Explanation Tables

Kareem El Gebaly
University of Waterloo
kelgebaly@uwaterloo.ca

Guoyao Feng
CMU
gfeng@andrew.cmu.edu

Lukasz Golab
University of Waterloo
lgolab@uwaterloo.ca

Flip Korn
Google Research
flip@google.com

Divesh Srivastava
AT&T Labs - Research
divesh@research.att.com

Abstract

We present a robust solution to the following problem: given a table with multiple categorical dimension attributes and one binary outcome attribute, construct a summary that offers an interpretable explanation of the factors affecting the outcome attribute in terms of the dimension attribute value combinations. We refer to such a summary as an explanation table*, which is a disjunction of overlapping patterns over the dimension attributes, where each pattern specifies a conjunction of attribute=value conditions. The* Flashlight *algorithm that we describe is based on sampling and includes optimizations related to computing the information content of a summary from a sample of the data. Using real data sets, we demonstrate the advantages of explanation tables compared to related approaches that can be adapted to solve our problem, and we show significant performance benefits of our approach.*

1 Introduction

Before making complex data-driven decisions, users often wish to obtain a summary of the available data sets to understand their properties; this is particularly important for auditing prior knowledge and/or policy, for example to ensure fairness. Motivated by this need, we present a robust solution to the following problem: given a table T with multiple categorical *dimension attributes* and one binary *outcome attribute* b, how can we construct a summary that offers an interpretable explanation of the factors affecting the binary outcome attribute in terms of the dimension attribute value combinations?

Consider the following example, drawn from [6]. An athlete may be interested in deciding when to exercise and what to eat before exercising to meet her target goals. To make these decisions in a data-driven fashion, she monitors her activities and maintains an exercise log. Table 1 shows her exercise log, with each row containing an ID, the time and day of the week of the exercise, the meal eaten before the exercise, and a binary attribute indicating the outcome of the exercise, i.e., whether a target goal was met. The three dimension attributes in this example are `day`, `time` and `meal`.

Table 2 illustrates a corresponding summary. It contains a list of *patterns* that specify subsets of tuples, where a pattern is a tuple from the data cube over the dimension attributes of T. Patterns consist of attribute values or the wildcard symbol "*" denoting all possible values. Each pattern is associated with the number of

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Table 1: An Exercise relation

id	day	time	meal	goal met?
1	Fri	Dawn	Banana	1
2	Fri	Night	Green salad	1
3	Sun	Dusk	Oatmeal	1
4	Sun	Morning	Banana	1
5	Mon	Afternoon	Oatmeal	1
6	Mon	Midday	Banana	1
7	Tue	Morning	Green salad	0
8	Wed	Night	Burgers	0
9	Thu	Dawn	Oatmeal	1
10	Sat	Afternoon	Nuts	0
11	Sat	Dawn	Banana	0
12	Sat	Dawn	Oatmeal	0
13	Sat	Dusk	Rice	0
14	Sat	Midday	Toast	0

Table 2: An explanation table over the Exercise relation

day	time	meal	goal met?	count
*	*	*	.5	14
Sat	*	*	0	5
*	*	Banana	.75	4
*	*	Oatmeal	.75	4

tuples it matches (count) and a fraction indicating how many of these tuples have the binary outcome attribute equal to one.

The first pattern in Table 2 matches all the tuples in Table 1 and indicates that, on average, half the workouts were successful. The second pattern reveals that all Saturday workouts were unsuccessful, which provides the most information about the outcome attribute beyond what we know so far from the first pattern. Intuitively, this is because the average value of the goal met attribute on Saturdays is significantly different than the average value of this attribute in the entire dataset, and there are sufficiently many Saturday tuples in the dataset. The last two patterns state that eating bananas or oatmeal led to successful workouts 75 percent of the time. Note that patterns may overlap. For example, the patterns (*,Morning,*) and (*,*,Banana) overlap, suggesting that working out in the morning after eating a banana could influence the distribution of the outcome attribute differently from working out in the morning or working out after eating a banana.

