Mining Logical Arithmetic Expressions From Proper Representations

Eitan Kosman* Ilya Kolchinsky† Assaf Schuster*

Abstract

Logical-arithmetic expressions are convenient for describing phenomena due to their expressiveness and comprehensibility. Therefore, we propose to target mining logical arithmetic expressions through a novel task called Logical-arithmetic expression mining (LAEM). Its goal is to discover expressive logical expressions that are representative for a database. It accepts a complex database as input and returns a set of representative expressions for the database. Driven by the success of machine learning models to recognize complex patterns, we argue that a thorough modeling of the learned representations could be exploited for generating interesting and representative mathematical expressions. To address this, in this paper we propose Soft dEcision Tree for logical arithmetic Expressions miNing (SEEN), an algorithm based on representation learning for generating logical expressions. Our mining mechanism partitions the learned representation space and assigns self-labels. Then, we use the self-labels to train a multivariate soft decision tree from which we generate logical arithmetic expressions. A comprehensive experimental study on 2 diverse real-world datasets shows that the proposed method is able to generate interesting expressions. The implementation is publicly available[1].

1 Introduction

The recent explosion in data volume has triggered rapid development of new data mining methods. Generally, we aim to discover knowledge hidden in a large database about an interesting feature or a complex combination of elements. In many cases, using logical expressions to describe the results, namely patterns, is convenient because of their interpretive abilities.

The pursuit of pattern mining algorithms is not new. As time progresses, more algorithms are proposed for mining patterns of different objectives, such as frequency [1] and high-utility [2], as well as multiple-truths with multiple objectives and constraints [3]. Meanwhile, the choice of objective remains an active research field. Indeed, sometimes we barely know how to define the objective, but we still need a set of representative patterns. Using common objectives such as frequency or correlation yields many redundant patterns which are not representative. Therefore, an alternative definition of objective is much needed.

The success of recent data-driven models in pattern recognition implies that such models hide useful data representations [4] that can be exploited for pattern mining and provide an alternative objective. A representation learned during solving a task could reveal patterns that are neither frequent nor correlated. Additionally, data representations can be learned without annotated data in a self-supervised or unsupervised manner. This opens the door to new possibilities of designing pattern mining algorithms whose objectives are very expressive, attained by targeting a learned prior rather than the raw data.

To date, pattern mining algorithms deal with diverse data types such as itemset mining [5] and sequential patterns [6]. However, they are restricted to discrete transactional databases, whereas continuous data becomes more common, sensor networks for instance. Besides, there is a growing interest in methods for representation learning of data [4]. However, they do not provide comprehensible patterns and the learned representations remain obscure. As we show in this paper, using logical expressions with arithmetic operators for describing complex relations reflected in the learned representation is highly promising for this purpose.

The motivation for mining logical arithmetic expressions from learned representations has more natural origins. First, interestingness of patterns is subjective. The ability to learn a task-specific representation makes representation learning a highly promising avenue for pattern mining. Second, many existing algorithms suffer from the pattern explosion problem in which an enormous amount of redundant and meaningless pattern candidates are generated. We identify the potential of mitigating it by training deep neural networks which are also researched in depth for scalability. Third, in recent years, a number of technologies have emerged for detecting a priori given expressions of the sort discussed above in real time, such as *complex event processing (CEP)* [7–12]. However, little work has been

*Computer Science Department, Technion Israel Institute of Technology. {eitan.k,assaf}@cs.technion.ac.il

†Red Hat Inc. ikolchin@redhat.com

[1] `https://github.com/ekosman/SEEN`

done to develop methods for mining these expressions from data. Mining complex expressions in the form of logical expressions could close this gap and exploit the benefit of this system by feeding it with the mined expressions for detection and perform proactive actions.

To summarize, the task we tackle is mining logical arithmetic expressions from learned representations. Although much research has been done on representation learning, our goal is different because we further focus on transforming it into logical expressions. Our algorithm, **S**oft d**E**cision Tree for logical arithmetic **E**xpressions mi**N**ing (SEEN), is based on entangled learning of data representation along with discovering comprehensible descriptions. SEEN is composed of 2 core components: 1. a black-box model for learning a representation of the data, and 2. a surrogate model for mining patterns from the obtained representation.

Contributions The main contributions of this paper are summarized as follows:

1. We formulate the problem of logical arithmetic expressions mining and propose SEEN, an algorithm for mining logical arithmetic expressions from learned representations.

2. We extend the expressiveness of patterns generated via traditional pattern mining methodologies. This introduces a new research opportunity for the development of pattern mining mechanisms through definitions of task-oriented objectives.

3. The method is validated through a series of extensive experiments on real-world datasets. We show that patterns generated by SEEN are qualitatively and quantitatively meaningful. We analyze the impact of hyper-parameters on the performance and the quality of the generated patterns.

2 Preliminaries and Related Works

Representation Learning [4] facilitates the identification of patterns through high-level abstraction. Three main approaches exist: unsupervised, self-supervised, and supervised learning. Classic unsupervised approaches include clustering algorithms such as K-means and Gaussian Mixture Model. In deep-learning, autoencoders compactly encode data while preserving significant features for reconstruction. Recently, there is an interest in self-supervised learning [13] that relies on natural properties of the data for labeling. For example, the embeddings learned through predicting image rotations [14] were found useful for classification from limited amount of labeled data, Contrastive predictive coding [15] learns to encode time series by forecasting and DeepWalk [16] learns vertex representations through local information from random walks. Lastly, supervised learning makes use of representation learning in transfer learning [17] by taking advantage of the learned representation for adapting to new tasks through fine-tuning.

