c'}{\zeta(c)} \frac{c-1}{c} k^{c+1} + O\left(k^c\right) \ . \tag{4}$$

Finally, combining Equations 2 and 4, we have proven that asymptotically

$$|F_k^c| = \frac{c'}{\zeta(c)} \left(\frac{\zeta(c)}{c'} \frac{c}{c-1} Q \right)^{c/c+1} = c'' Q^{c/c+1} \ , \tag{5}$$

with $c'' < 2$ since $\zeta(c) > 1$, and therefore

$$|F_k^c| = O\left(Q^{c/c+1}\right) \ . \tag{6}$$

□

Property 2. Given a finite set of points S in a discrete coverage space of size $N_- \times N_+$, the number of vertices on the convex hull $H(S)$ is bounded by $O\left(N^{2/3}\right)$.

Proof. The proof focusses on the upper left section H' of the convex hull; the proof for the three remaining parts of $H(S)$ is completely analogous. Let us denote the integer coordinates of the non-degenerate vertices of H' as (x_i, y_i), ordered by x_i, and define $(u_i, v_i) = (x_i - x_{i-1}, y_i - y_{i-1})$. Let us write $h = |H'|$. Without loss of generality we assume that $(x_0, y_0) = (0, 0)$ and that $(x_h, y_h) = (N_-, N_+)$. It holds that the slopes v_i/u_i of the consecutive edges of H' are positive and strictly monotonically decreasing. Following Lemma 1, we know that $|H'|$ is maximized if the (u_i, v_i)'s form a Farey set. Writing $Q = \sum_{i=1}^{h} u_i + v_i$, Lemma 2 tells us that $h \in O\left(Q^{2/3}\right)$. Finally,

$$\begin{aligned} Q = \sum_{i=1}^{h} u_i + v_i &= \sum_{i=1}^{h} x_i - x_{i-1} + y_i - y_{i-1} && (7) \\ &= x_h + y_h && (8) \\ &= N_- + N_+ = N \ , && (9) \end{aligned}$$

and hence we find that $|H(S)| \in O\left(N^{2/3}\right)$. □

What the above property tells us, is that in a two-dimensional coverage space, no matter how many candidate stamps points (corresponding to features) there are, the number of convex hull points that we ultimately need to consider to find an optimum, depends only on the number of examples in the dataset, and moreover, that this number of points is *sublinear* in the number of examples.

Note that this is in fact a worst-case result over all datasets with N examples, for all possible instantiations of N_+ and N_-. Moreover, in practice we can often expect the convex hull to be quite small. This is exemplified, for instance, by the fact that in expectation, the size of the convex hull of a random uniformly distributed set in the unit square grows only logarithmically in the number of points [Rényi and Sulanke, 1963].

Corollary 1. Given a set of points S in a c-dimensional coverage space $\times_{i=1}^{c}[0, N_i]$, the size of the convex hull $H(S)$ is bounded by $O\left(\left({}^{N}/_{c}\right)^{c-4/3}\right)$.

Proof. Consider all points $s \in H(S)$ for which the last $c-2$ coordinates $n_3, \ldots, n_c$ are fixed. This set of points must form a convex polygon, and therefore its size is bounded by $O\left((N_1+N_2)^{2/3}\right)$. It follows that $H(S) \in O\left(\prod_{i=3}^{c} N_i \cdot (N_1+N_2)^{2/3}\right) = O\left(\left({}^{N}/_{c}\right)^{c-4/3}\right)$. □

Property 3. Given a nominal attribute A, and a discrete target with c classes, in a dataset with N examples, the maximum number of values in the domain of A with a distinct class distribution is bounded by

$$|dom\,(A)| \in O\left(N^{{}^{c}/_{c}+1}\right) .$$

Proof. Follows from applying Lemma 2 to the class counts of each attribute value. □

The crucial requirement in the above properties, is that the coverage space be discrete. If one or several of the dimensions are continuous, Property 2 is no longer valid. If only one dimension is continuous, the complexity of Property 2 increases to $O\,(N)$, where N is the size of the discrete dimension. Otherwise, in the worst case all points in S can lie on its convex hull $H(S)$ (for instance, if the points lie on a sphere) and so $|H(S)|$ can grow arbitrarily large, independent of N. However, this would require that the coordinates of the points have unbounded precision, which in practice is not the case. If we use, say 32 bit precision reals (or integers), each dimension can in fact be seen as discrete (of size four billion); Property 2 then tells us that the hull size in two dimensions is bounded by $O\left((2\cdot 2^{32})^{2/3}\right) = O\left(2^{22}\right)$, roughly four million, which is still a quite reasonable worst case for large datasets. We show in Section 7 that the algorithms we present can in fact be used for data with a continuous target as well.

The above results tell us something about the number of points on the convex hull that we need to consider in the search for an optimal feature. However, they do not tell us how to consider only these points. To efficiently compute convex hulls in coverage space, we use the concept of a *Minkowski sum*.

Definition 11. The Minkowski sum of two sets $A, B \subseteq \mathbb{R}^k$ is defined as

$$A \oplus B = \{a + b \mid a \in A, b \in B\} .$$

Figure 3 illustrates the Minkowski sum of two convex polygons. The sum of the bottom left vertices of A and B, for instance, is the bottom left vertex of $A \oplus B$.

The following useful properties with regards to Minkowski sums and convex hulls can be observed.

Property 4. For two finite sets A and B, it holds that

$$\begin{aligned} H(A \cup B) &\subseteq H(A) \cup H(B) , \\ CH(A \oplus B) &= CH(A) \oplus CH(B) , \\ |H(A \oplus B)| &\leq |H(A)| + |H(B)| . \end{aligned}$$

In the last case, equality holds if and only if all vertices are non-degenerate.

