Chapter 3

The Exceptional Model Mining Framework

Exceptional Model Mining [23, 25, 71] is a data mining framework that can be seen as a generalization of the Subgroup Discovery (SD) [50, 55, 114] framework. SD strives to find descriptions that satisfy certain user-specified constraints. Usually these constraints include lower bounds on the quality of the description ($\varphi(D) \geq lb_1$) and size of the induced subgroup ($|G_D| \geq lb_2$). More constraints may be imposed as the question at hand requires; domain experts may for instance request an upper bound on the complexity of the description. Most common SD algorithms traverse[1] the search space of candidate descriptions in a general-to-specific way: they treat the space as a lattice whose structure is defined by a *refinement operator* $\eta : \mathcal{D} \rightarrow 2^{\mathcal{D}}$. This operator determines how descriptions can be extended into more complex descriptions by atomic additions. Most applications (including ours) assume η to be a *specialization operator*: every description D_i that is an element of the set $\eta(D_j)$, is more specialized that the description D_j itself. The algorithm results in a ranked list of descriptions (or the corresponding subgroups) that satisfy the user-defined constraints.

In traditional SD, subgroup exceptionality is measured in terms of the distribution of only a single target variable. Hence, the typical quality measure contains a component indicating how different the distribution over the target variable in the subgroup is, compared to its distribution

[1]we consider the exact search strategy to be a parameter of the algorithm

in the whole dataset. Since unusual distributions are more easily achieved in small subsets of the dataset, the typical quality measure also contains a component indicating the size of the subgroup. Thus, whether a description is deemed interesting depends on both its exceptionality and the size of the corresponding subgroup.

EMM can be seen as a generalization of SD. Rather than one single target variable, EMM uses a more complex target concept. An instance of Exceptional Model Mining is defined by the combination of a *model class* over the targets, and a *quality measure* over this model class. When an instance has been defined, subgroups are generated (we will discuss how in the next section) to be evaluated. Then, for each subgroup under consideration, we induce a model on the targets. This model is learned from only the data belonging to the subgroup. Using the quality measure, the subgroup is evaluated based on model characteristics, to determine which subgroups are the most interesting ones. The typical quality measure in EMM indicates how exceptional the model fitted on the targets in the subgroup is, compared to either the model fitted on the targets in its complement, or the model fitted on the targets in the whole dataset — we will discuss this fundamental choice in Section 3.2.2. Just like in traditional SD, exceptional models are sometimes easily achieved in small subgroups, so if necessary, an EMM quality measure also contains a component indicating the size of the subgroup.

As we will explore in Section 3.2, there are several canonical choices that can be made when designing a quality measure for a selected model class. However, the framework allows a quality measure to be any function assigning a quality quantification to a description. This allows EMM to search for just about any imaginable instantiation of "interesting" subgroups.

So far, we have talked about Exceptional Model Mining in an informal, colloquial manner. This is deliberate. The goal is to find interesting subgroups of a dataset, for whatever instantiation of "interesting" the user of EMM cares for, which is intrinsically subjective. Therefore, any formal definition of the EMM task will only concern a subset of what we attempt to achieve with EMM. Nevertheless, to provide a more precise handle on what we will be concerned with in the following chapters, we can consider the following task definition

Problem Statement (Top-q Exceptional Model Mining). *Given a dataset Ω, description language $\mathcal{D}$, quality measure φ, positive integer q, and set of constraints $\mathcal{C}$. The Top-q Exceptional Model Mining task delivers the list $\{D_1, \ldots, D_q\}$ of descriptions in the language $\mathcal{D}$ such that*

* $\forall_{1 \leq i \leq q} : D_i$ *satisfies all constraints in $\mathcal{C}$;*
* $\forall_{i,j} : i < j \Rightarrow \varphi(D_i) \geq \varphi(D_j)$*;*
* $\forall_{D \in \mathcal{D} \setminus \{D_1, \ldots, D_q\}} : D$ *satisfies all constraints in $\mathcal{C} \Rightarrow \varphi(D) \leq \varphi(D_q)$.*

Informally, we find the q best-scoring descriptions in the description language that satisfy all constraints in $\mathcal{C}$. This set encompasses both user-induced constraints and search strategy limitations. These limitations include information about the exact choice we make for the refinement operator η, guiding how new candidate subgroups are generated out of other subgroups, and the limits to which we will explore the search space. In the following section we discuss the choices made for the search space traversal and the refinement operator in the remainder of this dissertation. Note that the general EMM *framework* leaves the choice for these matters open.

