Queryable and Interpretable PU Learning through Probabilistic Circuits

Sieben Bocklandt[1,2] (✉), Vincent Derkinderen[1,2], Koen Vanderstraeten[3], Wouter Pijpops[3], Kurt Jaspers[3], Luc De Raedt[1,2,4], and Wannes Meert[1,2]

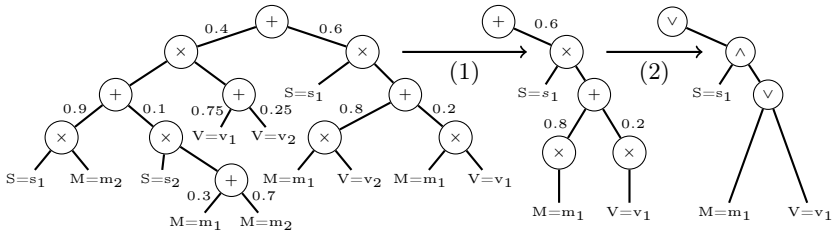
[1] Department of Computer Science, KU Leuven, Belgium
{sieben.bocklandt}@kuleuven.be
[2] Leuven.AI - KU Leuven Institute for AI, Belgium
[3] Tunify
[4] Center for Applied Autonomous Systems, Örebro University, Sweden

Abstract. We introduce a novel concept learning scenario that involves only positive and unlabeled (PU) data and focuses on interpretable models. Our scenario is motivated by a real-world application learning concepts for music playlists (e.g., 'relaxing music'). These concepts must be understood by humans and used as database queries. We demonstrate that probabilistic circuits offer a compelling solution for PU learning as they can effectively learn to represent joint probability distributions without the need for negative examples. However, achieving interpretability and seamless conversion into database queries presents additional challenges. To address these, we propose a novel approach that transforms a learned probabilistic circuit into a logic-based discriminative model. Notably, this is the first study to investigate probabilistic circuits in a PU learning framework, contributing two key innovations: (1) a new description length metric called aggregated entropy as a measure for interpretability; and (2) PUTPUT, an algorithm designed to prune low-probability regions from the circuit before converting it into a logic-based model, optimizing for both F_1-score and aggregated entropy.

1 Introduction

Our work is motivated by an application in the music streaming industry, in which music playlists play a crucial role. The automated curation of playlists is an active area of research [1, 4, 5, 24, 28]. A particularly effective approach is to represent a playlist as a concept (e.g., 'relaxing music') rather than as a fixed collection of songs. When a playlist is represented as a discriminative model, it can be automatically populated, allowing for the inclusion of newly released songs that fit the concept.

Learning a concept in this context involves meeting the following criteria:
(i) it is a PU learning task, i.e., the data is exclusively positive and unlabeled [3];
(ii) the learned model must be convertible into a database query; and
(iii) the model must be interpretable, allowing a music expert to inspect and validate it. The importance of interpretability is particularly crucial in a business



(a) **probabilistic circuit** (generative model)

(b) **pruned circuit**

(c) **matching logical formula** (discriminative model)

Fig. 1. Overview of our approach (in the context of a music playlist). (1) A probabilistic circuit over variables *Style*(S), *Mood*(M) and *Version*(V) is pruned to only contain the high probability regions. (2) This smaller circuit is transformed into a logic formula, acting as a discriminative model that is easier to inspect and verify by a domain expert, and can be turned into a database query.

setting. For instance, Tunify[5] provides a music streaming service specifically designed for businesses, in which they must ensure the quality of each playlist. In this context, even a single inappropriate song can disrupt the atmosphere in environments like wellness centres or funeral homes.

To meet all these criteria, we focus on models that are well-suited to a PU setting and can be converted into a logic formula. Decision trees and rule learning algorithms fulfil these requirements, but require that we first augment the dataset with negative examples while taking into account the PU-learning setting. For instance, the Rocchio classification approach identifies reliable negatives by selecting examples close to a prototype learned from the unlabeled data [3].