Given two convex polygons in the plane, the convex hull of their Minkowski sum is constructed as follows. Assuming the vertices (and edges) of A and B are sorted, say clockwise, the corresponding edge lists are merged, sorted by their slopes. Thus, we can compute the convex hull of the Minkowski sum of two convex polygons A and B, in

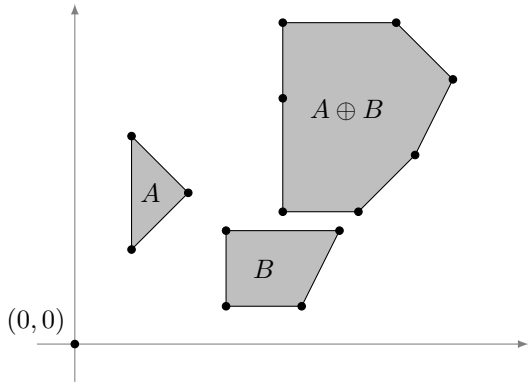


Figure 3. The Minkowski sum of two convex polygons. Each point in $A \oplus B$ is the sum of a point in A and a point in B.

$O(|A| + |B|)$ time. In Figure 3, each edge of $A \oplus B$ corresponds to an edge of either A or B. The resulting sum contains one degenerate hull point and hence $H(A \oplus B)$ contains six out of a maximum of seven points. In higher dimensions, based on Property 4, we can see that the convex hull of two convex polytopes A and B can be constructed by considering all possible pairs of vertices, taking $O(|A|\,|B|)$ time. This results in a superset of the convex hull, which is consequently computed in $O(|A|\,|B| \log |A|\,|B|)$ time in three dimensions, or $O\left((|A|\,|B|)^{\lfloor \frac{d}{2} \rfloor}\right)$ time in general.

5. Interval Features

In this section, we present a general algorithm to find a binary feature of the form $A \in]l, r]$ maximizing a convex quality measure φ, where A is a numeric attribute and l, r are taken from a set of candidate split points T, and this in arbitrary coverage spaces. The basic idea is to disregard the points in coverage space that are not on the convex hull, but instead to attempt to directly construct the hull using a divide-and-conquer strategy. We explore the computational complexity of the algorithm, focussing especially to the important two dimensional case, and furthermore present a separate algorithm for additive quality measures with linear time complexity.

5.1. Algorithm

We assume that the set of candidate split points $T = \{t_i\}_i$ is presented as a parameter to the algorithm. It may consist of all distinct values occurring in the data; alternatively, the user may want to restrict them for computational reasons, easy interpretation (e.g., only using multiples of 100), or because the user is interested in specific split points. Furthermore, we assume that T is sorted. As such, we omit the sorting time $O(|T| \log |T|)$ from our complexity analysis, focussing solely on the algorithm itself. We justify this by noting that the attribute only needs to be sorted once when the data is read; this order can be reused for every recursive subgroup specialization or decision tree split.

The split points in T induce a partition of $dom(A)$ consisting of *base intervals* $]t_i, t_{i+1}]$, defined by consecutive split points. First of all, some of the endpoints can be

pruned beforehand: following Elomaa and Rousu [2004], as a pre-processing step we can remove any endpoint t_i for which the adjacent base intervals $]t_{i-1}, t_i]$ and $]t_i, t_{i+1}]$ exhibit the same target distribution, since these points will never participate in an optimal solution. This pre-processing step can easily be implemented in linear time in $|T|$, and has the potential to considerably reduce the number of split points. However, in the worst case it may still hold that after pruning $|T| \in O(N)$, leaving a straightforward quadratic algorithm still infeasible for large datasets.

The BESTINTERVAL algorithm, given here as Algorithm 1, constructs the convex hull of all interval stamp points in coverage space. Let us write the search space as

$$I = \{\,]t, t'] \mid t, t' \in T, t < t' \} \ . \tag{10}$$

First, note that any interval $]t, t']$ can be expressed as the difference of the two half-intervals $]-\infty, t']$ and $]-\infty, t]$. To obtain the convex hull of I, we decompose I into disjoint subsets, and compute the convex hulls of these subsets, using Minkowski differences of sets of half-intervals. Note that I itself cannot directly be written as a single Minkowski difference of half-intervals due to the constraint $t < t'$.

For the purpose of exposition, assume $|T|$ is a power of two. Let T^k denote the partition of the set $\{\,]-\infty, t] \mid t \in T\}$ into 2^k equal-size bins, for $k = 1, \ldots, \log|T|$ (where the base of the logarithm is 2). Further, let us write T^k_ℓ for the ℓ-th bin, for $\ell = 1, \ldots, 2^k$. Then we define $I^k_\ell = T^k_{2\ell} \ominus T^k_{2\ell-1}$, that is, (slightly abusing notation),

$$I^k_\ell = \left\{\,]t, t'] \mid t \in T^k_{2\ell-1}, t' \in T^k_{2\ell} \right\} \ , \tag{11}$$

where ℓ ranges from 1 to 2^{k-1}. By construction, it holds that $t < t'$. We can now decompose I into disjoint subsets as follows,

$$I = \bigcup_{k=1}^{\log|T|} \bigcup_{\ell=1}^{2^{k-1}} I^k_\ell \ . \tag{12}$$

Now we compute the convex hull of the stamp points in coverage space that correspond to the intervals in I. For ease of notation, we identify intervals with their stamp points. Let us use the shorthand H^k_ℓ for $H(T^k_\ell)$. Using Property 4 we obtain

$$H(I) \subseteq \bigcup_{k=1}^{\log|T|} \bigcup_{\ell=1}^{2^{k-1}} H(I^k_\ell) \tag{13}$$

$$= \bigcup_{k=1}^{\log|T|} \bigcup_{\ell=1}^{2^{k-1}} H(T^k_{2\ell} \ominus T^k_{2\ell-1}) \tag{14}$$

$$= \bigcup_{k=1}^{\log|T|} \bigcup_{\ell=1}^{2^{k-1}} H(H^k_{2\ell} \ominus H^k_{2\ell-1}) \ . \tag{15}$$

The BESTINTERVAL algorithm computes the convex hulls of sets of half-intervals in a bottom-up fashion. That is, we start with the partition $T^{\log|T|}$, where each bin contains a single half-interval (lines 6–7). Thus, it trivially holds that $H^k_\ell = T^k_\ell$ for $k = \log|T|$. Then, for each subsequent k down to 1, using Property 4 and the fact that

$$T^k_\ell = T^{k+1}_{2\ell-1} \cup T^{k+1}_{2\ell} \ , \tag{16}$$

we can compute H^k_ℓ by combining $H^{k+1}_{2\ell-1}$ and $H^{k+1}_{2\ell}$ (lines 16–17). For each adjacent