A natural language for expressing summaries over multi-dimensional data is in terms of a disjunction of patterns over the dimension attributes, where each pattern specifies a conjunction of "attribute=value" conditions. This form of expression has appeared extensively in the data mining and OLAP literature, from datacubes to association rules, and has been employed more specifically in settings where an "explanation" of some property or outcome attribute in a data set in terms of the dimensions is needed; see [3, 4, 8, 12, 14] for examples. However, since these existing explanatory summaries are not always grounded in information theory, we argue that they are not fully optimized for *interpretability* in the following sense. We define the *accuracy* of an explanation E in terms of its information loss, quantified by the relative entropy with respect to some table T that it summarizes, where less information loss implies more accuracy. Since there is an inherent trade-off between accuracy and

explanation size, interpretability is a relative notion. Therefore, given two explanations, E and E', where E is at least as accurate as E' and E has fewer patterns than E' (i.e., $|E| < |E'|$), by Occam's Razor we should prefer E since it is the more *concise* one.[1]

We take a summary with fewer (disjunctive) patterns as being more concise, rather than the total number of "attribute=value" conditions, because it provides explanations that are data-driven rather than schema-driven. For example, when many tuples in T that are concentrated in a very narrow subspace have uniform labels, it is desirable to include a pattern containing them in a summary, but counting the number "attribute=value" conditions would penalize heavily for this pattern. Since negations are not used in patterns, intuitively the way that we are able to achieve conciseness is by allowing the patterns to *overlap* on the data set records that they match. In contrast, many existing methods such as decision trees use disjoint patterns, which can result in significantly more complex explanations (e.g., due to negations). Among the existing approaches that allow overlap, the semantics of the overlap are not rich enough to provide conciseness, as we demonstrate experimentally in this paper.

2 Explanation Tables

Our proposed summary is called *Explanation Tables*, an example of which was provided in Table 2 for the data set in Table 1. Next, we explain our algorithm for computing explanation tables; see [6, 5, 7] for further details and extensions to numeric outcome attributes, distributed computation and in-browser computation using Javascript.

2.1 Computing Explanation Tables

The main idea is to maintain *maximum-entropy estimates* $\hat{b}$ for the values of the outcome attribute b and refine them as new patterns are added to the explanation table. Let $t[b]$ and $t[\hat{b}]$ be the actual and estimated values of b, respectively, for a tuple $t \in T$. We use *Kullback-Leibler Divergence* (KL-Divergence) to quantify the difference between the actual and estimated distributions, which is defined as follows for binary attributes: $KL(b||\hat{b}) = \sum_{t \in T | t[b]=1} \log\left(\frac{1}{t[\hat{b}]}\right) + \sum_{t \in T | t[b]=0} \log\left(\frac{1}{1-t[\hat{b}]}\right)$. In each iteration of the algorithm, we greedily add a pattern to the explanation table that provides the most information gain, i.e., leads to the greatest reduction in KL-Divergence. We stop after k iterations, where k is a user-supplied parameter, yielding an explanation table with k patterns.

Recall Table 2. Based on the first pattern (the all-stars pattern), the maximum-entropy estimate of the outcome attribute is to set each value (of each of the 14 rows in Table 1) to 0.5. Of course, the actual values of the `goal met` attribute are binary, but we allow the estimates to be real numbers between zero and one. This maximum-entropy estimate only uses the information implied by the first pattern of the explanation table, which is that `AVG(goal met)=0.5`, without making any other assumptions.

As it turns out, the pattern (Sat,*,*) is then added to the explanation table because it provides the greatest information gain. As a result, we update the estimates $t[\hat{b}]$ as follows. Since the second pattern implies that all Saturday tuples have `goal met=0`, we set the estimates for rows 10 through 14 in Table 1 to zero. Next, for consistency with the first pattern, which requires the average value of the outcome attribute over the whole dataset to be 0.5, we must set the estimates for all non-Saturday rows to $\frac{7}{9}$ each. This gives us a maximum-entropy estimate that only considers the information contained in the first two patterns of the explanation table.