Soft Decision Trees (SDTs) [18] extend Decision Trees [19] by allowing data points to have static continuous membership distribution on all decision paths. Irsoy *et al.* [20] introduced dynamic distribution on the decision paths by conditioning soft decisions on the input. This brings the power of conditioning each decision on a multivariate function, increasing the expressiveness by allowing axis-unaligned decisions. As the decision functions are not conceptually restricted, others [21] use complex functions, *e.g.* neural networks. Adaptive neural trees [22] introduce transformation functions in the tree nodes that increase the expressiveness of the model. These transformations transform the input to a new domain, serving as input to the decision function and passed to the descendants for further computation.

Pattern Mining is a broad term encompassing techniques for extraction of knowledge from large databases, such as sequential pattern mining [6] and itemset mining [5]. Early algorithms include the Apriori algorithm for frequent association rules mining [23], as well as FP-growth [24], which is an efficient alternative. Over the years, many extensions supporting items with multiple attributes have been proposed [25] as well as algorithms supporting multi-item connections within a sliding window [26]. Representative pattern mining refers to generating patterns that are significant, such as patterns that have the greatest discriminative power [27]. Explainable clustering, a topic related to us due to the similarity of clusters and patterns, focuses on revealing clusters with statistically significant features that characterise the clusters. However, works on this topic [28] are restricted to uni-variate expressions due to their usage of uni-variate decision trees.

3 Problem Definition

In this section, we describe the problem considered in LAEM. We begin with a formal definition and then motivate it with an example. For the purpose of organization of the paper, we list the necessary notations in table 1.

3.1 Problem Statement Let $\mathcal{T}$ be a task (*e.g.* classification, reconstruction, or forecasting), and let $\mathcal{D}$ be a database containing training examples. We aim to generate patterns from the data representation learned during solving $\mathcal{T}$. For the purpose of this paper, we define a pattern as follows:

Definition 1. (Pattern) *Let* $S \subseteq \mathbb{R}^d$ *be a set of*

Table 1: Symbols and Notations.

Symbol	Definition
F	A feature extractor
G	Task solver on top of F
$Im(F)$	The image space of the function F
$\mathcal{D}$	Database of m data samples for training
γ	A reparameterization function for x_i
$\mathcal{T}$	A task which the model H aims to solve
P	A pattern / mathematical expression
$\mathcal{P}$	A space partition
k	Number of nearest neighbors
h	Height of a decision tree

points and let $X \subset S$ *be a proper subset of* S. *A pattern* P *is characterized by a phrase* $\varphi_X : \mathbb{R}^d \to \{True, False\}$ *that satisfies*

$$\varphi_X(x) = \begin{cases} True, & x \in X \\ False, & x \in S \setminus X \\ Undefined, & otherwise \end{cases}.$$

Usually, the phrase φ_X is a conjunction of conditions, as we exemplify in the following.

3.2 Example Scenario This example is shown in Figure 1. Consider a car company that strives to learn driving patterns of drivers who use its vehicles. It defines a task $\mathcal{T}$ for predicting whether a driver is going to be involved in a car accident given the readings from the sensors of the car. The company uses the LAEM mechanism and obtains the following patterns:

1. Sudden braking: $acceleration < -15\ [\frac{km}{sec\ hour}]$;
2. Turning left in tight corners:
   $steer > 130\ [deg]\ \wedge speed < 15\ [\frac{km}{hour}]$;
3. Dangerous maneuver:
   $|steer| > 130\ [deg] \wedge |acceleration| > 25\ [\frac{km}{sec\ hour}]$;

Notice that nothing but the definition of the task was required for the algorithm in order to find those patterns. This knowledge enables the design of safety equipment and alert systems by later using these expressions for detecting a driver at risk.

4 SEEN - Overview

In this section we present SEEN. As depicted in Figure 2, it separates the mining process into two core phases. The first phase intends to learn a good representation of the data with the guidance of learning to solve a task $\mathcal{T}$. Specifically, a good representation is: **1) Informative** - useful for solving $\mathcal{T}$ ($\mathcal{T}$ is easily solvable on top of it), and **2) Simple** - easily partitionable (identifying distinct patterns is easy).

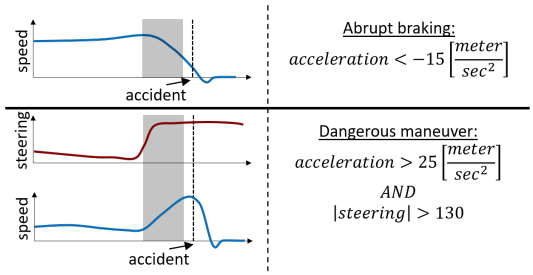


Figure 1: Discovering patterns from car sensors data. The obtained patterns *abrupt braking* and *dangerous maneuver* are followed by an event that defines the patterns. Notice that the terms *abrupt braking* and *dangerous maneuver* are not a result of the algorithm. However, it is meaningful to use those names for the conditions defined over the attributes of the patterns.

Equipped with a good representation, we move to the second phase to generate logical arithmetic expressions on top of it. The informativity property encourages the generated expressions to be significant for $\mathcal{T}$. The simplicity property will be used for partitioning the representation space. Each partition will be associated with a self-label. The self-labels will be used for training a soft decision tree (SDT), from which we generate expressions through extraction of its decision rules.