Algorithm 1: BESTINTERVAL(A, T, φ)

Input: numeric attribute A, sorted endpoints T, convex quality measure φ
Output: interval $]l, r]$ with $l, r \in T$ maximizing $\varphi(A \in]l, r])$

```
1  ]l, r] ← ]−∞, ∞]
2  φ_max ← φ(]l, r])
3  foreach threshold t_i in T do
4      compute stamp point s_i for A ∈ ]−∞, t_i]
5  k ← log |T|
6  for ℓ = 1 to 2^k do
7      H^k_ℓ ← {]−∞, t_ℓ]}
8  while k ≥ 1 do
9      for ℓ = 1 to 2^{k−1} do
10         compute H(I^k_ℓ) = H(H^k_{2ℓ} ⊖ H^k_{2ℓ−1})
11         foreach interval ]t_i, t_j] in H(I^k_ℓ) do
12             compute φ_ij = φ(s_j − s_i)
13             if φ_ij > φ_max then
14                 φ_max ← φ_ij
15                 ]l, r] ← ]t_i, t_j]
16     for ℓ = 1 to 2^{k−1} do
17         H^{k−1}_ℓ ← H(H^k_{2ℓ} ∪ H^k_{2ℓ−1})
18     k ← k − 1
19 return ]l, r]
```

pair of convex hulls of half-intervals, the convex hull of their Minkowski difference is computed to obtain intervals (line 10), and for each of the intervals it is checked whether it maximizes φ (lines 11–15). We remark that as such, we actually compute a superset of $H(I)$, but the relative overhead will be quite small.

5.1.1. Complexity analysis

Property 5. For a binary target, the time complexity of BESTINTERVAL is $O(N + |T|)$, that is, linear in the number of examples and split points.

Proof. The coordinates of the stamp points of all half-intervals $]-\infty, t]$ are obtained in $O(N + |T|)$ time with a single scan over that data (lines 3–4). The two major computational bottlenecks are the computation of the Minkowski difference $H(I^k_\ell)$ on line 10, and H^{k-1}_ℓ on line 17. Every H^k_ℓ corresponds to a convex polygon in coverage space. By construction, these polygons can be stored in sorted order. As a result, the computation of $H(I^k_\ell)$ is linear in $\left|I^k_\ell\right|$, since it is computed as the Minkowski sum of two convex polygons. The computation of H^{k-1}_ℓ is linear in $\left|H^{k-1}_\ell\right|$, by applying the Graham scan algorithm to the union of the two sorted polygons $H^k_{2\ell}$ and $H^k_{2\ell-1}$ [Graham, 1972]. Let us write the number of examples covered by I^k_ℓ as N^k_ℓ, then $\sum_\ell N^k_\ell = N$. For a fixed k and ℓ, Property 2 tells us that $\left|H^k_\ell\right| = O\left((N^k_\ell)^{2/3}\right)$. Using Jensen's inequality and the

fact that the function $(\cdot)^{2/3}$ is concave, it holds that

$$\sum_{\ell=1}^{2^k} (N_\ell^k)^{2/3} \leq 2^k \left(\frac{\sum_\ell N_\ell^k}{2^k} \right)^{2/3} . \tag{17}$$

Thus, we find that for a fixed k we need

$$\sum_{\ell=1}^{2^k} O\left(\left|H_\ell^k\right|\right) \leq \sum_{\ell=1}^{2^k} O\left((N_\ell^k)^{2/3}\right) \leq O\left(2^{k/3} N^{2/3}\right) \tag{18}$$

computations. Finally, summing over all $k = 1, \ldots, \log |T|$ we obtain

$$O\left(\sum_{k=1}^{\log|T|} 2^{k/3} N^{2/3} \right) = O\left(N^{2/3} |T|^{1/3} \right) . \tag{19}$$

Hence, for a binary target the total time complexity of the algorithm is

$$O\left(N + |T| + N^{2/3} |T|^{1/3} \right) = O\left(N + |T| \right) . \tag{20}$$

□

5.2. Algorithm for Additive Quality Measures

Algorithm 2 finds an optimal interval feature with respect to a linear quality measure φ. It has linear time complexity (for any type of target), while also being conceptually simpler than Algorithm 1, and can be performed in a single pass over the data.

The algorithm iterates over the endpoints (line 5), maintaining the best interval encountered so far. Assume that at endpoint t we have two intervals $]t_1, t]$ and $]t_2, t]$, such that the former has a higher quality. Then the following property states that for any future extensions $]t_1, t']$ and $]t_2, t']$, where $t' > t$, their relative difference in quality remains the same, even though their qualities might change.

Property 6. Given a linear quality measure φ, it holds that for any interval endpoints t_1, t_2, t, t' such that $t_1, t_2 < t < t'$,

$$\begin{array}{c} \varphi(A \in]t_1, t]) > \varphi(A \in]t_2, t]) \\ \Updownarrow \\ \varphi(A \in]t_1, t']) > \varphi(A \in]t_2, t']) \end{array} .$$

Proof. Follows from the additivity of φ and the fact that $]t_i, t'] =]t_i, t] \cup]t, t']$. □

Therefore, at each endpoint t_i only one candidate interval needs to be maintained. Every time a new right endpoint t_i is considered, a new left endpoint t_{i-1} is available, which is checked on lines 6–10. To be able to compare candidate intervals for different right hand sides, we use the quality of their maximal extension, i.e., $]t_{i-1}, \infty[$.

Property 7. The time complexity of BESTINTERVALADDITIVE is $O\left(N + |T|\right)$.