We focus on probabilistic circuits (see Figure 1a) as a competitive alternative, as they do not require negative examples and are therefore more directly applicable to and appropriate for the PU learning. These circuits [6] form a unified framework for tractable probabilistic models, representing a joint probability distribution while being capable of tractably handling various tasks. However, these circuits are more challenging to inspect and convert into a database query. To address this, we propose a new approach that extracts a logical formula (the discriminative model) from a learned probabilistic circuit, **leading to the following contributions.**

1. To the best of our knowledge, we are the first to study probabilistic circuits in an interpretable PU setting.
2. We propose the aggregated entropy as a new description length that measures the ease with which a domain expert can inspect the extracted formula.
3. More importantly, we introduce PUTPUT (Probabilistic circuit Understanding Through Pruning Underlying logical Theories). This is a new method that prunes the low probability regions from the circuit (see Figure 1b), as

[5] www.tunify.com

these are less likely to be part of the intended concept, whilst also considering the impact on interpretability (the aggregated entropy). After pruning these regions, we can easily extract a matching logical formula from the circuit (see Figure 1c) that can be inspected and can function as a database query.

Our evaluation demonstrates the effectiveness of our approach on a real-world use case of music playlist generation. Furthermore, a user study shows that aggregated entropy is better fit to measure human interpretability than the standard description length and evaluation on open-source datasets confirms that the method is more generally applicable beyond this use case.

The remainder of the paper is structured as follows. First, we provide the necessary background information and the problem statement in respectively Section 2 and 3. Then, in Section 4 and 5, we introduce our new description length called aggregated entropy, and our method, called PUTPUT. The empirical evaluation of these is presented afterwards, in Section 6. Finally, we discuss related work in Section 7 and conclude in Section 8.

2 Background

We first provide a brief primer on logic formulas, probabilistic circuits and PU learning.

2.1 Logic Formulas

A *literal* is a Boolean variable v or its negation $\neg v$. A *propositional logic formula* ψ is inductively defined as a literal, the negation of logic formula $\neg\psi_1$, the conjunction (read 'and') of two logic formulas $\psi_1 \wedge \psi_2$, or a disjunction (read 'or') $\psi_1 \vee \psi_2$, with the expected semantics. A *clause* $\mathcal{C}$ is a literal or disjunction of literals, such as $v_1 \vee \neg v_2$. A formula ψ is said to be in *conjunctive normal form* (CNF) iff it is a conjunction of clauses, such as $(v_1 \vee \neg v_2) \wedge (\neg v_1)$. A formula ψ is said to be in *disjunctive normal form* (DNF) iff it is a disjunction of conjunctions of literals, such as $(v_1 \wedge \neg v_2) \vee (\neg v_1)$.

We support categorical variables by admitting Boolean variables that represent equalities. That is, a Boolean variable could represent $style{=}jazz$, while another variable represents $style{=}rock$. We assume an implicit theory that forbids both variables to be true at the same time. For convenience, we may write $style{=}jazz$ rather than $v_{style=jazz}$. An *example* in our context is a value assignment to each categorical variable.

The *dual graph* $G_d(\psi)$ of a CNF formula ψ is a graph that connects clauses iff they share the same variable [29]. In our context we slightly change this definition to reason over categorical and binary variables. More formally, two clauses $\mathcal{C}$ and $\mathcal{C}'$ are joined by an edge iff there is a categorical or binary variable that is present in both clauses.

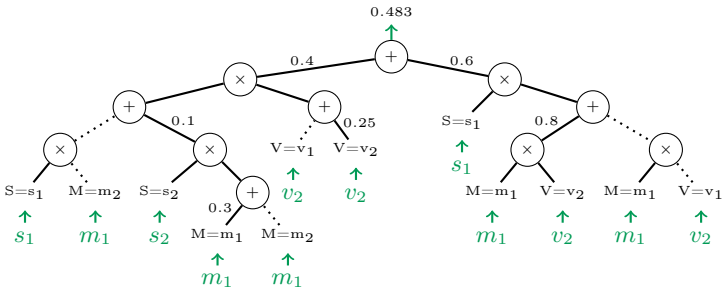


Fig. 2. Generation of a probability with given input $\{m_1, v_2\}$. We marginalise over S, as no value is assigned to S. Dashed lines indicate edges whose weight is irrelevant for the given input. After performing addition and multiplying with the relevant weights, the final result, $P(Mood = m_1, Version = v_2)$, is given at the root node.