4.1 Representation Learning Phase

4.1.1 Informativity Motivated Training Suppose we are given a dataset $\mathcal{D}$ and a task $\mathcal{T}$. We train $H = G \circ F$, implemented as a composition of two deep neural networks to solve $\mathcal{T}$. We follow the standard setting of learning the parameters of H by the empirical risk minimization approach and minimize some predefined loss function, $\mathcal{L}_{\mathcal{T}}$, associated with $\mathcal{T}$.

4.1.2 Simplicity Motivated Training We minimize a distance-based contrastive loss component as follows. Given a batch of data points $B \subseteq \mathcal{D}$, let $B_F = \{F(x) | x \in B\}$. For each $r \in B_F$ we define $\mathcal{N}_k(r)$ as the neighborhood of r which consists of its k nearest neighbors with respect to their euclidean distance from r. In turn, we aim to encourage F to create unscattered neighborhoods, which are expressed by clusters that are compact and well separated. We do so by conditioning the neighbors on their proximity to r:

$$\mathcal{L}_{simplicity} = \sum_{r \in B_F} \left(\sum_{r^{(1)} \in N_k(r)} -\log P(r^{(1)}|r) \right), \tag{4.1}$$

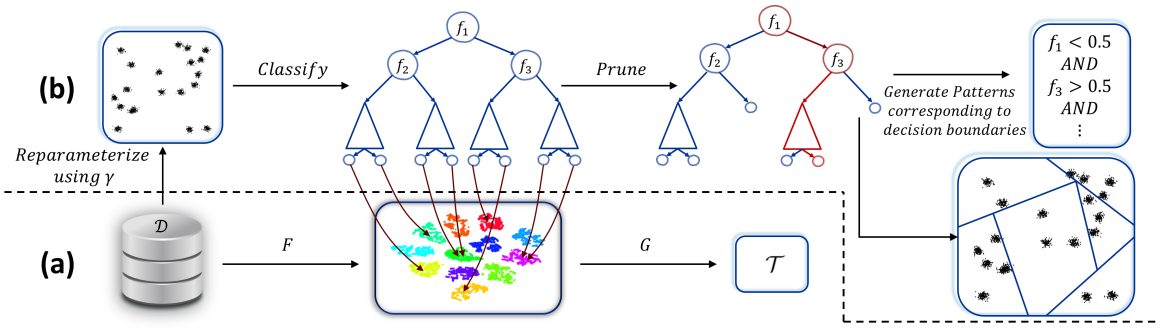


Figure 2: Flow overview of SEEN. **(a)** We train $H = G \circ F$ to solve $\mathcal{T}$, encouraging unscattered representation via minimizing both $\mathcal{L}_{simplicity}$ and $\mathcal{L}_{\mathcal{T}}$. **(b)** Partitioning $Im(F)$ and using the obtained partition for self-labeling. Then, use the self-labels for training a soft decision tree. The sibling leaves that point to the same cluster are pruned. Finally, mathematical expressions are extracted from the tree as conjunctions of the decision functions.

where $P(r^{(1)}|r)$ is parameterized by *softmax* with negative samples from B_F :

$$P(r^{(1)}|r) = \frac{e^{\ jr\ \ r^{(1)}j_2^2}}{\sum_{r^{(2)}2B_F nfrg} e^{\ jr\ \ r^{(2)}j_2^2}}. \tag{4.2}$$

By minimizing Equation 4.1, we specifically encourage maximizing the expression in Equation 4.2 which in turn minimizes distances between neighbors in the nominator, and pushes apart non-neighbors in the denominator. Extensions of this objective function arise by adding more constraints. For instance, in classification problems we can constrain neighborhoods by forcing homogeneity with respect to the target label, *i.e.* $\forall r^0 \in \mathcal{N}_k(r) : y_r = y_{r^0}$, where y_r, y_{r^0} are the target labels associated with r, r^0 respectively.

4.1.3 Summing Up the Representation Learning Stage

We obtain the parameters of H, Θ_H, through training batch-wise to minimize $\mathcal{L}_{\mathcal{T}}$ and $\mathcal{L}_{simplicity}$ via the following objective:

$$\hat{\Theta}_H = \underset{\Theta_H}{\operatorname{argmin}}\, \mathcal{L}_{\mathcal{T}} + \alpha \cdot \mathcal{L}_{simplicity}. \tag{4.3}$$

4.2 Expressions Generation Phase

4.2.1 Partitioning

The learned representation $\mathcal{D}_F = \{F(x)|x \in \mathcal{D}\}$ is used for creating a finite partition $\mathcal{P}$ of $Im(F)$. We use this partition for self-labeling of the data points. Formally, let $\mathcal{P} = \{X_1, ..., X_l\}$ be a partition of $Im(F)$ into l sub-spaces. Given a reparameterization function γ, we define the self-labelled dataset $\mathcal{D}_{sl} = \{(\gamma(x), y)|x \in \mathcal{D} \wedge F(x) \in X_y\}$ with reparameterized points originating from $\mathcal{D}$ and labels representing the membership to one of the l partitions. The choice of γ is up to the user and can also be left as the identity function. In our experiments, we use various clustering algorithms for partitioning, including *Gaussian Mixture Model* with selection of optimal number of clusters by computing the Davies-Bouldin-Index and *DBSCAN*.