Since the patterns of an explanation table may overlap, it is not obvious how to compute a maximum-entropy estimate in the general case. We use an approximation technique called *iterative scaling* [1, 2] to compute updated estimates whenever a new pattern is added to the explanation table.

[1] Usually the goal is not to (losslessly) capture every detail in the data set but represent it at some granularity that achieves a balance between filtering the noise while preserving the signal.

Algorithm 1 Greedy algorithm template for constructing explanation tables

Require: T, k

1: $E \leftarrow (*, \ldots, *)$
2: **for** each $t \in T$ **do**
3: $\quad t[\hat{b}] \leftarrow$ fraction of rows in T with $t[b] = 1$
4: **end for**
5: **for** $i = 1$ to $k - 1$ **do**
6: $\quad P \leftarrow$ set of candidate patterns
7: $\quad p_{opt} \leftarrow \arg\max_{p \in P}$ information gain(p)
8: $\quad E \leftarrow E \cup p_{opt}$
9: $\quad$ update $t[\hat{b}]$ for each $t \in T$ using iterative scaling
10: **end for**

Algorithm 1 shows a greedy algorithm template for constructing an explanation table E with k patterns from table T. We iterate k times, once to obtain each pattern, with the all-stars pattern added first. In each subsequent iteration, we identify a set P of candidate patterns, we then add to E the pattern from P with the highest information gain, and we finally update the maximum-entropy estimates $t[\hat{b}]$ via iterative scaling.

2.2 Sampling to Improve Efficiency

A naive instantiation of the template of Algorithm 1 considers all possible patterns in line 6 during each iteration. This may not be feasible for large datasets, both in the number of rows and the number of columns. To improve efficiency, we *prune* the candidate space by drawing a random sample s from T in line 6 (a different sample in each iteration of the loop) and setting P to be only the patterns from the data cube of the sample. The intuition is that patterns with frequently occurring combinations of dimension attribute values are likely to be sampled and also to have high information gain. In line 7, we compute the gain of only the patterns in P, as described below.

We illustrate our solution using Table 1. Suppose that the pattern (*,*,*) has been added to the explanation table, meaning that $t[\hat{b}] = 0.5$ for each tuple in T. Suppose that we draw a sample s that consists of tuples 4, 9 and 11 from Table 1.

The first step is to compute the *Lowest Common Ancestor* (LCA) of each pair of tuples having one tuple from s and one from T. To compute the LCA, non-matching (categorical) attribute values are replaced with stars. For example, the LCA of (Sun,Morning,Banana) and (Fri,Dawn,Banana) is (*,*,Banana). Table 3 shows all the LCAs in our example, including the same LCAs computed from different pairs of tuples. Each row of Table 3 corresponds to one of the 14 tuples from T and each column corresponds to one of the three tuples from the sample s.

In the second step, we do a group-by over the LCAs, and, for each distinct pattern, we compute the three aggregates shown in Table 4: count, sum($t[b]$) and sum($t[\hat{b}]$). Ignore "Ancestors" for now; we will explain them in the next step. To understand these three aggregates, consider (*,*,Banana). It was generated as an LCA five times, as shown in Table 3. Thus, its count is five. To understand sum($t[b]$), note that the five occurrences of (*,*,Banana) in the LCA set are derived from (Fri,Dawn,Banana) which has $t[b] = 1$, (Sun,Morning,Banana) which has $t[b] = 1$, (Mon,Midday,Banana) twice which has $t[b] = 1$, and finally (Sat,Dawn,Banana) which has $t[b] = 0$. The sum of these $t[b]$'s is four, including counting (Mon,Midday,Banana) twice. Similarly, the sum of the $t[\hat{b}]$'s is 2.5, including counting (Mon,Midday,Banana) twice; recall that at this point every tuple in T has $t[\hat{b}] = 0.5$.