Also noteworthy is the fact that this problem statement includes the traditional Subgroup Discovery problem. This is a feature rather than a bug: we consider SD to be encompassed by EMM. In our view, Subgroup Discovery is simply a version of Exceptional Model Mining in which m, the number of targets as introduced in Section 2.1, is set to 1.

3.1 Search Strategy

Since the goal of SD/EMM is to find interesting subsets of the data, the corresponding search space is potentially exponentially large. Hence, we cannot simply explore this space by brute force; we need to find a more sophisticated search strategy. Part of the problem is already solved by only allowing subgroups. Since subgroups are subsets of the data for which a description exists, the set of subgroups is smaller than the set of subsets (although exactly *how much* smaller the set is, depends on the choice of description language $\mathcal{D}$). When many attributes in the dataset are numeric, the difference is not very substantial.

There are two main schools of thought in the community on how to overcome this problem, each with their own focus. The one, following canonical SD papers [55, 114], restricts the attributes in the dataset to be nominal and imposes an anti-monotonicity constraint on the used quality measure. Then the resulting search space can occasionally be explored exhaustively. The other resorts to heuristic search. This allows the attributes to be numeric as well, and facilitates a general quality measure. Since EMM is developed to capture any concept of interestingness in subgroups, we find allowing for any quality measure and numeric attributes more important than exhaustiveness. Hence we select the heuristic path. Exhaustive SD methods will be discussed in further detail in Section 3.3.1.

In the EMM setting, usually the *beam search* strategy is chosen, which performs a level-wise search. On each level, the best w (for *search width*) descriptions according to our quality measure φ are selected, and refined to create the candidate descriptions for the next level. The search is constrained by an upper bound on the complexity of the description (also known as the *search depth*, d) and a lower bound on the support of the corresponding subgroup. This search strategy combines the advantages of a greedy method with those of the implicit parallel search: as on each level w alternatives are considered, the search process is less likely to end up in a local optimum than a pure greedy approach, while selecting the w best descriptions at each level keeps the process focused, hence tractable.

3.1.1 Refinement Operator and Description Language

An important part of the beam search strategy is generating the set of candidate descriptions for the next level, by refining another description. This process is guided by the refinement operator η and the description language $\mathcal{D}$, for which we detail our choices in this section. Our description language $\mathcal{D}$ consists of logical conjunctions of conditions on single attributes.

We treat the numeric attributes with a particular kind of discretization, starting by fixing a positive integer $b \leq N$ (the number of *bins*) before the EMM process starts. On the first search level, when the generating description has no conditions, the discretization we apply is equal to static pre-algorithm discretization of the attribute into b bins of equal size. However, on each subsequent search level, our generating descriptions consist of a positive number of conditions, hence they cover strictly less than N records. Since on these levels we consider a discretization into b equal-sized bins of the attribute-values *within the generating non-empty description*, the bins may be different for each generating description. This *dynamic discretization* during the process draws more information from the attribute than we would use when statically discretizing it beforehand.

When η is presented with a description D to refine, it will build up the set $\eta(D)$ by looping over all the descriptive attributes $a_1, \dots, a_k$. For each attribute, a number of descriptions will be added to the set $\eta(D)$, depending on the attribute type

if a_i is binary: add $D \cap (a_i = 0)$ and $D \cap (a_i = 1)$ to $\eta(D)$;

if a_i is nominal, with values $v_1, \dots, v_g$: add $\{\ D \cap (a_i = v_j),\ D \cap (a_i \neq v_j)\ \}_{j=1}^{g}$ to $\eta(D)$;

if a_i is numeric: order the values of a_i that are covered by the description D; this gives us a list of ordered values $a_{(1)}, \dots, a_{(n)}$ (where $n = |G_D|$). From this list we select the split points $s_1, \dots, s_{b-1}$ by letting

$$\forall_{j=1}^{b-1} : s_j = a_{\left(\left\lfloor j\frac{n}{b}\right\rfloor\right)}$$

Then, add $\{\ D \cap (a_i \leq s_j),\ D \cap (a_i \geq s_j)\ \}_{j=1}^{b-1}$ to $\eta(D)$.

Informally, when presented with a description D, η will build a set of refinements by considering the descriptive attributes one by one. Each such refinement will consist of the conditions already present in D, plus one new condition. If an encountered attribute a_i is binary, 2 refined descriptions will be added to $\eta(D)$: one for which D holds and a_i is true, and one for which D holds and a_i is false. If the attribute a_i is nominal with g different values, $2g$ refined descriptions will be added to $\eta(D)$: for each of the g values, one where D holds and the value is present, and one where D holds and any of the $g - 1$ other values is present. If the attribute a_i is numeric, we divide the values for a_i *that are covered by* D into a predefined number b of equal-sized bins. Then, using the $b-1$ *split points* $s_1, \dots, s_{b-1}$ that separate the bins, $2(b-1)$ refined descriptions will be added to $\eta(D)$: for each split point s_j, one where D holds and a_i is less than or equal to s_j, and one where D holds and a_i is greater than or equal to s_j.