Algorithm 2: BESTINTERVALADDITIVE(A, T, φ)

Input: numeric attribute A, sorted set of endpoints T, additive quality measure φ
Output: interval $]l, r]$ with $l, r \in T$ maximizing $\varphi(A \in]l, r])$

$]l, r] \leftarrow]-\infty, +\infty]$
$\varphi_{\max} \leftarrow \varphi(A \in]l, r])$
$h_{\max} \leftarrow -\infty$
$t_{\max} \leftarrow -\infty$
foreach t_i **in** T in increasing order **do**
  compute stamp point s_{i-1} for $A \in]t_{i-1}, \infty[$
  $h \leftarrow \varphi(s_{i-1})$
  if $h > h_{\max}$ **then**
    $h_{\max} \leftarrow h$
    $t_{\max} \leftarrow t_{i-1}$
  if $\varphi(A \in]t_{\max}, t_i]) > \varphi_{\max}$ **then**
    $]l, r] \leftarrow]t_{\max}, t_i]$
    $\varphi_{\max} \leftarrow \varphi(A \in]t_{\max}, t_i])$
return $]l, r]$

6. Value Set Features

Here we present a general algorithm to find a binary feature of the form $A \in V$ maximizing a convex quality measure φ, where A is a nominal attribute and V is a subset of the domain of A, and this in arbitrary coverage spaces. As in the previous section, the basic idea is to disregards the points in coverage space that are not on the convex hull, but instead to attempt to directly construct the hull using a divide-and-conquer strategy. We explore the computational complexity of the algorithm, focussing on the important two-dimensional case, and present a separate algorithm for additive quality measures with linear time complexity. We also look at the algorithm by Breiman et al. [1984] for the two-dimensional case, pointing out that its complexity can be slightly lowered.

6.1. Algorithm

Similar to the interval algorithm from the previous section, we assume that the domain of the nominal attribute is known to the algorithm. Let us write $d = |dom\,(A)|$ for the size of the domain of A. If the domain is not known in advance, it must be determined from the data, requiring $\Omega\,(d \log d)$ time. However, since this needs to be done only once when the data is read, and we focus only on the algorithm itself, we omit this term from the complexity analysis later on.

Let us consider the stamp points of all features $A \in V$ in coverage space. Since φ is convex, we know that the subset V maximizing φ lies on the convex hull of these points. The BESTVALUESET algorithm, given as Algorithm 3, makes use of the fact that the values $v \in dom\,(A)$ are mutually exclusive. Hence, the stamp point of any value set $V \subseteq dom\,(A)$ can be expressed as the sum of the stamp points of its individual values $v \in V$. Consequently, the power set of $dom\,(A)$, denoted as P, can be written as a Minkowski sum as follows (identifying value sets with their stamp points),

$$P = \oplus_{v \in dom(A)} \{0, v\} \ ,$$

where 0 denotes an empty value, with stamp point in the origin. As such, we can compute the convex hull $H(P)$ in a bottom-up fashion. Assume again for the sake of exposition that $d = |dom\,(A)|$ is a power of two, and let us consider the values in some arbitrary order $v_1, \ldots, v_d$. We partition V into 2^k subsets of equal size, and we denote by V_ℓ^k the ℓ-th subset of values of this partition, where k ranges from 0 to $\log d$, and ℓ ranges from 1 to 2^k. Further, denote by P_ℓ^k the powerset of V_ℓ^k, and let us use the shorthand $H_\ell^k = H(P_\ell^k)$. Applying Property 4, we find that

$$H(P_\ell^{k-1}) = H(P_{2l-1}^k \oplus P_{2l}^k) \tag{21}$$

$$= H(H_{2l-1}^k \oplus H_{2l}^k)\ . \tag{22}$$

Thus, starting from $H_\ell^{\log d}$, we can iteratively build up the convex hull $H_1^0 = H(P)$.

6.1.1. Complexity analysis

Property 8. For a binary target, the worst case time complexity of BESTVALUESET is $O\,(N + d)$, i.e., linear in the number of examples and the size of the domain.

Proof. The stamp point coordinates can be computed in $O\,(N + d)$ time (lines 3–4). Let us write N_ℓ^k for the number of examples covered by V_ℓ^k. Based on Property 2, we have that $\left|H_\ell^k\right| = O\left({N_\ell^k}^{2/3}\right)$. Hence, we find that we can compute H_ℓ^{k-1} (line 10) in linear time with respect to $\left|H_\ell^k\right| = O\left({N_\ell^k}^{2/3}\right)$. Due to the function $(.)^{2/3}$ being convex, and using Jensen's inequality, we find

$$\sum_{l=1}^{2^k} \left|H_\ell^k\right| = \sum_{l=1}^{2^k} O\left({N_\ell^k}^{2/3}\right) \leq O\left(2^{k/3} N^{2/3}\right)\ . \tag{23}$$

Summing over all k, we obtain

$$\sum_{k=1}^{\log d} O\left(2^{k/3} N^{2/3}\right) = O\left(N^{2/3} d^{1/3}\right)\ , \tag{24}$$

resulting in a final complexity of $O\left(N + d + N^{2/3} d^{1/3}\right) = O\,(N + d)$. □

6.1.2. CART algorithm

A different algorithm with the same goal was proposed by Breiman et al. [1984] in the context of decision trees, reproduced here as BESTVALUESETCART (Algorithm 4). First, all values are sorted based on their positive/negative ratio. Then the hull is incrementally constructed by adding the values one by one in decreasing order with respect to this ratio. For each intermediate value set, the quality of the corresponding feature is computed, and in the end the best one is reported. Note that this only constructs the upper part of the convex hull; the lower part is analogously formed by considering the values in increasing order. The upper hull thus forms a chain, and the lower hull consists of the upper hull's complements. Therefore, for symmetric quality measures computing the lower convex hull is redundant.