4.2.2 Training a surrogate SDT

We use the self-labeled dataset $\mathcal{D}_{sl}$ for training a SDT to predict the membership of a data point to a sub-space. The SDT is initialized as a full binary tree of height h. Each internal node b is associated with a routing function $f_b(x) = \sigma(w_b^T x + t_b)$ (not necessarily restricted to this choice), which takes a data point x as input and outputs the probability of routing it to one of its descendants. Here, w_b, t_b are the parameters of the decision function of an internal node and $\sigma(x) = \frac{1}{1+e^{\ x}}$ is the sigmoid function. In addition, the decision function in a leaf predicts the label of x. We parameterize the decision function of a leaf node b as $D_b = w_b$, where $w_b \in \mathbb{R}^l$. Moreover, the decision of an internal node is computed as a convex combination of the decisions of its descendants:

$$D_b(x) = f_b(x) \cdot D_{b_{left}}(x) + (1 - f_b(x)) \cdot D_{b_{right}}(x). \tag{4.4}$$

Here, b_{left} and b_{right} denote the left and right descendants of b. Figure 3 illustrates the prediction process.

We follow the SDT training procedure described in [21]. For a leaf node b, let $p_b = \{r, ..., b\}$ denote the path from the root r to b, and let $P(p_b|x)$ be the path probability calculated as a product of the soft decision functions through p_b. Assuming the target label of x is y, the expected error of predicting the true label is

$$\mathcal{L}_{tree} = \sum_{b2Leaves} P(p_b|x) \cdot CE(f_b(x), y), \tag{4.5}$$

where $CE(\cdot, \cdot)$ is the cross-entropy between the distribution predicted by the leaf node and the one-hot encoded

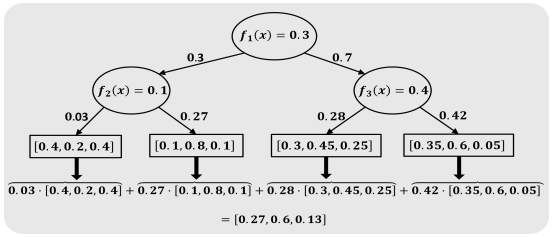
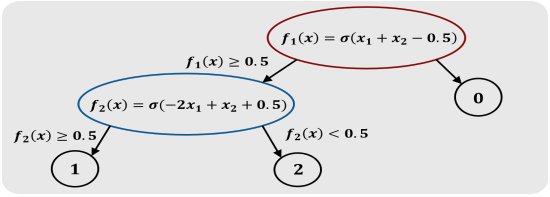
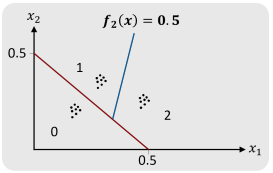


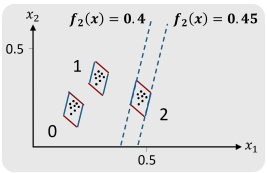
Figure 3: An example of an SDT. The decision function in each leaf is given by a fixed vector of label probabilities. Edges represent cumulative decision weights for the node descendants. The final decision is a weighted average of the label probabilities of the leaf nodes.



(a) SDT with untightened decision functions



(b) Untightened



(c) Tightened

Figure 4: Visualization of the tree's decision functions (4a). The decision boundaries partition the space into subspaces containing regions that do not represent patterns (4b). Figure 4c shows the tightened boundaries.

vector representing the true distribution of the target label y. In addition, in order to encourage simple routing functions, we consider sparse solutions using the pruning method proposed in [29]. In short, it prunes the weights with small absolute magnitude after each training episode. Finally, the objective is to find the set of parameters representing all the decision functions in the tree so that $\hat{\Theta}_{tree} = \text{argmin}_{\Theta_{tree}} \mathcal{L}_{tree}$.
Here, Θ_{tree} denotes the union of the parameters of the whole tree including the parameters of the leaves.

4.2.3 SDT Pruning Two leaves that descend from the same node and represent the same label can be pruned without changing the prediction of the tree. We perform a post-order traversal on the tree and prune leaves of the same label. This results in shallower trees that are simpler in terms of path lengths and number of parameters, which in turn enhances the comprehensibility of the generated expressions. We provide a pseudo-code of this procedure in the supplementary materials.

4.2.4 Tight Expressions Extraction The SDT allows us to generate expressions that describe its decision paths. That said, we use maximum likelihood estimation for assessing the membership of a data point to a leaf, formally defined as $X_b = \{x \in \mathcal{D} | \text{argmax}_{b' \in Leaves} P(p_{b'}|x) = b\}$. Then, we convert the soft decision at each internal node to hard decision using:

$$D_b(x) = \begin{cases} D_{b_{left}}(x), & f_b(x) \geq 0.5 \\ D_{b_{right}}(x), & \text{otherwise} \end{cases} \tag{4.6}$$

Next, for each path p_b so that $b \in Leaves$, we extract the set of decision functions incorporated into the path nodes. Since the decision boundaries are shared across various paths, the conditions extracted using this methodology are not tight, as shown in the example depicted in Figure 4. We address this by tuning the decision thresholds of each decision function with respect to the data points X_b associated with each leaf. For example, in Figure 4 we tighten the decision boundaries using data points associated with pattern 2. The decision boundary $f_2(x) < 0.5$ is converted to $f_2(x) \in [0.4, 0.45]$. This is done by iterating over X_b for each decision function and finding the minimum and maximum thresholds of the underlying data. The thresholds tuning algorithm is detailed in procedure *tuneDecisions* in the supplementary materials.