In the third step, we generate the *ancestors* of each distinct LCA pattern p, defined as p itself plus all other patterns that are the same as p except one or more of the attribute values in p have been replaced by wildcards. These are shown in the rightmost column of Table 4. Each ancestor of p is associated with the same count,

Table 3: Computing LCAs (first step of the algorithm)

T \s	(Sun,Morning,Banana)	(Thu,Dawn,Oatmeal)	(Sat,Dawn,Banana)
(Fri,Dawn,Banana)	(*,*,Banana)	(*,Dawn,*)	(*,Dawn,Banana)
(Fri,Night,Green salad)	(*,*,*)	(*,*,*)	(*,*,*)
(Sun,Dusk,Oatmeal)	(Sun,*,*)	(*,*,Oatmeal)	(*,*,*)
(Sun,Morning,Banana)	(Sun,Morning,Banana)	(*,*,*)	(*,*,Banana)
(Mon,Afternoon,Oatmeal)	(*,*,*)	(*,*,Oatmeal)	(*,*,*)
(Mon,Midday,Banana)	(*,*,Banana)	(*,*,*)	(*,*,Banana)
(Tue,Morning,Green salad)	(*,Morning,*)	(*,*,*)	(*,*,*)
(Wed,Night,Burgers)	(*,*,*)	(*,*,*)	(*,*,*)
(Thu,Dawn,Oatmeal)	(*,*,*)	(Thu,Dawn,Oatmeal)	(*,Dawn,*)
(Sat,Afternoon,Nuts)	(*,*,*)	(*,*,*)	(Sat,*,*)
(Sat,Dawn,Banana)	(*,*,Banana)	(*,Dawn,*)	(Sat,Dawn,Banana)
(Sat,Dawn,Oatmeal)	(*,*,*)	(*,Dawn,Oatmeal)	(Sat,Dawn,*)
(Sat,Dusk,Rice)	(*,*,*)	(*,*,*)	(Sat,*,*)
(Sat,Midday,Toast)	(*,*,*)	(*,*,*)	(Sat,*,*)

sum($t[b]$) and sum($t[\hat{b}]$) values as p. We then take all the ancestors and compute another group-by on each unique pattern to get new count, sum($t[b]$) and sum($t[\hat{b}]$) values. These are shown in the four leftmost columns of Table 5. For example, (*,*,Banana) is an ancestor of itself (with count 5), of (*,Dawn,Banana) (with count 1), of (Sat,Dawn,Banana) (with count 1) and of (Sun,Morning,Banana) (with count 1). Thus, its count in Table 5 is eight, and sum($t[b]$) and sum($t[\hat{b}]$) are computed similarly.

Note that at this point, after generating the ancestors of all the LCAs, the resulting set of 20 patterns is exactly the data cube of s. This is our candidate set P referenced in line 6 of the algorithm.

In the fourth step, we "correct" the counts and sums so that they correspond to the true count of tuples matching each pattern and the true sums of the matching $t[b]$ and $t[\hat{b}]$ values. To do this, we consider the number of tuples in the sample that match each pattern. For example (*,*,Banana) matches two tuples in s, namely tuples 4 and 11, so we divide the aggregates we have computed for (*,*,Banana) by two. This gives the corrected $count = 4$, $sum(t[b]) = 3$ and $sum(t[\hat{b}]) = 2$. The rightmost four columns of Table 5 show the frequency in sample of each candidate pattern p (i.e., how many tuples in s it matches) and the corrected aggregates.

Finally, we select the pattern with the highest information gain based on the computed aggregates. We approximate the information gain of a pattern p as $count \times (sum(t[b]) \log \frac{sum(t[b])}{sum(t[\hat{b}])} + (1 - sum(t[b])) \log \frac{1-sum(t[b])}{1-sum(t[\hat{b}])})$. In our example, the pattern (Sat,*,*) has the highest information gain of $5 \times (0 + \log(\frac{1}{0.5})) = 5 \log 2$, so it is added to the explanation table.