3.1.2 Beam Search Algorithm for Top-q EMM

Having described our choices for the search strategy and refinement operator that we will use in the remainder of this thesis, we can now describe and analyze an algorithm for the top-q Exceptional Model Mining problem stated earlier in this chapter. The pseudocode is given in Algorithm 1. In the algorithm, we assume that there is a subroutine called SATISFIESALL that tests whether a candidate description satisfies all conditions in a given set. Among the abstract datastructures we assume, the Queue is a standard queue with unbounded length. The PriorityQueue(x) is a queue containing at most x elements, where elements are stored and sorted with an associated quality; only the x elements with the highest qualities are retained, while other elements are discarded. In a straightforward but not too naive implementation, a PriorityQueue is built with a heap as its backbone. In this case the elementary operations, *insert_with_priority* for adding an element to the PriorityQueue and *get_front_element* for removing the element with the highest quality from the PriorityQueue, have a computational cost of $\mathcal{O}(\log x)$ [60, pp. 148–151].

Many statements in the algorithm control the beam search process in a straightforward manner. However, the process is also controlled by the interplay between the different (Priority-)Queues, which is more intricate and deserves attention. The resultSet is a PriorityQueue maintaining the q best found descriptions so far. Nothing is ever explicitly removed from the resultSet, but if the quality of a description is no longer among the q best, it is automatically discarded. Hence, the resultSet maintains the final result that we seek. The beam is a similar PriorityQueue, but with a different role. Here, the w best found descriptions so far *on the current search level* are maintained. When all candidates for a search level have been explored, the contents of the beam are moved into the unbounded but (by then) empty Queue candidateQueue, to generate the candidates for the next level.

Complexity

Since EMM is a highly parametrized algorithm, instantiated by a model class and quality measure, we need to introduce some notation before we

Algorithm 1 Beam Search for Top-q Exceptional Model Mining

```
Input: Dataset $\Omega$, QualityMeasure $\varphi$, RefinementOperator $\eta$,
    Integer $w$, $d$, $q$, Constraints $\mathcal{C}$
Output: PriorityQueue resultSet
 1: candidateQueue ← new Queue;
 2: candidateQueue.enqueue({});            ▷ Start with empty description
 3: resultSet ← new PriorityQueue(q);
 4: for (Integer level ← 1; level ≤ d; level++) do
 5:     beam ← new PriorityQueue($w$);
 6:     while (candidateQueue ≠ ∅) do
 7:         seed ← candidateQueue.dequeue();
 8:         set ← $\eta$(seed);
 9:         for all (desc ∈ set) do
10:             quality ← $\varphi$(desc);
11:             if (desc.SATISFIESALL($\mathcal{C}$)) then
12:                 resultSet.insert_with_priority(desc,quality);
13:                 beam.insert_with_priority(desc,quality);
14:     while (beam ≠ ∅) do
15:         candidateQueue.enqueue(beam.get_front_element());
16: return resultSet;
```

can analyze the computational complexity of the algorithm. We write $M(n, m)$ for the cost of learning a model from n records on m targets, and c for the cost of comparing two models from the chosen model class.

Theorem 1. *The worst-case computational complexity of Algorithm 1 is*

$$\mathcal{O}\left(dwkN\left(c + M(N, m) + \log(wq)\right)\right)$$

Proof. We start our analysis at the innermost loop, working bottom-up. Line 12 inserts an element into a PriorityQueue of size q, which costs $\mathcal{O}(\log q)$. Line 13 does the same for a PriorityQueue of size w, and hence costs $\mathcal{O}(\log w)$. The conditions checked in line 11 are the user-induced constraints a domain expert may impose on the resulting descriptions. These usually are relatively simple conditions concerning for instance a minimal number of records covered by the descriptions. As such, they are relatively cheap to check. For all reasonable constraints a domain expert may come up with, the necessary information can be extracted during the same scans of the dataset we need when, for instance, computing the quality of the description in the preceding line. As such, we assume the computational complexity of line 11 to be dominated by the complexity of line 10. The worst-case scenario is that all descriptions pass the test, hence the commands inside the if-loop need to be computed every time. Thus, the total complexity of lines 11 through 13 is $\mathcal{O}(\log w + \log q) = \mathcal{O}(\log(wq))$.