4.2.5 Summing Up the Expressions Generation Stage The pattern generation stage is summarized in Algorithm 1. A discussion on the hyper-parameters is provided in the supplementary materials.

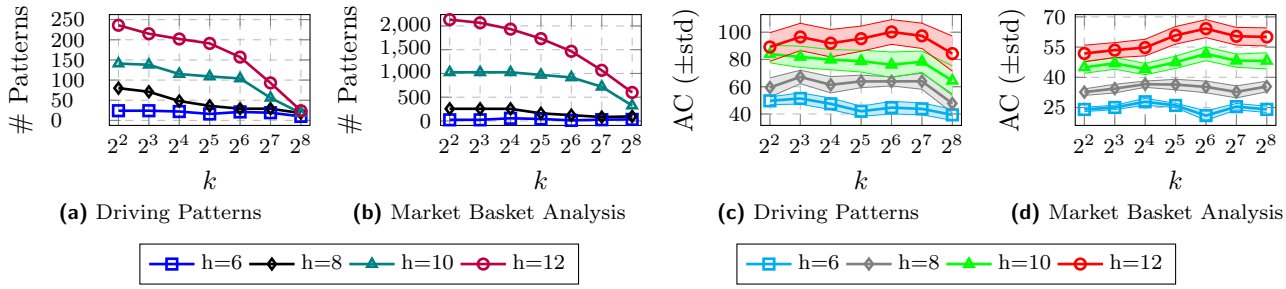
Algorithm 1 Expressions Generation

Input: $\mathcal{D}$, F, γ, h
Output: *A set of expressions*

$\mathcal{D}_F \leftarrow \{F(x) | x \in \mathcal{D}\}$
$\mathcal{P} \leftarrow partition(\mathcal{D}_F)$
$\mathcal{D}_{sl} \leftarrow \{(\gamma(x), y) | x \in \mathcal{D} \wedge x \in X_y \in \mathcal{P}\}$ } Sec. 4.2.1

$tree \leftarrow trainSDT(h, \mathcal{D}_{sl})$ ➤ Sec. 4.2.2
$tree \leftarrow decisionWisePruning(tree)$ ➤ Sec. 4.2.3
$assignToLeaves(\mathcal{D}, tree)$
$Exps \leftarrow \emptyset$
for $b \in Leaves$ **do**
  $funcs \leftarrow \{v.function | v \in p\}$
  $funcs \leftarrow tuneDecisions(funcs, b.X)$
  $Exps \leftarrow Exps \cup \{funcs\}$
} Sec. 4.2.4

return $Exps$



(a) Driving Patterns **(b)** Market Basket Analysis **(c)** Driving Patterns **(d)** Market Basket Analysis

Figure 5: Left: Number of generated patterns under varied number of neighbors (k) and tree heights (h). Typically, the number of patterns decays as the number of neighbors grows due to the larger neighborhoods, which reduce the number of self-labels. Additionally, a shallower tree initialization results with fewer patterns due to the reduction in the number of leaves. Right: Average comprehensibility (AC). The AC has a very moderate downward trend as the number of neighbors grows. Apparently, it grows linearly as the SDT gets deeper. Its standard deviation gets bigger as h grows (represented as the filled area), suggesting that deeper trees are pruned more aggressively.

5 Experimental Evaluation

In this section, we evaluate SEEN in several experiments. Specifically, our goals are the following:

- Qualitatively examine the generated patterns. We present case studies with interesting insights.
- Analyze the impact of the various parameters on the quality of the generated patterns.

Additional implementation details are provided in the supplementary materials.

5.1 Evaluation Metrics Traditional pattern mining algorithms [5,6] are evaluated in terms of their objective. However, such metrics are not applicable for us because the generated patterns are not intended to be frequent nor correlated. Instead, we strive to achieve other qualifications. As *Occam's razor* encourages simple explanations, we evaluate the *Comprehensibility* [30].

Definition 2. (Comprehensibility) *The comprehensibility of a pattern P is the number of elements P is composed of. For example, the comprehensibility of the pattern $[[X+Y+W>3]\ AND\ [Z+W<2]]$ is 5.*

We also consider counting the number of generated patterns because too many results makes it difficult to examine the generated patterns. Typically, a simple model that generates a reasonable amount of patterns would be considered to be better than a model that generates too many (redundant) patterns.

5.2 Case Study: Driving Patterns (Self-Supervised) Dataset: Comma.ai [31] captures over 33 hours of driving data. It is segmented into 2019 1-minute-long segments of highway driving containing GPS, thermometers, 9-axis IMU and CAN-bus data. In total, the dataset contains readings from 18 sensors.

Preprocessing: All the signals are interpolated and then resampled at 200 $[HZ]$. Additionally, all the signals are normalized by dividing them by the maximum absolute value of the corresponding signal. Then, a sliding window of length 2000, which represents a time frame of $\frac{2000}{200} = 10$ $[seconds]$ is used in order to segment the samples. This results in 101465 samples in total. We further split the dataset by randomly selecting 25% of the samples for the pattern generation stage.