It is easy to see that due to the sorting step the computational complexity of the BESTVALUESETCART algorithm is $O\,(N + d \log d)$. For attributes with a large domain, that is, with $d = O\,(N)$, the time complexity can approach $O\,(N \log N)$. We point out

Algorithm 3: BESTVALUESET(A, φ)

Input: nominal attribute A, convex quality measure φ
Output: value set $V \subseteq dom\,(A)$ maximizing $\varphi(A \in V)$

$V_{\max} \leftarrow dom\,(A)$
$\varphi_{\max} \leftarrow \varphi(V_{\max})$
foreach value v **in** $dom\,(A)$ **do**
  compute stamp point n_v of $A = v$
$k \leftarrow \log |dom\,(A)|$
for $\ell = 1$ **to** 2^k **do**
  $H_\ell^k = \{n_{v_\ell}\}$
while $k \geq 1$ **do**
  for $\ell = 1$ **to** 2^{k-1} **do**
    compute $H_\ell^{k-1} = H\left(H_{2\ell}^k \oplus H_{2\ell+1}^k\right)$
  $k \leftarrow k - 1$
foreach V **in** H_1^0 **do**
  if $\varphi(V) > \varphi_{\max}$ **then**
    $V_{\max} \leftarrow V$
    $\varphi_{\max} \leftarrow \varphi(V)$
return $V_{\max}$

Algorithm 4: BESTVALUESETCART(A, φ)

Input: nominal attribute A, convex quality measure φ
Output: value set $V \subseteq dom\,(A)$ maximizing $\varphi(A \in V)$

$V_{\max} \leftarrow dom\,(A)$
$\varphi_{\max} \leftarrow \varphi(V_{\max})$
foreach value v **in** $dom\,(A)$ **do**
  compute n_v^+, n_v^- for $A = v$
$\mathbb{V} \leftarrow \bigcup_{n_v^+/n_v^-} \left\{\left\{v' \mid n_{v'}^+/n_{v'}^- = n_v^+/n_v^-\right\}\right\}$
$V' \leftarrow \emptyset$
$(n^+, n^-) \leftarrow (0, 0)$
foreach V **in** $\mathbb{V}$ decreasing w.r.t. n_V^+/n_V^- **do**
  $V' \leftarrow V' \cup V$
  $(n^+, n^-) \leftarrow (n^+ + n_V^+, n^- + n_V^-)$
  if $\varphi(n^+, n^-) > \varphi_{\max}$ **then**
    $V_{\max} \leftarrow V'$
    $\varphi_{\max} \leftarrow \varphi(n^+, n^-)$
return $V_{\max}$

that a straightforward optimization can be applied, by noting that we do not need to check degenerate points on the convex hull. Thus, rather than checking all values v individually, we can group all values with the same ratio n_v^+/n_v^- (line 5). For large d, it is quite likely that several values have the same ratio, and hence this can reduce the number of evaluations (lines 8–13). More importantly, when sorting, the relative order of values

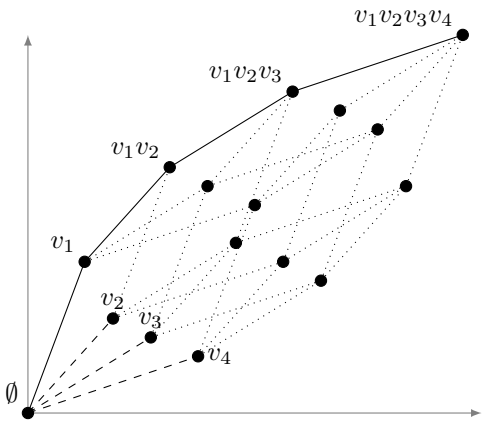


Figure 4. Illustration of the BESTVALUESETCART algorithm, for $d = 4$. The upper convex hull (full lines) is constructed by incrementally adding the individual lines sorted by their angle (dashed lines).

with the same ratio is irrelevant. With this observation, we prove that the algorithm can be made to run in linear time, by using a sorting algorithm that properly takes duplicates into account. The basic idea is that even for large d, sorting is always bounded by $O(N)$.

Property 9. The BESTVALUESETCART algorithm can be implemented to run in $O(N + d)$ time.

Proof. Let us consider the set of all *distinct* positive/negative ratios

$$\rho = \left\{ n_v^+ / n_v^- \mid v \in dom(A) \right\} , \tag{25}$$

and the empirical distribution μ over ρ, defined as

$$\mu(r) = \left| \left\{ v \mid n_v^+ / n_v^- = r \right\} \right| / d \tag{26}$$

for $r \in \rho$. A sorting algorithm that properly deals with duplicates takes $O(\eta)$ time per entry on average, where

$$\eta = -\sum_r \mu(r) \log \mu(r) \tag{27}$$

is the entropy of its input data [Sedgewick and Bentley, 2002]. We will show that $d \cdot \eta \in O(N)$, that is, the complexity of sorting is bounded by the number of examples. Let $\delta = |\rho| \leq d$, and write $a = d/\delta$. Now, the entropy η is maximized if μ is the uniform distribution, i.e., if $\mu(r) = 1/\delta$ for all ratios r, and there are a values for each distinct ratio. Property 2 tells us that in the worst case $\delta \in O\left((N/a)^{2/3}\right)$, as such we find that

$$d \cdot \eta \leq d \cdot \log \delta \in O\left(a(N/a)^{2/3} \log(N/a)^{2/3} \right) = O(N) . \tag{28}$$

Hence, the total time complexity of the algorithm is $O(N + d)$. □

Algorithm 5: BESTVALUESETADDITIVE(A, φ)

Input: nominal attribute A, additive quality measure φ
Output: value set $V \subseteq dom\,(A)$ maximizing $\varphi(A \in V)$

```
foreach value v in dom(A) do
    compute stamp point s_v for A = v
V_max ← {v ∈ dom(A) | φ(s_v) ≥ 0}
return V_max
```

6.2. Algorithm for Additive Quality Measures

Algorithm 5 can be seen as a simplification of Algorithm 4 for an additive quality measure. Using the additivity of φ, the following property shows that updating a value set V with a value v having a positive quality, increases the total quality.

Property 10. Let $V \subseteq dom\,(A)$ and $v \in dom\,(A)$ with $v \notin V$. If $\varphi(A = v) \geq 0$ then

$$\varphi(A \in V \cup \{v\}) \geq \varphi(A \in V)\ .$$

Equality holds if and only if $\varphi(A = v) = 0$.

Proof. Follows directly from the additivity of φ. □

Therefore, it is not necessary to construct and check the entire convex hull. Instead, we can directly construct the optimum: the value set that maximizes φ is the union of all attribute values with non-negative quality.

Property 11. The time complexity of BESTVALUESETADDITIVE is $O\,(N + d)$.