Line 10 computes the quality of a description. In the worst case, this requires the learning of two models: one on the description and one on its complement, and comparing these models. Hence: $\mathcal{O}(c + 2M(N, m))$ $= \mathcal{O}(c + M(N, m))$ (recall the definition of c and $M(N, m)$, as introduced just before Theorem 1). In line 9, a loop is run for all refinements of a seed description. By our choice of η, the worst case would be if every descriptive attribute were nominal (or numeric) having N distinct values. For each of the k descriptors (cf. Section 2.1), we would then generate $2N$ refinements. The loop is thus repeated $2kN$ times, which costs $\mathcal{O}(kN)$. Hence, the total complexity of lines 9 through 13 is $\mathcal{O}(kN(c + M(N, m) + \log(wq)))$.

Line 8 enumerates all refinements of one description, which we have just analyzed to cost $\mathcal{O}(kN)$. Line 7 dequeues an element from an ordinary Queue, which can be done in $\mathcal{O}(1)$. Line 6 loops all previously analyzed lines as many times as there are elements in the candidateQueue. This queue never has more than w elements, since it is always emptied before (in line 15) at most w new elements are added to the queue. Hence, the total complexity of lines 6 through 13 is $\mathcal{O}(w(kN + kN(c + M(N, m) + \log(wq)))) =$ $\mathcal{O}(wkN(c + M(N, m) + \log(wq)))$.

On the same level we find line 5, which costs $\mathcal{O}(1)$, and the while-loop of lines 14 through 15, which costs $\mathcal{O}(w \log w)$ if done extremely naively. These lines are dominated in complexity by lines 6 through 13. All these lines are enveloped by a for-loop starting at line 4, which is repeated d times. Lines 1 through 3 and 16 can be computed in constant time, and so the total computational complexity of Algorithm 1 becomes

$$\mathcal{O}(dwkN(c + M(N, m) + \log(wq))) \qquad \square$$

This complexity seems relatively benign; we see no factors with exponents higher than one, and the worst parameter has complexity $\mathcal{O}(w \log w)$, which is tractable for a generous range of values for w. However, there are some variables in the complexity expression, which can lead to higher powers of parameters if we fill them in by selecting a model class and quality measure. For instance, if we would perform traditional Subgroup Discovery with this algorithm, we would be searching for descriptions having an unusually high mean for one designated target. Hence, the model computation complexity becomes $M(N, 1) = \mathcal{O}(N)$, and the model comparison cost becomes $c = \mathcal{O}(1)$. Thus, the total computational complexity of Beam Search for Top-q Subgroup Discovery would be $\mathcal{O}(dwkN(N + \log(wq)))$, which is quadratic in the number of records in the dataset.

Note that this computational complexity is in many respects a worst-case scenario, whose bounds a real-life run of the algorithm is unlikely to meet. Since data of such high cardinality is rarely obtained, the number of refinements of a seed description is usually much lower than $2kN$. Also, unlike in the worst-case scenario, the beam search converges in such a way that per search level the subgroups reduce in size, hence the modeling is done over progressively smaller parts of the dataset. Also noteworthy are the facts that when a dataset is extended with more data of the same cardinality, the algorithm scales linearly, and that the number of candidates under consideration is roughly equal per search level, except for level $d = 1$.

3.1.3 Alternatives to Beam Search

Whereas the traditional EMM framework strives to find exceptional descriptions by searching through the descriptive attribute space, and evaluating on the target attribute space, interesting results have been obtained by taking a more symmetrical approach to the two subspaces of the data. The EMDM algorithm [69] strives to effectively find exceptional models by iteratively improving candidate descriptions, exploiting structure in both spaces. Each iteration consists of two steps, one for Exception Maximization (EM) and one for Description Minimization (DM). In the EM step, a compression-based quality measure guides the search for subsets having an unusual model. In the DM step, a rule-based classifier is employed to find a concise description that crafts a subgroup from the found subset. Upon convergence, or when a threshold on the number of iterations is surpassed, the subgroups are reported.

The well-known FP-Growth algorithm has been adapted, to enable exhaustive EMM. Lemmerich et al.'s *generic pattern growth* algorithm (GP-Growth) [72] strives to avoid scanning the whole dataset to evaluate descriptions. Instead, it builds a special data structure, in which the key information of the model learned for a description is summarized. Such a summary is called a *valuation basis*. It contains enough information to determine the quality of any refinement of the description. The GP-Growth algorithm can reduce the memory requirement and runtime of an EMM instance by more than an order of magnitude, but only when a valuation basis can be found that is suitably condensed. This depends on the computational expense of the model class: if a parallel single-pass algorithm with sublinear memory requirements exists to compute the model from a given set of records, profit can be gained from GP-Growth. Most of the model classes we will discuss can benefit from GP-Growth, but in Chapter 6 we will see a model which cannot.