Representation Learning Methodology: We employ Contrastive Predictive Coding [15] for representation learning of the sensors data. Our model is based on the architecture described in [15], consisting of a 4 layer 1D convolutional stack with fixed kernel sizes of 5, stride of 2 and an increasing order of output channels: $18 \to 32 \to 64 \to 128 \to 256$. Consequently, every element of the convolutional stack output has a receptive field of $2^4 = 16$ time units, *i.e.* 80 $[ms]$. Then follows a GRU layer [32] with a hidden state of size 256 whose last output is used for multi-horizon forecasting of the next 30 timesteps. For the rest of this section, we use the last element of the series predicted by the GRU layer as the learned representation. In all experiments in this case, the amount of samples in every batch remains fixed at 512, while varying k, the size of the neighborhood.

Partitioning: A Gaussian Mixture Model (GMM) with various number of clusters is trained for clustering the obtained embeddings and the number of clusters is determined by selecting the model corresponding to the lowest Davies-Bouldin-Index. The obtained GMM is used for re-labeling the data by assigning cluster indices.

Reparameterization (γ): In order to simplify the generated patterns, each window is resampled at 1 $[HZ]$ while maintaining a context length of 10 seconds, that is, each sample results in a series of 10 equally-spaced

$0.48 \leq 6.49 \cdot T_0.steering\_angle + 5.89 \cdot T_2.steering\_angle + 6.6 \cdot T_4.steering\_angle \leq 0.69$ AND $2.02 \leq 1 \cdot T_0.accelerometer_{down} + 1 \cdot T_7.accelerometer_{down} \leq 2.31$
$2.605 \leq -T_9.wheel\_speed_{front\ left} + 1.15 \cdot T_5.accelerometer_{down} \leq 2.63$ AND $0.847 \leq -1 \cdot T_8.gyro_{forward} - 0.797 \cdot T_9.gyro_{forward} + 7.1 \cdot T_3.gyro_{forward} \leq 5.23$
$0.848 \leq 1 \cdot T_2.gyro\_down \leq 0.91$ AND $36.72 \leq 1 \cdot T_1.gyro\_right \leq 36.75$ AND $0.19 \leq 1 \cdot T_5.accelerometer_{forward} + 8.72 \cdot T_2.gyro_{forward} \leq 0.98$
$0.55 \leq 1 \cdot T_5.gyro_{forward} + 1 \cdot T_9.gyro_{forward} \leq 0.63$ AND $0.023 \leq 1 \cdot T_1.gyro_{down} + 1 \cdot T_9.gyro_{down} \leq 0.049$ AND $2.45 \leq 1 \cdot T_5.accelerometer_{forward} + 8.69 \cdot T_2.gyro_{forward} \leq 2.46$

Table 2: Example patterns for the comma.ai dataset. Units are: gyro$[\frac{rad}{sec}]$, accelerometer$[\frac{meter}{sec^2}]$, speed$[\frac{meter}{sec}]$, steering$[deg]$. T_i represents a collection of attributes containing sensors readings. Specifically, we use 10-seconds time windows sampled at $1[HZ]$, thus T_i contains the sensors readings at the i^{th} second from the window beginning.

timestamps $\{T_0, ..., T_9\}$.

SDT Pruning: We prune the parameters of the tree after every single episode so that each decision function retains at most 5 non-zero weights.

Results: We begin with a qualitative examination of patterns generated by SEEN. The patterns are shown in Table 2. The first row contains an expression that focuses on the accelerometer and steering angle information from various time stamps. Its first condition relates to the steering angle in the beginning of the sequence (T_0), after 2 seconds (T_2) and after 4 seconds (T_4). The contrast of their coefficients combined with the limited values range of the expression and the short time frame implies that the driver steers the wheel sharply. Moreover, the second condition of the first expression focuses on the readings of the accelerometer axis pointing down. This is interesting because acceleration change in this direction is often a result of a change in the down force of the car. This settles with the first condition because the down force varies sharply during rapid maneuvers. The second row contains an expression that focuses on wheel speed, accelerometer and gyro. Notice that only the front-left wheel sensor participate in this expression. This is because we used weights pruning [29] that has found the other wheel sensors to be redundant. The gyro facing forward in the second condition indicate the slope of the road. Specifically, the inversion of the sign of the coefficients between T_3 and the later timestamps T_8, T_9 imply that the driver encounters a sloppy road.

We continue the study by examining the impact of the parameters on the results. Figure 5a shows the number of resulting patterns for various number of neighbors k and SDT height initializations h. Unsurprisingly, the number of patterns decays for a shallower initialization since a shallower tree results in less leaves. However, although the number of patterns is expected to increase exponentially as h grows, it seems that the pruning methodology discussed in Section 4.2.3 softens it. Another unsurprising phenomenon is that the number of patterns decays as k grows. This is attributed to the bigger neighborhoods resulting in fewer self-labels being created in Section 4.2.1.

The last series of experiments examines how the comprehensibility (Definition 2) is affected under varied parameters, shown if Figure 5c. Again, we unsurprisingly observe that higher tree initializations lead to higher comprehensibility. This is attributed to the number of decision rules that grow linearly with respect to the height of the tree. As for the impact of the neighborhood size (k) on the comprehensibility, we observe no significant effect.

5.3 Case Study: Market Basket Analysis (Unsupervised)

Dataset: Groceries [33] contains purchase orders from the grocery stores. The total number of grocery types is 167 and there are 38765 records. We group orders by the buyer ID and date, resulting with 14963 baskets, each of which contains multiple groceries.