7. Experimental Evaluation

In this section, we demonstrate the efficiency of the presented algorithms, furthermore show that using a richer pattern language results in subgroups of higher quality, and present results on a real-world medical insurance dataset.

Table 1 presents the characteristics of the datasets we used. The last column shows which attribute is the target, and whether it is nominal or numeric. We created six synthetic datasets and used five publicly available benchmark datasets. The *Independent num* and *Independent nom* datasets consists of one numeric (respectively nominal) attribute and an independent binary target c, with probability 50%. The values of the numeric attribute are all distinct, the domain of the nominal attribute is of size $N/10^3$. The *Farey num* and *Farey nom* datasets both correspond to the worst case convex hull, proportional to $N^{2/3}$; in the numeric case, this is the number of (non-pruneable) split points, in the nominal case, this is the domain size. The *Gaussian target* dataset has a numeric attribute A with domain $[0, 10]$, and a numeric target that equals $\mathcal{N}(5, 1)(A)$. The *Diverse domain* dataset has a nominal attribute with a domain that grows linearly with N, where the target mean (ranging between 0 and 10) of value v_i is a linear function of i, with standard deviation 1. For each type of synthetic dataset, we generated instances with sizes ranging between 10^6 and 10^7. The benchmark datasets we used are the following. The *Ames housing* dataset is available from the Journal of Statistics Education data archive [De Cock, 2011]. From the UCI ML Repository we used the *Adult*, *CMC*, and *Mushroom* datasets [Frank and Asuncion, 2010]. Finally, the *Wine*

Table 1. Main characteristics of the synthetic, benchmark, and real-world datasets used in the experiments. Shown are the number of records N, the number of nominal and numeric attributes, and the target.

		attributes		
Dataset	# records	nominal	numeric	target
Independent num	10^6–10^7	0	1	$c = 1$ (nom)
Independent nom	10^6–10^7	1	0	$c = 1$ (nom)
Farey num	10^6–10^7	0	1	$c = 1$ (nom)
Farey nom	10^6–10^7	1	0	$c = 1$ (nom)
Gaussian target	10^6–10^7	0	1	t (num)
Diverse domain	10^6–10^7	1	0	t (num)
Adult	48 842	8	6	$income > 50k$ (nom)
Ames housing	2 930	45	34	$price$ (num)
CMC	1 473	7	2	$method = 1$ (nom)
Mushroom	8 124	22	0	$poisonous$ (nom)
Wine	9 600	5	4	$saleprice$ (num)
Achmea	2 117 013	8	8	$pharmacy$ (nom)

dataset is composed of observations derived from 10 years of tasting ratings reported in the Wine Spectator Magazine [Costanigro et al., 2009].

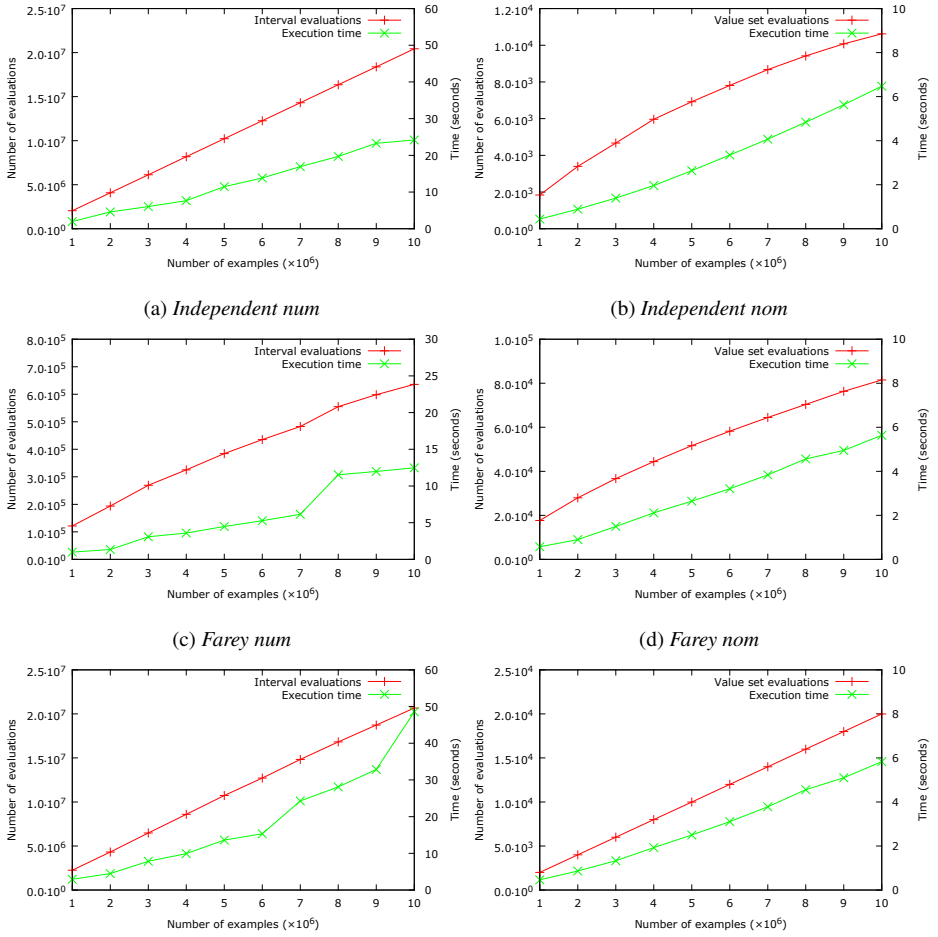
We integrated the algorithms into the open-source subgroup discovery tool Cortana [Meeng and Knobbe, 2011].[2] All experiments were executed on a system with a 2.4GHz dual quad core processor and 24GB of memory, running Linux. For the synthetic datasets, the minimum support threshold was set to 1. For each benchmark dataset, a minimum support of 1% and maximum refinement depth of 3 were used, the search strategy was beam search with a beam width of 100. The multi-threading option of Cortana was not used.