2.2 Probabilistic Circuits

A probabilistic circuit (PC) $\mathcal{M} := (\mathcal{G}, \theta)$ is a probabilistic model, representing a joint probability distribution $\mathrm{P}(\mathbf{X})$ over random variables $\mathbf{X}$ through a directed acyclic graph (DAG) $\mathcal{G}$ parameterised by θ [6]. Each node in the DAG defines a computational unit, which is one of three types — an input, sum, or product node. Every leaf in $\mathcal{G}$ constitutes an input node, while every inner node is either a sum or a product node. An example of a probabilistic circuit is given in Figure 1a.

In our setting, each input node of a PC represents a distribution over a categorical random variable X. Furthermore, we assume without loss of expressivity that the distribution has all probability mass over a single value $x \in X$. In other words, an input node labeled $X = x$ in Figure 1a outputs the probability (0 or 1) that the input is equal to x. Each product node represents a factorisation of incoming distributions over different random variables, and each sum node represents a mixture, i.e., a weighted sum over the distributions leading into it. The weights of the mixture are indicated on the edges in Figure 1a.

Figure 2 shows an evaluation of the PC to compute $P(Mood = m_1, Version = v_2)$, marginalising over *Style*. First, we set the input of the PC. For example, each input node associated to M has input m_1, leading them to output 1 for $M{=}m_1$ and 0 for $M{=}m_2$. We marginalise over *Style*, denoted as S, thus each input node $S{=}s$ receives input s and outputs 1. These outputs propagate through the PC, performing the sum- and product operations, resulting in the output probability at the root node.

A probabilistic circuit $\mathcal{M}$ can easily be converted into a logic formula ψ that captures the nonzero probability input instances of the circuit. Assuming only nonzero weights on the edges, ψ can be extracted from $\mathcal{M}$ by replacing every product node with $\wedge$, every sum node with $\vee$, and removing the weights of every edge. Additionally, the input nodes are converted into literals such as $S{=}s_1$. Figure 3 shows the logic formula ψ extracted from Figure 1a.

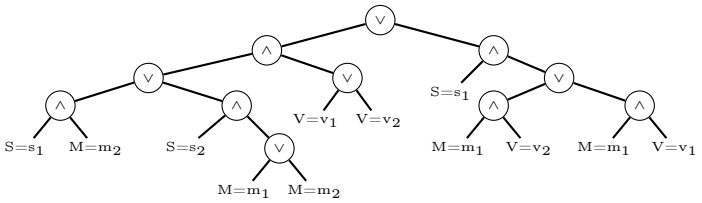


Fig. 3. Formula derived from the circuit in Figure 1a.

2.3 Learning from Positive and Unlabeled Data

PU learning methods address the task of learning from positive and unlabeled data [3]. Neural networks, support vector machines, bagging approaches and density estimators can all be used in this setting [17, 21, 37, 38]. These approaches however do not result in a classifier that is easily convertable into a database query and inspectable by a domain expert. As an alternative approach, one can first enhance the dataset with reliable negative examples before learning a decision tree or a ruleset – both of which are easier to convert into a query and inspect. For instance, Rocchio classification identifies reliable negative examples as examples that are close to a prototype learned on the unlabeled data [3]. A similar approach, that was also used in the context of music playlist generation, considers the likelihood to identify reliable negative examples [15].

3 Problem Statement

The motivating use case of our work is a problem occurring in the workflow of music streaming provider *Tunify*. They have a database of annotated music where each song is represented by a fixed set of categorical features. These can be objective (*BPM*, *Year*, *Lyricist*, ...) or subjective (*Mood*, *Feel*, ...). As one of their services, they provide a predefined selection of playlists. These playlists are represented as logical formulas ψ that can be used as queries on a database to obtain the songs currently matching ψ.

However, *Tunify* must also be able to assure customers that all playlists are safe, e.g., to avoid a black metal song appearing in a playlist intended as happy songs for children, or for a funeral home. Therefore, to enable easier inspection by a music expert, we wish to extract an interpretable formula ψ that acts as a discriminative classifier covering the high probability regions of the learned probabilistic circuit. The focus on high probability regions allows ψ to cover only the most relevant examples.

Given: a probabilistic circuit $\mathcal{M}$ as described in Section 2, a set of examples $\mathcal{E}$, a description length DL and a nonzero probability threshold t.