3.2 How to Define an EMM Instance?

As previously described, an EMM instance is defined by the choice of model class over the targets, and quality measure over the model class. In the following four chapters we define several such instances. Before that, we discuss some general themes that recur in EMM instance definitions.

The choice of model class is usually inspired by a real-life problem. For instance, when the goal is to find deviating dependencies between several species in an ecosystem, one is drawn towards graphical models such as Bayesian networks and Markov models. If we can formulate the relation between the targets for which we are interested in finding exceptions, this usually naturally directs our attention to a particular model class.

3.2.1 Quality Measure Concepts

Having chosen a model class, we need to define a quality measure that extracts characteristics from the learned models, and extracts from these characteristics a quantification of how different the models are from each other. Usually such a quantification is relatively straightforward to design. For instance, if the model class is a regression model with two variables, one could take the difference between the estimated slopes in each model as quality measure. However, such a quantification is typically not enough to design a proper measure for the quality of a description. After all, deviations from the norm are easily achieved in very small subsets of the data. Hence, directly taking a difference quantification as quality measure probably leads to descriptions of very small subgroups, which are usually not the most interesting ones to domain experts. Therefore, we somehow want to represent the size of a subgroup in a quality measure.

In some of the canonical quality measures for Subgroup Discovery, such as Weighted Relative Accuracy (WRAcc) [35], the size of a subgroup is directly represented by a factor n or $\sqrt{n}$. Though their simplicity is appealing, we find it somewhat counter-intuitive to have a factor in a quality measure that explicitly favors subgroups covering the entire dataset over smaller subgroups. A slightly more sophisticated way to represent the subgroup size, is to multiply (i.e. weigh) the quantification of model difference with the *entropy* of the split between the subgroup and its complement. The entropy captures the information content of such a split, and favours balanced splits (1 bit of information for a 50/50 split) over skewed splits (0 bits for the extreme case of either subgroup or complement being empty). The entropy function $\varphi_{\text{ef}}(D)$ is defined (in this context) as

$$\varphi_{\text{ef}}(D) = -{}^{n}/_{N} \lg {}^{n}/_{N} - {}^{n^C}/_{N} \lg {}^{n^C}/_{N}$$

Another way to direct the search away from extremely small subgroups, is by employing a quality measure based on a statistical test. For certain models there may be hypotheses of the form

H_0 : model parameter for description = model parameter for complement
H_1 : model parameter for description $\neq$ model parameter for complement

which we can test, usually involving some statistical theory, to derive an expression for which we can compute a p-value. Then, using $1 - p$ as the quality measure, we have constructed a measure ranging from 0 to 1 for which higher values indicate more interesting descriptions.

Sections 4.1 (φ_{scd}), 5.1 (φ_{sed}) and 7.4 (φ_{ssd}) feature examples of quality measures that are directly based on a statistical test. In Sections 4.3 (φ_{ent})

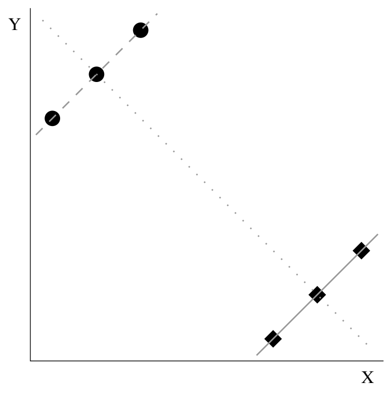

Figure 3.1: Should we compare a subgroup G_D to its complement G_D^C, or to the whole dataset Ω?

and 6.1 (φ_{weed}) we find examples of quality measures employing the entropy function. Quality measures from Sections 4.3 (φ_{abs}), 5.3 (φ_{BDeu} and φ_{Hel}), 6.3 (φ_{ed}), and 7.1 (φ_{Cook}) consist solely of a difference quantification (occasionally these are statistically inspired, but they are not directly based on an established statistical test).

3.2.2 Compared to what?

So far we have discussed quality measure development as a means of assessing how different two learned models are from one another, and how to ensure that subgroups have a substantial size. However, we have neglected a cardinal point. Since a quality measure should assign a quality to a description, its model should be compared, but to which other model? There are two options: we can compare the model for a description of a subgroup G_D either to the model for its complement G_D^C, or to the model for the whole dataset Ω. The simple constructed example from Figure 3.1 illustrates that these two comparisons can lead to very different outcomes.