7.1. Performance

Figure 5 shows the results of performance experiments with the BESTINTERVAL and BESTVALUESET algorithms on the synthetic datasets. We measure the number of feature evaluations and the execution time (excluding reading the data) of each algorithm, as a function of the number of dataset records. The quality measures we used are Information Gain for nominal targets, and standardized z-score for numeric targets. For numeric attributes, all values occurring in the data were used as candidate split points. The reported execution times are averaged over three runs per dataset.

First, note that the algorithms can handle datasets with millions of records in just a few seconds, and even for the largest datasets require less than a minute. The number of considered features (intervals or value sets) grows at most linearly with respect to dataset size in all figures. In particular, the number of evaluations in Figures 5b and 5d grow slightly sublinearly, in accordance with Property 2. Further, we see that the execution times scale proportionally to the number of feature evaluations, and linearly in the number of examples.

[2] The Cortana tool can be downloaded from http://datamining.liacs.nl/cortana.html.



(a) *Independent num* (b) *Independent nom*

(c) *Farey num* (d) *Farey nom*

(e) *Gaussian target* (f) *Diverse domain*

Figure 5. Number of feature evaluations and execution times in seconds of the BESTINTERVAL and BESTVALUESET algorithms as a function of the number of dataset records, for the synthetically generated datasets.

7.2. Quality

Table 2 contains the results of qualitative experiments on the benchmark datasets, for several quality measures. We run the Cortana subgroup discovery tool for several quality measures, using beam search with a beam width of 100 and a maximal search depth of 3. We calculate the average quality of the top 10 subgroups, allowing different types of feature descriptions for nominal and numeric attributes, that is, standard equalities and inequalities, value sets instead of single values, intervals instead of inequalities, and both value sets and intervals. As to be expected, the table shows that using intervals or value sets as descriptions increases the average quality of the discovered subgroups, whereas using both types of descriptions increases average subgroup quality the most. This is a

Table 2. Average score of the top 10 discovered subgroups using either standard descriptions (equalities and inequalities), value sets, intervals, or both, for various quality measures, with minimum support=1%, search depth=3, and beam width=100.

Dataset	φ	$\leq \mid =$	$\leq \mid \{\}$	$[] \mid =$	$[] \mid \{\}$
Adult	WRAcc	0.0948	0.1019	0.1006	**0.1030**
	Bin	0.1418	0.1854	0.1798	**0.1886**
	IG	0.1529	**0.1887**	0.1708	**0.1887**
Ames housing	z-score	31.158	32.473	35.682	**35.683**
CMC	WRAcc	0.0499	0.0502	0.0504	**0.0506**
	Bin	0.1376	0.1396	0.1383	**0.1411**
	IG	0.0712	0.0741	0.0803	**0.0819**
Mushroom	WRAcc	0.1910	**0.2283**	0.1910	**0.2283**
	Bin	0.2853	**0.3529**	0.2853	**0.3529**
	IG	0.5046	**0.8932**	0.5046	**0.8932**
Wine	z-score	33.021	39.134	67.485	**70.078**

direct result of an increased feature search space, allowing for higher quality subgroups to be discovered.

7.3. Real-World Demonstration

We describe an application of our methods to the field of fraud detection in a medical insurance dataset, obtained from Achmea, the largest health insurance company in the Netherlands. We aim to illustrate the value of the proposed extension in expressive power, and additionally demonstrate that a scalable solution has been achieved that provides accurate results on large data, in an interactive setting. The results pertain to a dataset describing pharmacies. In total, some 2.1 million records of individual medication are available, involving 100 pharmacies: one pharmacy under investigation and 99 of its peers (comparable pharmacies). A total of 16 attributes, both nominal and numeric, describe each prescription, and an additional binary target indicates the pharmacy under investigation. This pharmacy contributes 11 380 records (0.54%) to the dataset. The attributes provide information about the patient involved, such as their birth date and gender, as well as various pharmaceutical categories and the dosage and amount of medication. With the aim of characterizing deviating claim behaviour at this pharmacy, and potentially understanding population differences at this pharmacy, we look for interesting subgroups using Cortana, including the extensions described in this article.

For the WRAcc measure, some interesting individual conditions found were as follows:

number of pieces $\in]29, 204]$

day of the week $\in \{$*Monday, Tuesday, Thursday*$\}$

These conditions cover roughly 1.5 and 1.2 million examples, respectively. A more specific and accurate result, discovered at depth $d = 2$ is as follows:

cost $\in]8.69, 512.12] \wedge$ *date of birth* $\in]$*December 18, 1916, June 24, 1959*$]$

This result, which covers about a million records with a target share of 0.87%, demonstrates nicely how our method finds the optimal boundaries of intervals with much detail, and furthermore does so in a dynamic fashion: the optimal birth dates were computed on

the subset of records that satisfy the first condition. A similar result involving value sets is as follows:

$$\textit{delivery-code} \in \{1, 3, 4, 5, 10, 11, 15, 16, 20, 21\} \wedge \textit{cost} \in]7.49, 512.12]$$

The 10 codes, out of a total of 20, describe several legal regulations concerning compensation for complicated preparation, delivery at night, and so on. The codes in this value set all relate to delivery during regular office hours, as opposed to the codes for evenings, nights and Sundays.

The data also includes attributes of the Anatomical Therapeutic Chemical Classification (ATC) system, which is a multi-level classification of drugs. Theses attributes, providing different levels of the ATC hierarchy, tend to produce large value sets of possibly hundreds of codes. Although such value sets may be interesting to the expert, they do not necessarily represent high-level insight into the domain. For this reason, we have excluded nominal attributes with cardinalities above 100.

Despite the size of the dataset and the numeric data involved, producing a set of subgroups at depth $d = 2$ took only 12 seconds; a depth $d = 3$ run took 50 seconds. A minimum coverage of 1 000 records and a beam-width of 100 were used.

8. Conclusions

We presented efficient algorithms for finding optimal binary features in labeled data with respect to a convex quality measure. These features have descriptions in the form of intervals for numeric attributes, and value sets for nominal attributes. By directly constructing the convex hull of the set of all feature stamp points in coverage space, we can disregard vast portions of the search space. We presented some general theoretical properties of convex hulls in coverage spaces, and showed that in the important case of a binary target, the time complexity of the presented algorithms is linear in the number of examples. Experiments on synthetic, benchmark, and real-world data, demonstrated that as such, even for large datasets we can efficiently discover high quality features with rich descriptions.