Objective: $\mathcal{E}_{HPR} \subseteq \mathcal{E}$ are examples in the high probability regions of $\mathcal{M}$, i.e., the examples $e \in \mathcal{E}$ for which $P_{\mathcal{M}}(e) \geq t$. Using $\mathcal{E}_\psi$ to indicate all examples in $\mathcal{E}$ that are covered by ψ, the goal is to find formula ψ such that:

1. $\mathcal{E}_\psi$ is as close to $\mathcal{E}_{HPR}$ as possible [6]: $\arg\max_\psi F_1(\mathcal{E}_\psi, \mathcal{E}_{HPR})$ (1)
2. whilst minimising the description length of ψ: $\arg\min_\psi DL(\psi)$ (2)

4 A New Description Length

The approach that we will propose captures the high probability regions of a probabilistic circuit as a logical formula that is easier to inspect and understand by a domain expert. The standard description length, i.e., the number of literals in the formula, is not fine-grained enough to fully capture interpretability. This description length fails to account for subtle variations in how users perceive complexity, leading to weaker alignment with interpretability trends. To address this limitation, we introduce a new description length measure called *aggregated entropy*. Before describing it in detail, we first provide the motivation for its introduction.

4.1 Motivation of Aggregated Entropy

First, consider that nested logic formulas consisting of multiple levels of $\wedge$ and $\vee$ are more complex than flat forms such as CNF and DNF. Second, consider that DNFs are more complex to inspect once categorical variables are involved. This is due to what Ryszard Michalski, one of the founders of the field of machine learning, called an internal disjunction: a disjunction over a categorical variable to indicate its possible values, for example '$style{=}jazz \ \vee \ style{=}rock$'. He argued for their interpretability from a cognitive perspective in a concept learning setting [20]. We illustrate this using the following example from our application.

$$\begin{aligned}&(style = jazz \wedge feel = happy) \vee (style = jazz \wedge feel = exciting) \vee \\ &(style = rock \wedge feel = happy) \vee (style = rock \wedge feel = exciting)\end{aligned} \tag{3}$$

In this case, a CNF representation is preferred.

$$(style = jazz \vee style = rock) \wedge (feel = happy \vee feel = exciting) \tag{4}$$

An additional advantage of CNF formulas is that they can easily be extended to remove undesired examples e (e.g., songs) as examples are conjunctions of attributes: $\psi \wedge \neg e$ which is equal to $\psi \wedge (\neg v_1 \vee \cdots \vee \neg v_n)$.

Description lengths provide a quantitative measure of complexity for the information content within data or models. DUCE [22] and BoolXAI [27], for instance, use the literal count as a description length for formulas in DNF and CNF respectively. Similarly, description length finds an application in information theory, as illustrated by the Huffman encoding which minimises the number

[6] Note that F_1-score is preferred over accuracy here, as there can be significantly more true negatives than true positives.

of bits required to describe a sentence [16]. These description lengths capture the length of a logical formula.

Naturally, a longer formula is less desirable from an interpretability standpoint. Still, we argue these description lengths are insufficient in the context of this work because they neither consider the complexity that arises when variables are present in multiple clauses, nor do they consider the categorical variables. We therefore propose a new description length that we call the aggregated entropy of a CNF formula.

4.2 Aggregated Entropy

This new description length is based on information theory and keeps two principles in mind:

- A CNF formula is easier to understand when a variable is present in only a few clauses.
- A CNF formula is easier to understand when a categorical variable allows either very few or many values. As an example, consider a formula that expresses that a music style is only allowed to be metal, or one that expresses everything except metal.

Aggregated entropy approximates this by quantifying the number of bits needed to represent a clause and its directly linked clauses, as a proxy for how much a user needs to memorise when reading the model.

Definition 1 (Entropy of a variable within a clause). *The entropy of a categorical variable X within a clause $\mathcal{C}$ and with $\phi(\alpha) = -\alpha\ log_2(\alpha)$ is defined as*

$$E_{var}(\mathcal{C}, X) = \phi(\frac{|\mathcal{C}(X)|}{|X|}) + \phi(\frac{|X| - |\mathcal{C}(X)|}{|X|}), \tag{5}$$

with $\mathcal{C}(X)$ the set of Boolean variables in $\mathcal{C}$ that are associated with X, and $|X|$ the total number of possible values for X. In other words, $|\mathcal{C}(X)|/|X|$ is the fraction of possible values for X that are mentioned within clause $\mathcal{C}$.