Suppose that we have a two-dimensional target space, and we are concerned with finding descriptions having a deviating regression line in these two dimensions. Figure 3.1 depicts the target space, and the six records in the example dataset. The dotted grey line is the regression line of the whole dataset, with slope -1. Now suppose that we find the description D covering the records depicted as circles. The dashed grey line is the regression line of G_D, with slope 1. The solid grey line is the regression line of G_D^C, also having slope 1. When gauging the exceptionality of a description solely by the slope of the regression line, we find G_D interesting when compared to Ω, but not at all when compared to G_D^C. Of course, the assessment changes when we include the intercept in the evaluation.

The problem as displayed in Figure 3.1 is underdetermined; we have not enough information to formulate an opinion on whether the subgroup should be deemed interesting. It can therefore not be used to illustrate whether comparing to G_D^C or to Ω is preferable; it merely illustrates that a different choice may lead to a different outcome.

There is not always a clear-cut preferred choice whether to compare to G_D^C or to Ω. Sometimes, the real-life problem at hand can point in one direction: if we are interested in deviations from a possibly inhomogeneous norm, it makes more sense to compare to Ω, whereas if we are interested in dichotomies, it makes more sense to compare to G_D^C. On other occasions, a statistically inspired quality measure may *require* choosing either Ω or G_D^C, to prevent violation of mathematical assumptions. Lastly, when the model class is so complicated that learning models from data covered by descriptions has a nontrivial computational expense, efficiency might dictate the choice: when comparing n descriptions to Ω, learning $n+1$ models suffices, but when comparing them to G_D^C, learning $2n$ models is required.

The previous two practical considerations supersede any personal preference that we outline; if the model class choice and quality measure design somehow require comparing to either G_D^C or Ω, then that is the way to go. However, when given the choice, we would consider comparing to Ω preferable. After all, Exceptional Model Mining is designed as a Local Pattern Mining task, where we strive to find coherent subsets of the data where something interesting is going on. The goal is to pinpoint many such deviations of the norm, possibly overlapping, without consideration for the coherence and model parameters occurring in the remainder of the dataset. When we compare the model for a subgroup G_D to the model for Ω, we evaluate a subgroup by comparing its behavior to the behavior for the entire dataset. This implies that we strive to find subgroups deviating from the norm. By contrast, when we compare the model for a subgroup G_D to the model for G_D^C, we evaluate a subgroup by comparing its behavior to the behavior on the complement of the dataset. This implies that we strive to find schisms in the dataset: not necessarily one subgroup deviating from the norm, but rather a partitioning of Ω into two subgroups displaying clearly contrasting behavior. We think this is a very interesting task, but it may not strictly adhere to the goals of Exceptional Model Mining.

3.3 Related Work

Exceptional Model Mining extends a vast body of work, of which this section contains some highlights. First we discuss the search strategies developed to deal with the exponential search space. Then we look into other local pattern mining tasks, and other extensions of Subgroup Discovery. Finally, we discuss how similar questions arise in other data mining disciplines, and what distinguishes them from EMM.

3.3.1 Search Strategies for SD/EMM

When striving to find interesting subsets of a dataset, the search space is exponential in the number of records. By restricting the problem to finding interesting *subgroups*, i.e. subsets with a concise description, the search space remains theoretically exponential in size, but we obtain a handle with which we can tackle the problem. Traditionally [55], this is done by compelling all attributes in the dataset to be nominal. In this case, occasionally exhaustive search is possible, using filters akin to the anti-monotonicity constraints known from frequent itemset mining. When not all attributes are nominal, traditionally there was no other option than to resort to heuristic search.

Recently, Grosskreutz and Rüping developed a new pruning scheme with accompanying SD algorithm, MergeSD [45], which allows for exhaustive mining even when the attributes are taken from a numeric domain. Their key idea is to exploit bounds between related numeric descriptions to prune with optimistic estimates, thus reducing the search space to tractable levels. Unfortunately, the pruning scheme cannot be used with any quality measure; implicitly a constraint similar to anti-monotonicity is imposed.

In work dedicated to expanding the description language $\mathcal{D}$ available to Subgroup Discoverers, Mampaey et al. introduced an efficient treatment of numeric attributes [76]. The description space is not explored exhaustively. Instead, the algorithm finds richer descriptions efficiently, by finding an optimal interval for every numeric attribute, and an optimal value set for every nominal attribute. The efficiency comes from considering only descriptions that lie on a convex hull in ROC space, and evaluating them with a convex quality measure. Hence, the method is only suitable for a target concept that can be properly expressed in ROC space, i.e. traditional SD with a nominal target, and a convex concept of interestingness.