Preprocessing: Each record is encoded as a binary vector $\{0, 1\}^{167}$. 1 indicates the presence of an item in the transaction. We do not split this dataset in order to maintain the distribution consistent during both representation learning and pattern generation phases.

Representation Learning Methodology: We train an autoencoder for reconstruction while randomly omitting items with the probability of 0.5. Our model is composed of an autoencoder with the following hidden layer sizes: $167 \rightarrow 128 \rightarrow 89 \rightarrow 50 \rightarrow 89 \rightarrow 128 \rightarrow 167$. Learning is done via minimizing the binary cross entropy between the autoencoder output and the original binary representation vector of the unomitted record. For the rest of this section, we use the intermediate embedding of size 50 as the learned representation. In all experiments in this case, the amount of samples in every batch remains fixed at 512.

Partitioning: We partition the learned representation space via DBSCAN.

Reparameterization (γ)**:** We do not reparameterize this dataset (γ can be referred as the identity).

SDT Pruning: We prune the parameters of the tree after every single episode so that each decision function retains at most 4 non-zero weights.

$meat > 0.5$ AND $hamburger\ meat > 0.5$ AND $rolls/buns > 0.5$ AND $cream\ cheese \leq 0.5$ AND $curd \leq 0.5$ AND $coffee \leq 0.5$
$frozen\ vegetables > 0.5$ AND $semi-finished\ bread > 0.5$ AND $fruit/vegetable\ juice > 0.5$ AND $rolls/buns > 0.5$ AND $bottled\ beer > 0.5$ AND $citrus\ fruit > 0.5$ AND $soda > 0.5$ AND $hamburger\ meat \leq 0.5$ AND $butter \leq 0.5$ AND $meat\ spreads \leq 0.5$ AND $frankfurter \leq 0.5$ AND $specialty\ cheese \leq 0.5$ AND $butter\ milk \leq 0.5$ AND $berries \leq 0.5$
$sausage > 0.5$ AND $other\ vegetables > 0.5$ AND $zwieback > 0.5$ AND $soda > 0.5$ AND $domestic\ eggs \leq 0.5$ AND $hygienearticles \leq 0.5$ AND $spices \leq 0.5$ AND $canned\ fruit \leq 0.5$ AND $root\ vegetables \leq 0.5$ AND $shopping\ bags \leq 0.5$ AND $yogurt \leq 0.5$ AND $pastry \leq 0.5$
$0.003 \leq 0.22 \cdot whole-milk + 0.16 \cdot yogurt \leq 0.382$ AND $0.014 \leq 0.22 \cdot whole-milk + 0.43 \cdot rolls/buns \leq 0.65$ AND $0.693 \leq 2.76 \cdot whole-milk + 2.43 \cdot yogurt + 1.92 \cdot brown-bread \leq 7.12$ AND $0.031 \leq 1.65 \cdot rolls/buns + 1.73 \cdot soda \leq 3.38$

Table 3: Example patterns for the Groceries dataset. Since every item is a binary variable indicating whether it is included in a transaction or not, the patterns can be represented as a logical clause. Green conditions describe inclusion of an item while red conditions describe exclusion of an item. Notice that multivariate conditions can be converted to OR statements, highlighted in brown. Note that for uni-variate conditions, we only include one side of the inequalities as the other side is meaningless in this context due to the binary form of the variables.

Results: Table 3 includes example patterns. This case exemplifies the usage of SEEN for generating logical clauses since every binary variable is an indicator for the inclusion or exclusion of an item in a transaction. We distinguish between 2 types of conditions and exemplify a method to convert them into logical conditions. An unary condition is converted directly to inclusion or exclusion of an item. For example, the condition $meat > 0.5$ implies that $meat$ is included in the pattern. Similarly, the condition $cream\ cheese \leq 0.5$ implies that $cream\ cheese$ is excluded. The multivariate conditions represent the OR operator, since at least one item must be included in order to satisfy the condition. We further present the conversion of 2 chosen patterns from Table 3 into clauses. The first row of the table is converted to the following clause:

$$meat \ \wedge \ \neg cream\ cheese \ \wedge \ \neg curd \ \wedge \ \neg coffee \\ \wedge \ hamburger\ meat \ \wedge \ rolls/buns$$

Interestingly, it can be inferred as a combination of ingredients that make up a hamburger meal, which might describe consumption habits of certain people. An important question that rises is - could traditional itemset mining algorithms [5] find such a pattern? In order to answer this question, we've counted the number of transactions satisfying this clause. Amongst all 14963 transactions in the groceries dataset, there is only 1 such transaction, which means that traditional frequent itemset mining [5] would not find it, despite being an interesting itemset. Indeed, we've observed similar behavior for the other patterns in Table 3.

We continue with the analysis of the pattern in the fourth row of Table 3, converted to the following clause:

$$(whole\ milk \vee yogurt) \ \wedge \ (whole\ milk \vee rolls/buns) \wedge \\ (whole\ milk \vee yogurt \vee brown\ bread) \ \wedge (rolls/buns \vee soda)$$

This example introduces a new variety of itemset mining. While traditional approaches do not support OR conditions between items, we were able to generate such itemsets because we use multivariate functions. Hence, it is very expressive and new powerful insights could be obtained from these results. Moving to the analysis of the impact of the tree height and number of neighbors, a similar behavior as in the driving case study is observed in Figures 5b and 5d.