Definition 2 (Aggregated entropy of a clause). *The aggregated entropy $DL_{cl}(\mathcal{C})$ of a clause $\mathcal{C}$ is a description length that aggregates the entropy of its variables,*

$$DL_{cl}(\mathcal{C}) = \sum_{X \in \mathcal{C}} E_{var}(\mathcal{C}, X), \tag{6}$$

where we use $X \in \mathcal{C}$ to consider the categorical variables that are present in clause $\mathcal{C}$.

The ease with which a CNF formula ψ is understood decreases when variables are present in multiple clauses. Therefore, while considering the aggregated entropy for each clause $\mathcal{C}$ within ψ, we also consider the aggregate of its neighbouring clauses, i.e., the clauses $\mathcal{C}'$ with whom $\mathcal{C}$ shares variables.

Definition 3 (Aggregated entropy of a CNF). *The aggregated entropy $DL(\psi)$ of a CNF formula ψ is*

$$DL(\psi) = \sum_{\mathcal{C} \in \psi} \left[DL_{cl}(\mathcal{C}) + \sum_{\mathcal{C}' \in \psi | e(\mathcal{C}, \mathcal{C}') \in G_d(\psi)} DL_{cl}(\mathcal{C}') \right], \tag{7}$$

with $e(\mathcal{C}, \mathcal{C}') \in G_d(\psi)$ denoting the clause neighbour relationships through the dual graph defined in Section 2.

Example of aggregated entropy Given formula ψ over categorical variables $\{A, B, X\}$ with $|A| = 5$, $|B| = 6$, and $|X| = 7$.

$$\psi = \underbrace{(a_1 \vee a_2 \vee a_3)}_{\mathcal{C}_1} \wedge \underbrace{(a_1 \vee b_1 \vee b_2)}_{\mathcal{C}_2} \wedge \underbrace{(x_1 \vee x_2 \vee x_3)}_{\mathcal{C}_3}$$

We compute $DL(\psi)$ using the following clause entropies:

$$DL_{cl}(\mathcal{C}_1) = \phi(\frac{3}{5}) + \phi(\frac{2}{5}),\ DL_{cl}(\mathcal{C}_2) = \phi(\frac{1}{5}) + \phi(\frac{4}{5}) + \phi(\frac{2}{6}) + \phi(\frac{4}{6})$$
$$DL_{cl}(\mathcal{C}_3) = \phi(\frac{3}{7}) + \phi(\frac{4}{7})$$

with $\phi(\alpha) = -\alpha log_2(\alpha)$. Clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ both mention categorical variable A, resulting in an edge $e(\mathcal{C}_1, \mathcal{C}_2)$ in the dual graph of ψ. This results in
$DL(\psi) = \underbrace{DL_{cl}(\mathcal{C}_1) + DL_{cl}(\mathcal{C}_2) + DL_{cl}(\mathcal{C}_3)}_{\text{clause entropies}} + \underbrace{DL_{cl}(\mathcal{C}_2)}_{e(\mathcal{C}_1,\mathcal{C}_2)} + \underbrace{DL_{cl}(\mathcal{C}_1)}_{e(\mathcal{C}_2,\mathcal{C}_1)} \approx 6.21$

In the rest of this paper we minimise $DL(\psi)$ to obtain more preferred CNF formulas ψ.

4.3 User Study

To evaluate whether our newly proposed description length better captures human interpretability compared to using the number of literals, we conducted a user study. Participants (N = 46) were asked to answer 12 questions. In each question, they were given i) a playlist description in the form of a CNF formula, and ii) a set of songs, each represented as a list of categorical attributes, after which they were asked to select all songs that were covered by the playlist.

We posit that descriptions that are less interpretable, lead to a longer response time. Furthermore, a description length should align with this, similarly assigning a higher length. Therefore, we compare the aggregated entropy and the classic description length that is based on the number of literals, by their ability to align with the trend in response time.