The computational complexity of our approach is $O(|s| \times |T|)$ to compute the LCAs and their ancestors, plus $O(|datacube(s)|)$ to generate ancestors, plus $O(|datacube(s)| \times |s|)$ to join the patterns with the sample and compute the corrected aggregates. Note that $|datacube(s)|$, the size of the data cube of the sample s, is likely to be much smaller than $|datacube(T)|$. In fact, based on the results we reported in [6], our approach (called the "Flashlight Strategy" in that work) can be orders of magnitude faster than computing the information gain of all possible patterns in $datacube(T)$, with only slightly worse information gain.

An important aspect of our sampling-based approach is that sampling is used only to restrict the space of candidate patterns, not to compute their information gain. In other words, the $sum(t[b])$ and $sum(t[\hat{b}])$ aggregates are computed over the underlying dataset, not just the sample. In [6], we also investigated an even faster approach, termed "Laserlight", in which these aggregates are computed over the sample. This is faster

Table 4: Computing aggregates over the LCAs from Table 3 (second step of the algorithm) and generating ancestors (third step of the algorithm)

Pattern	count	sum($t[b]$)	sum($t[\hat{b}]$)	Ancestors
(*,*,*)	21	9	10.5	(*,*,*)
(*,*,Banana)	5	4	2.5	(*,*,Banana), (*,*,*)
(*,Dawn,*)	3	2	1.5	(*,Dawn,*), (*,*,*)
(Sat,*,*)	3	0	1.5	(Sat,*,*), (*,*,*)
(*,*,Oatmeal)	2	2	1	(*,*,Oatmeal), (*,*,*)
(*,Dawn,Banana)	1	0	.5	(*,Dawn,Banana), (*,Dawn,*), (*,*,Banana), (*,*,*)
(*,Dawn,Oatmeal)	1	0	.5	(*,Dawn,Oatmeal), (*,Dawn,*), (*,*,Oatmeal), (*,*,*)
(*,Morning,*)	1	0	.5	(*,Morning,*), (*,*,*)
(Sat,Dawn,*)	1	0	.5	(Sat,Dawn,*), (Sat,*,*),(*,Dawn,*),(*,*,*)
(Sat,Dawn,Banana)	1	0	.5	(Sat,Dawn,Banana), ..., (Sat,*,*), (*,Dawn,*),(*,*,*)
(Sun,*,*)	1	1	.5	(Sun,*,*), (*,*,*)
(Sun,Morning,Banana)	1	1	.5	(Sun,Morning,Banana), ..., (*,*,Banana), (*,*,*)
(Thu,Dawn,Oatmeal)	1	1	.5	(Thu,Dawn,Oatmeal), ..., (*,*,Oatmeal),(*,*,*)

because it does not require a join with the underlying table T to compute the LCAs; instead it simply computes a data cube over the sample s. However, we observed in [6] that the improvement in efficiency of "Laserlight" comes at a steep cost of information gain, making the strategy we described here ("Flashlight") a more effective choice for constructing informative explanation tables.

3 Experimental Results

In this section, we provide a brief experimental evaluation of the information content of explanation tables compared to related techniques. The experiments were performed on a 3.3 GHz AMD machine with 16 GB of Ram and 6 MB of cache, running Ubuntu Linux. We use the following datasets.

- `Upgrade`: 5795 United Airline ticket records obtained from `https://www.diditclear.com/`. Each record contains seven attributes about the tickets (origin airport, destination airport, date, booking class, etc.) and a binary attribute indicating whether the passenger was upgraded to business class. This data set has over 346,000 possible patterns.