6 Conclusions

In this paper, we presented SEEN a novel method for mining mathematical expressions from learned representations. Using a soft decision tree, it effectively generates expressive and representative mathematical expressions from the knowledge obtained in the representation learning phase. The experiments conducted on multiple datasets show that SEEN has great expressive power and comprehensibility at the same time. For future work, we consider two possible directions: replacing the soft decision tree with different surrogate models for pattern generation, and discovering new representation learning methodologies from which we can generate representative and interesting patterns.

Acknowledgement

The research leading to these results was supported by the Israel Science Foundation (grant No.191/18). This research was partially supported by the Technion Hiroshi Fujiwara Cyber Security Research Center and the Israel National Cyber Directorate.

References

[1] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data mining and knowledge discovery*.

[2] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Roger Nkambou, Bay Vo, and Vincent S Tseng. *High-utility pattern mining*. Springer, 2019.

[3] Jian Pei and Jiawei Han. Can we push more constraints into frequent pattern mining? In *The sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000.

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8), 2013.

[5] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Bay Vo, Tin Truong Chi, Ji Zhang, and Hoai Bac Le. A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(4), 2017.

[6] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Rage Uday Kiran, Yun Sing Koh, and Rincy Thomas. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1), 2017.

[7] Maor Yankovitch, Ilya Kolchinsky, and Assaf Schuster. Hypersonic: A hybrid parallelization approach for scalable complex event processing. In *Proc. 2020 ACM SIGMOD Int. Conf. on Management of Data*, 2022.

[8] Koral Chapnik, Ilya Kolchinsky, and Assaf Schuster. Darling: Data-aware load shedding in complex event processing systems. *Proc. VLDB Endowment*, 2022.

[9] Ilya Kolchinsky and Assaf Schuster. Real-time multi-pattern detection over event streams. In *Proc. Int. Conf. on Management of Data*, pages 589–606, 2019.

[10] Ilya Kolchinsky and Assaf Schuster. Efficient adaptive detection of complex event patterns. *Proc. VLDB Endowment*, 11(11), 2018.

[11] Ilya Kolchinsky and Assaf Schuster. Join query optimization techniques for complex event processing applications. *Proc. VLDB Endowment*, 11(11), 2018.

[12] Ilya Kolchinsky, Izchak Sharfman, and Assaf Schuster. Lazy evaluation methods for detecting complex events. In *Proc. 9th ACM Int. Conf. on Distributed Event-Based Systems*, pages 34–45, 2015.

[13] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Zhaoyu Wang, Li Mian, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *arXiv preprint arXiv:2006.08218*, 1(2), 2020.

[14] J Emmanuel Johnson, Sairam Sundaresan, Tansu Daylan, Lisseth Gavilan, Daniel K Giles, Stela Ishitani Silva, Anna Jungbluth, Brett Morris, et al. Rotnet: Fast and scalable estimation of stellar rotation periods using convolutional neural networks. *arXiv preprint arXiv:2012.01985*, 2020.

[15] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[16] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proc. 20th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*.

[17] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*.

[18] Alberto Suárez and James F Lutsko. Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12), 1999.

[19] Leonard A Breslow and David W Aha. Simplifying decision trees: A survey. *Knowledge engineering review*, 12(1), 1997.

[20] Ozan Irsoy, Olcay Taner Yıldız, and Ethem Alpaydın. Soft decision trees. In *Proc. 21st Int. Conf. on Pattern Recognition (ICPR2012)*. IEEE, 2012.

[21] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

[22] Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *Int. Conf. on Machine Learning*. PMLR, 2019.

[23] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2), June 1993.

[24] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM SIGMOD Int. Conf. on Management of Data*, SIGMOD '00, New York, NY, USA, 2000. Association for Computing Machinery.

[25] László Gadár and János Abonyi. Frequent pattern mining in multidimensional organizational networks. *Scienti c reports*, 9(1), 2019.

[26] Qingsong Wen, Zhe Zhang, Yan Li, and Liang Sun. Fast robuststl: Efficient and robust seasonal-trend decomposition for time series with complex patterns. In *Proc. 26th ACM SIGKDD Int. Conf. on Knowledge Discovery & Data Mining*.

[27] Xing Wang, Jessica Lin, Pavel Senin, Tim Oates, Sunil Gandhi, Arnold P Boedihardjo, Crystal Chen, and Susan Frankenstein. Rpm: Representative pattern mining for efficient time series classification. In *EDBT*.

[28] Sanjoy Dasgupta, Nave Frost, Michal Moshkovitz, and Cyrus Rashtchian. Explainable k-means and k-medians clustering. *arXiv preprint arXiv:2002.12538*.

[29] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*.

[30] Haydemar Núñez, Cecilio Angulo, and Andreu Català. Rule extraction from support vector machines. In *Esann*, 2002.

[31] Harald Schafer, Eder Santana, Andrew Haden, and Riccardo Biasini. A commute in data: The comma2k19 dataset, 2018.

[32] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[33] Heeral Dedhia. Groceries dataset, Sep 2020.

[34] Alexander Artikis, Chris Baber, Pedro Bizarro, Carlos Canudas-de Wit, Opher Etzion, Fabiana Fournier, Paul Goulart, Andrew Howes, John Lygeros, Georgios Paliouras, et al. Scalable proactive event-driven decision making. *IEEE Technology and Society Magazine*, 33(3):35–41, 2014.