The Spearman correlation between the response time is 0.63 for the aggregated entropy, and 0.55 for the number of literals, indicating the presence of correlation for both. To determine whether the difference in strengths is statistically significant, we apply Steiger's Z-test for dependent correlations, obtaining a

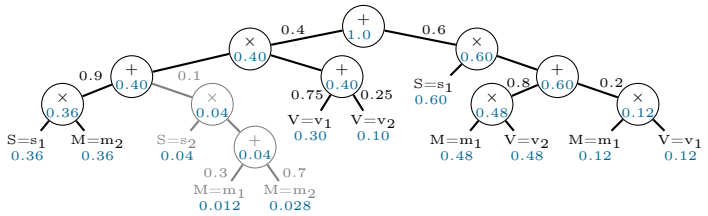


Fig. 4. The result after pruning the probabilistic circuit from Figure 1a, with the generative significance method set to eliminate the five nodes with the lowest top down probabilities (shown in blue).

p-value of 0.03. Since $p < 0.05$, we conclude that the correlation with aggregated entropy is significantly stronger, and thus, that it is a more appropriate measure for human interpretability. Our results further show in a pairwise comparison that the aggregated entropy aligned with user response trends in 77% of cases, compared to 61% for the number of literals metric. The full questionnaire and additional details about the user study can be found in Appendix D.

5 Method

The input probabilistic circuit $\mathcal{M}$ is a generative model that can be transformed into a discriminative one: given an example e, we can compute $P_{\mathcal{M}}(e) \geq t$ (with t the given threshold). However, because the discriminative model must function as a database query and be inspectable by a domain expert, we instead consider extracting a logical formula ψ from the circuit using the approach described in Section 2.2.

Importantly, the resulting formula would be too general as it covers any nonzero probability instance of the circuit. To solve this problem, we propose to first prune the probabilistic circuit $\mathcal{M}$ in a way that only the high probability regions remain. Afterwards, we can extract a logical formula ψ from the pruned circuit and convert it to a CNF. This formula then acts as a discriminative classifier that indicates whether a given example belongs to the high probability region of the input probabilistic circuit $\mathcal{M}$.

We propose PUTPUT, a new two-step approach that prunes a probabilistic circuit while considering the F_1-score and aggregated entropy. The first step eliminates circuit edges that originate from sum nodes, using existing pruning functions. This results in a circuit only covering the high probability regions. A second step eliminates input nodes to further decrease the aggregated entropy, while maintaining the F_1-score of the first step as a lower bound.

5.1 Step 1: Pruning Sum Nodes

Pruning functions. Dang, Liu, and Van den Broeck (2022) proposes four pruning functions for a probabilistic circuit. The first function randomly eliminates

Algorithm 1 Step 2: Pruning input nodes in PC

Input: Pruned PC $\mathcal{M}'$, high-probability examples $\mathcal{E}_{HPR}$, examples $\mathcal{E}$
Output: $\mathcal{M}'$ with pruned input nodes
1: $lower\_bound = F_1(\mathcal{E}_{\mathcal{M}'}, \mathcal{E}_{HPR})$
2: **repeat**
3: **for** each input node n in $\mathcal{M}'$ **do**
4: $\mathcal{M}'' \leftarrow \mathcal{M}'/n$ ◁ Prunes node n
5: **if** $F_1(\mathcal{E}_{\mathcal{M}''}, \mathcal{E}_{HPR}) \geq lower\_bound$ **then**
6: $\mathcal{M}' \leftarrow \mathcal{M}''$
7: **until** $\mathcal{M}'$ has not changed
8: **return** $\mathcal{M}'$

sum node inputs, while the second approach eliminates them based on their corresponding mixture weight. Both of these functions were identified as less performant so we do not consider them. The third function is based on generative significance, pruning those sum node inputs that contribute the least to the circuit output. In Figure 4 we show its application on the circuit of Figure 1a, annotating each node by its top down probability, i.e., the probability that the node will be visited when unconditionally drawing samples from the circuit. This function is parameterised by the number of edges it must eliminate. The fourth approach is based on circuit flows, which works similar to the third approach but it first adjusts the sum node mixture weights by conditioning on a given dataset. In this way, it considers how many samples from the dataset flow through each node. This pruning function is parameterised by the number of edges it must eliminate and the dataset on which to condition.

Applying pruning functions. PUTPUT first identifies the preferred values of the parameters (i.e., the number of edges to eliminate) of the pruning function that lead to the highest F_1-score (Equation 1). This can be achieved exhaustively or by using a search function such as golden section search [25]. We used the latter in our evaluation to conclude that pruning based on circuit flows is the preferred pruning function. The result of step 1 is a pruned probabilistic circuit such that the F_1-score is maximised (see Figure 4).

5.2 Step 2: Pruning Input Nodes

The first step already decreases the aggregated entropy due to its correlation with circuit size. The second step decreases this even further by considering for each input node whether it is beneficial to prune them. While the first step prunes some sum node inputs, the second step prunes them further and in addition also prunes input nodes that lead into product nodes.