- `Adult`: U.S. Census data containing demographic attributes such as occupation, education and gender, and a binary outcome attribute denoting whether the given person's income exceeded \$50K. This data set was downloaded from the UCI archive at `archive.ics.uci.edu/ml/datasets/Adult/` and contains 32561 tuples, 8 attributes and $436,414$ possible patterns.

We measured the information content of explanation tables and other summaries in terms of the *average information gain*. This is the improvement in KL-Divergence compared to having only the all-wildcards pattern and knowing only the fraction of tuples in the whole data set having outcome 1.

To compute explanation tables, we used the sampling-based approach from [6] which we described earlier, with a sample size of 16, which we observed to offer a good trade-off between running time and information gain on the tested datasets (we observed diminishing returns in terms of information content for larger sample sizes). The reported information gains are averages of five runs with different samples.

We compare explanation tables with four related approaches. The first three were compared in [6], whereas a comparison with the last one is a novel contribution of this paper.

Table 5: Computing aggregates over the LCAs and their ancestors (fourth step of the algorithm)

Pattern	count	sum($t[b]$)	sum($t[\hat{b}]$)	Frequency in sample	$count$	sum($t[b]$)	sum($t[\hat{b}]$)
(*,*,*)	42	21	21	3	14	7	7
(*,*,Banana)	8	6	4	2	4	3	2
(*,*,Oatmeal)	4	3	2	1	4	3	2
(*,Dawn,*)	8	4	4	2	4	2	2
(*,Morning,*)	2	1	1	1	2	1	1
(Sat,*,*)	5	0	2.5	1	5	0	2.5
(Sun,*,*)	2	2	1	1	2	2	1
(Thu,*,*)	1	1	.5	1	1	1	.5
(*,Dawn,Banana)	2	1	1	1	2	1	1
(*,Dawn,Oatmeal)	2	1	1	1	2	1	1
(*,Morning,Banana)	1	1	.5	1	1	1	.5
(Sat,*,Banana)	1	0	.5	1	1	0	.5
(Sun,*,Banana)	1	1	.5	1	1	1	.5
(Thu,*,Oatmeal)	1	1	.5	1	1	1	.5
(Sat,Dawn,*)	2	0	1	1	2	0	1
(Sun,Morning,*)	1	1	.5	1	1	1	.5
(Thu,Dawn,*)	1	1	.5	1	1	1	.5
(Sun,Morning,Banana)	1	1	.5	1	1	1	.5
(Sat,Dawn,Banana)	1	0	.5	1	1	0	.5
(Thu,Dawn,Oatmeal)	1	1	.5	1	1	1	.5

1. Decision trees, as an example of a human-interpretable classifier. Each root-to-leaf path in a decision tree can be thought of as a pattern (of values of the dimension attributes), with the leaf node providing a prediction for the value of the outcome attribute. We computed the information gain of decision trees by comparing the predicted values of the outcome attribute with the true values.

   We used the Weka J48 implementation [10] of the C4.5 algorithm, which provides a *confidence* parameter to control pruning and the number of tuples a leaf node can cover. We considered both multi-way split decision trees in which a node can have many children and binary split decision trees in which a node has exactly two children. Since there is no direct way to enforce decision tree size in Weka, we performed a grid search over the parameter space to obtain trees with specific numbers of root-to-leaf paths. To do this, in our experiments, we varied confidence in the range $[0.01 - 0.59]$ and the minimum number of tuples per leaf in the range $[2 - 20]$.

2. The SURPRISE operator for data cube exploration [13], which identifies regions of a datacube whose distribution of the outcome variable is different than the distribution in the regions already seen by the user. As in explanation tables, these regions are represented by patterns of values of the dimension attributes. However, while SURPRISE also uses information gain to find informative patterns, for efficiency reasons it only considers non-overlapping patterns or patterns that are fully contained in each other (corresponding to rolling-up or drilling-down in the datacube). We implemented this technique in C++ based on the algorithm given in [13]. For a fair comparison with explanation tables, we start with the knowledge of the average value of the outcome attribute in the dataset, and we request the k most informative patterns.