Acknowledgements

This research is partially supported by the Netherlands Organization for Scientific Research (NWO) under project nr. 612.065.822 (Exceptional Model Mining), and by a Postdoc grant from the Research Foundation—Flanders (FWO).

References

Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on management of data*, pages 207–216, 1993.

Martin Atzmüller and Frank Puppe. SD-Map–A fast algorithm for exhaustive subgroup discovery. In *Proceedings of PKDD*, pages 6–17, 2006.

Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and regression trees*. Chapman & Hall/CRC, 1984.

Toon Calders, Nele Dexters, Joris J.M. Gillis, and Bart Goethals. Mining frequent itemsets in a stream. *Information Systems*, 2013. in press.

Philip A. Chou. Optimal partitioning for classification and regression trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):340–354, 1991.

John H. Conway and Richard K. Guy. Farey fractions and Ford circles. *The Book of Numbers*, pages 152–154, 1996.

Marco Costanigro, Ron C. Mittelhammer, and Jill 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*, 24:1117–1135, 2009.

Dean De Cock. Ames, ia real estate data, 2011. URL `http://www.amstat.org/publications/jse/`.

Tapio Elomaa and Juho Rousu. Efficient multisplitting revisited: Optima-preserving elimination of partition candidates. *Data Mining and Knowledge Discovery*, 8(2): 97–126, 2004.

Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1022–1029, 1993.

Andrew Frank and Arthur Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

Takeshi Fukuda, Yasuhido Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Mining optimized association rules for numeric attributes. *Journal of Computer and System Sciences*, 58(1):1–12, 1999.

Johannes Fürnkranz and Peter A. Flach. Roc 'n' rule learning—towards a better understanding of covering algorithms. *Machine Learning*, 58(1):39–77, 2005.

Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information processing letters*, 1(4):132–133, 1972.

Henrik Grosskreutz and Stefan Rüping. On subgroup discovery in numerical domains. *Data mining and knowledge discovery*, 19(2):210–226, 2009.

Wilhelmiina Hämäläinen. Kingfisher: an efficient algorithm for searching for both positive and negative dependency rules with statistical significance measures. *Knowledge and Information Systems*, 32(2):383–414, 2012.

Franciso Herrera, Cristóbal Carmona, Pedro González, and María del Jesus. An overview on subgroup discovery: foundations and applications. *Knowledge and Information Systems*, (29):495–525, 2010.

Branko Kavšek, Nada Lavrač, and Viktor Jovanoski. Apriori-sd: Adapting association rule learning to subgroup discovery. In *Proceedings of Intelligent Data Analysis (IDA)*, pages 230–241, 2003.

Willi Klösgen. *Handbook of Data Mining and Knowledge Discovery*. Oxford University Press, New York, 2002.

Marvin Meeng and Arno Knobbe. Flexible enrichment with Cortana – software demo. In *Proceedings of BeneLearn*, pages 117–119, 2011.

Petra K. Novak, Nada Lavrač, and Geoff I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *The Journal of Machine Learning Research*, 10:377–403, 2009.

James E. Nymann. On the probability that k positive integers are relatively prime. *Journal of Number Theory*, 4(5):469–473, 1972.

Franco P. Preparata and Michael I. Shamos. *Computational geometry: an introduction*. Springer, 1985.

Alfred Rényi and Rolf Sulanke. Über die konvexe hülle von n zufällig gewälten punkten. *Probability Theory and Related Fields*, 2:75–84, 1963.

Piotr Rzepakowski and Szymon Jaroszewicz. Decision trees for uplift modeling with single and multiple treatments. *Knowledge and Information Systems*, 32(2):303–327, 2012.

Robert Sedgewick and Jon Bentley. Quicksort is optimal. *Knuthfest, Stanford University, Stanford*, 2002.

Stefan Wrobel. An algorithm for multi-relational discovery of subgroups. In *Proceedings of Principles of Data Mining and Knowledge Discovery (PKDD)*, pages 78–87, 1997.

Dr. Michael Mampaey

Michael Mampaey works as a post-doctoral researcher at Bonn University. He got his Ph.D. from Antwerp University on the topic of Summarizing Data with Informative Patterns. The work presented here was produced while being a post-doctoral researcher at both Utrecht and Leiden university.

Dr. Siegfried Nijssen

Siegfried Nijssen received his PhD in computer science from the Universiteit Leiden, The Netherlands. He is a post-doctoral researcher at KU Leuven, Belgium, since 2006 and an assistant professor at the Universiteit Leiden since 2012. His research interests include pattern mining, constraint-based data mining, declarative data mining, and applications of these techniques in bioinformatics and chemoinformatics. He was co-chair of ECML PKDD 2013 and is a member of the editorial board of the Machine Learning journal.

Dr. Ad Feelders

Ad Feelders obtained his Ph.D. in 1993 from Tilburg University in The Netherlands. Currently he is an assistant professor at Utrecht University. His main research interests are Exceptional Model Mining and the exploitation of prior knowledge in data mining. He is a member of the editorial boards of *Intelligent Systems in Accounting, Finance and Management* and *Intelligent Data Analysis*.

Rob Konijn

Rob Konijn got his MSc. in Business Analytics at the VU University in Amsterdam. He is a Ph.D. candidate at Leiden University working in the Data Mining group. His research focuses mainly on outlier detection and Subgroup Discovery, applied to health insurance data. He is currently working at Achmea, the biggest health insurance company in the Netherlands.

Dr. Arno Knobbe

Arno Knobbe got his Ph.D. in Computer Science from Utrecht University, the Netherlands. The topic of his thesis was Multi-Relational Data Mining, a branch of Data Mining that deals with structured data in relational form. After working as a post-doctoral researcher in Utrecht, he started a research group on Data Mining at Leiden University. His current research is primarily focused on two topics: Subgroup Discovery and Mining Sensor Data. He initiated the Cortana SD tool, which features amongst others the work described in this paper.