The F_1-score resulting from the first step is used as a lower bound in this second step. Pruning children of a product node may influence whether it is beneficial to prune a node that was previously considered. PUTPUT therefore employs an iterative procedure that reconsiders all input nodes until no more

changes are made. The pseudocode for this step is shown in Algorithm 1. The circuit resulting from step 1 is denoted as $\mathcal{M}'$, while $\mathcal{E}_{\mathcal{M}'}$ are the examples $e \in \mathcal{E}$ for which $P'_{\mathcal{M}}(e) > 0$, as these are the examples covered by the logical formula derived from $\mathcal{M}'$ (see Section 2.2). Note that probability threshold t is implicitly present in $\mathcal{E}_{HPR}$. If we apply this second step on Figure 4, we obtain Figure 5.

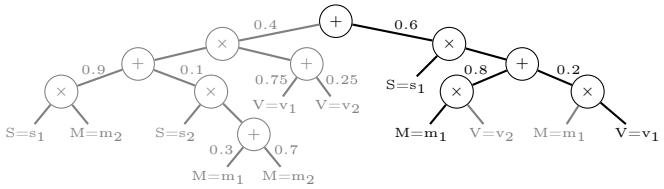


Fig. 5. The circuit after applying step 2 of PUTPUT, with $\mathcal{E}_{HPR} = \{(s_1, m_1, v_1), (s_1, m_1, v_2), (s_1, m_2, v_1)\}$.

6 Evaluation

The proposed PUTPUT method is empirically evaluated to answer the following research questions:

1. Which pruning function results in the highest F_1-score (step 1)?
2. Does pruning of the input nodes improve the aggregated entropy (step 2)?
3. How does PUTPUT perform more generally, including on the music playlist generation task?

6.1 Setup

Given a dataset of positive and unlabeled examples $\mathcal{E}$, we first learn a probabilistic circuit $\mathcal{M}$ using the Hidden Chow Liu Tree method [18] available in the JUICE package [10]. Next, we find the probability threshold t that identifies the high probability regions. A higher threshold t leads to a smaller, more precise region. However, if the threshold t is too high, the resulting formula will overfit, leading to lower recall and thus reduced F_1-score. The user's choice of t, which dictates the trade-off, is therefore important. As this depends on a specific use case, we instead determine an appropriate value for t through a more generalized approach, using the elbow method on the generated posterior probabilities, which is inspired by work on finding the reject threshold in pattern recognition [7].

The elbow method selects a probability threshold t by first ordering all examples based on their probability, in descending order. More formally, we use $\mathcal{E}$ to denote a set of examples, $\mathcal{M}$ to denote a probabilistic circuit, and $p(e)$ to denote the probability of $e \in \mathcal{E}$ according to $\mathcal{M}$. Let $\mathcal{L} = [e_1, ..., e_n]$ be the examples in

$\mathcal{E}$, ordered according to probability $p(e)$, such that e_1 has the highest probability and e_n the lowest. The elbow method then finds threshold t by searching for a sudden sharper decrease in probability. I.e., find example e_i such that

$$i = \underset{x=1..|\mathcal{E}|}{\arg\min}(x \mid \frac{p(e_{x+1}) - p(e_x)}{p(e_x) - p(e_{x-1})} \geq 0.3), \tag{8}$$

after which the threshold is defined as $t = p(e_i)$. The value of 0.3 in the elbow method is a user-specified parameter. We decided this value by analysing the results of the music use case in collaboration with a music expert of *Tunify*. Since there is no expert for the open-source datasets, we reuse the same threshold t throughout the experiments.

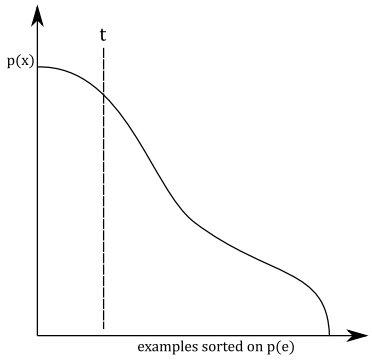


Fig. 6. The elbow method selects threshold t, identifying high probability regions.

Using this threshold t, we determine the high probability region examples $\mathcal{E}_{HPR} \subseteq \mathcal{E}$. Afterwards, we apply PUTPUT to prune $\mathcal{M}$, leading to the discriminative formula ψ.