   The SURPRISE operator includes two parameters: the desired number of patterns and an accuracy threshold ϵ. A lower ϵ indicates a finer algorithm resolution but slower runtime. To fairly represent SURPRISE, we have chosen two values for ϵ for each data set: SURPRISE_cmp, which uses ϵ that gives comparable

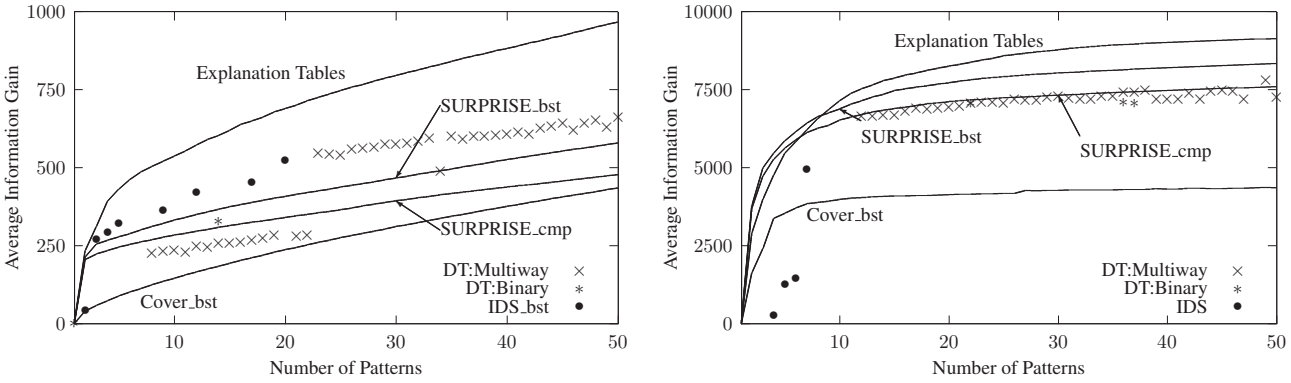

Figure 1: Average information gain of various methods using `Upgrade` (left) and `Adult` (right)

runtime to *Flashlight*, and SURPRISE_bst, which uses ϵ that yields the best possible information gain without running out of memory.

3. Data summarization techniques from [8, 9] which find the fewest patterns that cover most of the tuples with outcome 1. These methods require a confidence threshold that denotes the minimum fraction of 1-outcomes per pattern. We experiment with a variety of confidence thresholds, including $\{0.4, 0.5, \ldots, 0.9, 0.95, 0.99\}$, and report the runtime of the one with the best information gain, denoted Cover_bst. After generating the summaries, we apply the maximum entropy principle to derive an estimated distribution of the outcome attribute and we compute information gain based on this.

4. Interpretable Decision Sets (IDS) [11], as another recent example of a human-interpretable classifier. An IDS is a set of independent if-then classification rules. We obtained the source code from the authors' Github repository[2]. The IDS generation algorithm includes a support threshold ϵ, which determines the lowest possible support of a given classification rule. We started with $\epsilon = 1$ and considered lower values until the runtime became too long. We varied ϵ in decrements of 0.02 until the runtime exceeded one hour for the `Upgrade` dataset and three hours for the larger `Adult` dataset. As with decision trees, we computed the information gain of IDS by comparing the predicted values of the outcome attribute with the true values.

Figure 1 plots the average information gain using `Upgrade` (left) and `Adult` (right) as a function of the number of patterns (or, in the case of decision trees, the number of leaves). Decision trees are represented as individual points on the graphs, corresponding to the numbers of leaves we were able to obtain using the grid search explained above. Similarly, IDS results are represented as individual points that indicate the best possible information gain we could obtain using the above grid search. On the other hand, SURPRISE and Cover allow us to control the number of patterns, so these are represented as lines. We conclude that explanation tables outperform all other approaches in terms of information gain, except for small numbers of patterns (below 8) on the `Adult` data set, in which case SURPRISE has a slightly higher information gain. This is likely due to the fact that explanation tables explicitly optimize for information content. On the other hand, decision trees and SURPRISE are restricted to non-overlapping patterns; Cover optimizes for the number of patterns that cover all the positive examples without considering information gain; and IDS is a classifier that optimizes jointly for predictive power, conciseness and interpretability.