Another problem stemming from the exponential search space is the redundancy in a resulting description set. When a description is deemed interesting, small variations will very likely deliver other descriptions that are also quite interesting. Therefore it is not uncommon, especially when there are numeric attributes in the dataset, to find the top of a description chart dominated by many copies of what technically may all be slightly different descriptions, which in practice all indicate the same underlying concept. Van Leeuwen et al. [70] introduced three degrees of subgroup redundancy, and incorporated selection strategies based on these redundancies in a beam search algorithm. This results in non-exhaustive, but interestingly different search strategies.

The only work so far on exhaustive Exceptional Model Mining, is Lemmerich et al.'s GP-Growth algorithm [72], which was discussed in detail earlier in this chapter. It can severely reduce the memory requirement and runtime of an EMM instance, but only when a parallel single-pass algorithm with sublinear memory requirements exists to compute the model from a given set of records. This can be done for relatively simple model classes, but not for more computationally expensive model classes (cf. Section 3.1.3).

3.3.2 Similar Local Pattern Mining Tasks

Subgroup Discovery research orginated in the mid-nineties, in a simple single-table setting with a binary target attribute [55], and in a multi-relational setting [114]. Tasks that are very similar to, but slightly different from Subgroup Discovery, include Contrast Set Mining [4], where the goal is to find "conjunctions of attributes and values that differ meaningfully in their distributions across groups", and Emerging Pattern Mining [21], which strives to find itemsets whose support increases substantially from one dataset to another. One could view the latter task as an amalgamation of two separate Subgroup Discovery runs (one for each dataset), followed by a search for classification rules (where a found subgroup has class 1 when found on dataset 1, and class 2 when found on dataset 2). Kralj Novak et al. provide a framework unifying Contrast Set Mining, Emerging Pattern Mining, and Subgroup Discovery [65].

Giving a full overview of all work related to Subgroup Discovery is beyond the scope of this dissertation; such overviews are available in the literature (for instance: [50]). In the remainder of this section we focus on work related to supervised local pattern mining with a more complex goal.

As the antithesis to Contrast Set Mining, Redescription Mining [39, 91] seeks multiple descriptions of the same subgroups, originally in itemset data. Recent extensions incorporate nominal and numeric data [38].

Umek et al. [109] consider Subgroup Discovery with a multi-dimensional output space. They approach this data by considering the output space first: agglomerative clustering in the output space proposes candidate subgroups that have records similar in outcomes. Then, a predictive modeling technique is used to test for each identified candidates whether they can be characterized by a description over the input space.

One of the few papers that explicitly seeks a deviating model over a target attribute, concerns Distribution Rules [54]. In this work, there is only one numeric target, and the goal is to find subgroups for which the distribution over this target is significantly different from the overall distribution, measured in terms of the Kolmogorov-Smirnov test for goodness of fit. Since rules are evaluated by assessing characteristics of a model, this can be seen as an early instance of Exceptional Model Mining, albeit considering only one target attribute.

3.3.3 Similar Tasks with a Broader Scope

General concepts from EMM, like fitting different models to different parts of the data, or identifying anomalies in a dataset, appear in tasks beyond Local Pattern Mining. In this section we discuss a few such tasks, and how they relate to EMM.

In Outlier Detection, traditionally the goal is to identify records that deviate from a general mechanism. Usually there is no desire to find a coherent set of such outliers, which can succinctly be described: identifying non-conforming records is enough. As Outlier Detection becomes more and more mature and sophisticated, we witness more attention towards the underlying mechanism making a point an outlier, for instance in recent work by Kriegel et al. [66]. Their method to detect outliers in arbitrarily oriented subspaces of the original attribute space also delivers an explanation with each outlier, consisting of two parts: an error vector, pointing towards the expected position of the outlier, and an outlier score, quantifying the likeliness that this point is an outlier. Searching for the reason for outliers is a step towards bridging the gap with finding coherent deviating subsets as done in EMM, although the approaches differ vastly. Alternatively, Konijn et al. [63] have designed a hybrid method, post-processing regular Outlier Detection results with a Subgroup Discovery run. This enables higher-level analysis of Outlier Detection results.