In terms of running time, it took 47 seconds on `Upgrade` and 252 seconds on `Adult` to generate explanation tables with 50 patterns. SURPRISE_cmp took roughly the same time (by design), whereas SURPRISE_bst

[2]https://github.com/lvhimabindu/interpretable_decision_sets

was 2-4 times slower and Cover_bst was 5-8 times slower. On the other hand, decision trees and IDS took much longer (several hours) since they had to be executed multiple times with different parameter values in order to obtain a desired number of patterns.

4 Discussion

We emphasize that our problem is different from prediction since explanation conciseness is also critical. Hence, instead of focusing on prediction accuracy we focus on interpretability, which is a relative notion that captures an inherent trade-off between accuracy and conciseness: given two explanations, E and E', if E is at least as accurate as E' and E is more concise than E', we say that E is more interpretable than E'. While there exist some classifiers that could be considered human-interpretable as a secondary goal (e.g., decision trees and partial monotonic functions), in practice they are used in ensembles which hinders their interpretability. Therefore, explanation tables may have a potentially interesting role to play when the outcome attribute corresponds to the predictions from an ensemble method, or from an even more opaque model such as variants of neural networks, since they may enable auditing or debugging the model. This application would be an interesting direction to explore for future work. Future work could also include a user study to evaluate the explanatory power of explanation tables for various tasks (e.g., why was this example classified as such?) as well as the extension to (unordered) categorical outcome attributes.

References

[1] A. Berger: The improved iterative scaling algorithm: A gentle introduction, 1997.

[2] J. N. Darroch and D. Ratcliff: Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 1470-1480, 1972.

[3] M. Das, S. Amer-Yahia, G. Das and C.Yu: MRI: Meaningful Interpretations of Collaborative Ratings, 1063–1074, 2011.

[4] D. Fabbri and K. LeFevre: Explanation-Based Auditing. *P*VLDB 5(1): 1–12 (2012)

[5] G. Feng, L. Golab, D. Srivastava: Scalable Informative Rule Mining. ICDE 2017: 437-448

[6] K. El Gebaly, P. Agrawal, L. Golab, F. Korn, D. Srivastava: Interpretable and Informative Explanations of Outcomes. PVLDB 8(1): 61-72 (2014)

[7] K. El Gebaly, L. Golab, J. Lin: Portable In-Browser Data Cube Exploration. IDEA workshop at KDD 2017: 35-39

[8] L. Golab, H. Karloff, F. Korn, D. Srivastava, B. Yu: On Generating Near-optimal Tableaux for Conditional Functional Dependencies. *PVLDB* 1(1): 376-390 (2008)

[9] L. Golab, F. Korn, D. Srivastava: Efficient and Effective Analysis of Data Quality using Pattern Tableaux. *IEEE Data Eng. Bull.* 34(3): 26-33 (2011)

[10] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten: The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1): 10-18 (2009)

[11] H. Lakkaraju, S. H. Bach, J. Leskovec: Interpretable Decision Sets: A Joint Framework for Description and Prediction. KDD 2016: 1675-1684

[12] S. Roy, L. Orr, D. Suciu: Explaining Query Answers with Explanation-Ready Databases. *PVLDB* 9(4): 348–359 (2015)

[13] S. Sarawagi: User-Cognizant Multidimensional Analysis. *VLDB Journal* 10(2-3): 224-239 (2001)

[14] X. Wang, X. L. Dong, A. Meliou: Data X-Ray: A Diagnostic Tool for Data Errors. SIGMOD 2015: 1231–1245