When fitting a regression function to a dataset with a complex underlying distribution, one could employ Regression Clustering [116]. The idea is to simultaneously apply $K > 1$ regression functions to the dataset, clustering the dataset into K subsets that each have a simpler distribution than the overall distribution. Each function is then regressed to its own subset, resulting in smaller residual errors, and the regressions and clustering optimize a common objective function. Catering for parts of the dataset where a fitted model is substantially different is a shared idea between Regression Clustering and EMM. However, in Regression Clustering the subsets are not necessarily coherent, easy to describe subgroups: the goal is not to explore exceptionalities, but to give a well-fitting partition.

A similar caveat holds for the well-known Classification And Regression Trees [7], where a nominal or numeric target concept is assigned a different class or outcome depending on conditions on attributes. While the recursive partitioning given by the tree ensures that every path from the root to a leaf constitutes a coherent, easy to describe subgroup, there is again no explicit search for exceptionalities. A partition that performs well is enough, and if multiple exceptional phenomena that happen to have similar effects on the target are found in the same cell of the partition, the CART algorithm judges this as a good outcome while from the Exceptional Model Mining viewpoint it is not.

As an extension of the regression tree algorithm provided by CART, where the leaves contain numeric values as opposed to the classes found in the leaves of a decision tree, the M5 system [90] produces trees having multivariate linear regression models in the leaves. Instead of learning a global model for the entire dataset, M5 partitions the dataset by means of the internal nodes of the tree, and learns a local model for each leaf. Essentially, the resulting tree can be seen as a piecewise linear regression model. M5 can also be seen as a sibling of Regression Clustering, but with an easy-to-describe partition and a hierarchical clustering. As is the case with CART and Regression Clustering, contrary to EMM the goal of M5 is not to find exceptionalities but to completely partition the data, and the focus is on the overall performance in the target space rather than separation of exceptional phenomena.

Contrary to ordinary decision trees, where the classes are found in the leaves of the tree and the internal nodes merely contain conditions for classification, a Predictive Clustering Tree (PCT) [5] has each internal node and each leaf corresponding to a cluster. A cluster is represented by a prototype, and a distance measure is assumed that computes the distance between prototypes hence clusters. Given all this, the decision tree algorithm is adapted to select in each node the condition maximizing the distance between the clusters in its children. Defining a quality measure that finds an optimal separation between a subset of the data and its complement, is a common concept in PCT and EMM. However, the goal of PCT is not to find global exceptionalities, but rather find a partition of the data that is optimal in some sense.

The work on PCTs has been generalized to concern the general problem of mining on a dataset with structure on the output classes, whether this structure takes the form of dependencies between classes (tree-shaped hierarchy, directed acyclic graph) or internal relations between classes (sequences). A tree ensemble method for such data was proposed by Kocev et al. [61]. Their method is able to give different predictions for parts of the dataset that behave differently from the norm. Contrary to EMM, there is no explicit identification of the deviating subgroup and model.

3.4 Software

In the following chapters, we will introduce model classes and quality measures, and run experiments with the corresponding Exceptional Model Mining instances. These experiments are primarily performed with the *Cortana* discovery package [78]: a Java implementation that is an open-source spin-off of the Safarii Data Mining system.

Cortana is not limited to Exceptional Model Mining; it provides multiple supervised Local Pattern Mining tasks. The user can set the task he/she wants Cortana to perform by selecting a *target concept*. For the simplest target concept, SINGLE_NOMINAL, the user must highlight one nominal attribute, and Cortana will perform Subgroup Discovery with that attribute as target. Similarly, for the SINGLE_NUMERIC target concept, one numeric attribute needs singling out, for Cortana to use as numeric target in a Subgroup Discovery run. Several Exceptional Model Mining instances are covered by other target concepts: DOUBLE_CORRELATION handles the Correlation model from Chapter 4, the MULTI_LABEL target concept corresponds to the Bayesian network model from Chapter 6, and the DOUBLE_REGRESSION target concept concerns the simple Regression model from Section 7.4. For each target concept, a range of quality measures is available that allow the user to define exactly what sort of exceptional subgroups Cortana should search for. The subgroup validation method we develop in Chapter 8 is also available in Cortana.

Independent of the choice of target concept and quality measure, Cortana provides a parametrized search algorithm. Several search strategies are included (breadth-first, depth-first, best-first, beam, and cover-based beam

[70]), and the user can select one of many strategies for dealing with numeric attributes. Furthermore, conditions can be set on the minimal subgroup size, the minimal subgroup quality, the maximal description length, the maximal number of subgroups to present at the end of the algorithm, and the maximal amount of total time the algorithm spends on the task.

Apart from many more things, Cortana provides a parametrized version of the Beam Search algorithm for Top-q Exceptional Model Mining, as detailed in Algorithm 1. It is available online, at `http://datamining.liacs.nl/cortana.html`.