6.2 Datasets

- *Tunify* provided real-world data, 360 000 songs, annotated with 14 categorical features having 7 to 120 possible values, and a set of intended playlist concepts. From this private data 15 classes where constructed, each representing a different playlist concept that has to be learned. We consider two types of concepts, based on the playlists used by customers from *Tunify*.
  - *Single playlist concept*: 5 known product playlist concepts with their respective songs, e.g., `Rock`.
  - *Disjunctive playlist concepts*: 10 combinations of two known concepts that have a disjunctive form, e.g., `Rock or Easy Lounge`.
- the *FMA* music dataset [12] used for genre classification (8 classes, 9217 examples), which is the open source dataset that is most similar to the *Tunify* dataset.

- black and white images of
  - MNIST (10 classes, 2500 examples) [13],
  - fashionMNIST (10 classes, 2500 examples) [36],
  - EMNIST letters (26 classes, 2600 examples) [8].
- the *mushrooms* (2 classes, 8124 examples) [31] and *splice* (3 classes, 3190 examples) [32] datasets of the UCI machine learning repository.

Setup. A class in a dataset determines the positive labels. We consider a PU learning setting wherein most of the examples are unlabeled. More specifically, for each class in each dataset, we create 10 subsets of the data and randomly label only 5% of the positively labeled examples, resulting in 740 PUTPUT datasets. We evaluate on the full dataset.

Example. We provide a small example to motivate our goal of extracting a discriminative classifier, and to illustrate the general applicability beyond music playlist generation. We learned a probabilistic circuit on black and white images of MNIST data containing 13 positively labeled examples representing the digit 0. We then applied PUTPUT to obtain the following logical formula ψ with (W=White, B=Black).

$$p_{12,22}{=}W \wedge p_{14,15}{=}B \wedge p_{14,16}{=}B \wedge (p_{8,15}{=}W \vee\ p_{8,17}{=}W) \wedge (p_{15,9}{=}W \vee p_{13,12}{=}B)$$

Apparently, the high probability region of the learned probabilistic circuit only considers seven of the 784 pixels to predict whether an MNIST digit depicts a 0. Figure 7 shows two MNIST digits that match ψ and are part of the high probability regions of the probabilistic circuit. The latter can be verified by evaluating the probabilistic circuit for the given image. In addition to being more interpretable, the domain expert can also use description ψ as a starting point to further refine their intended concept.

6.3 Experiments

Experiment 1: comparing pruning methods. To address research question 1, we evaluate the first step of PUTPUT with the two previously described pruning functions on the open-source benchmarks. Table 1 indicates that pruning by circuit flows results in the best average F_1-score and circuit size, the latter of which likely leads to a decreased aggregated entropy. In the following experiments, we therefore use pruning by circuit flows.

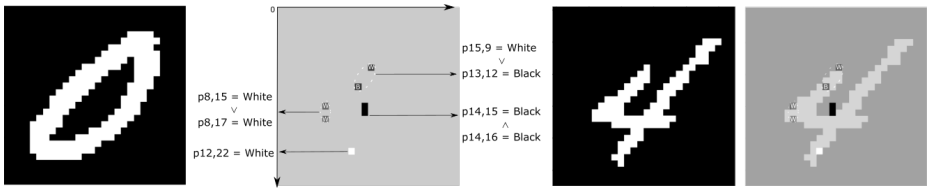


Fig. 7. Two MNIST images matching the logical formula ψ extracted by PUTPUT from a learned probabilistic circuit.

Table 1. Results of experiment 1 after the first step of PUTPUT, showing the average F_1-score and circuit size (and standard deviations) in relation to $\mathcal{E}_{HPR}$. The abbreviation gen. sign. refers to the generative significance method.

Function	**F_1-score**	**Circuit size** (# nodes)
no pruning		3086 ± 1163
circuit flows	0.313 ± 0.23	1812 ± 812
gen. sign.	0.285 ± 0.24	2281 ± 904

Table 2. Results of experiment 2, showing average and standard deviation for various metrics, in relation to examples $\mathcal{E}_{HPR}$, after PUTPUT step 1 and 2.

Step	**F_1-score**	**Aggregated entropy**	**Precision**	**Recall**	**PC size** (# nodes)	**LC size** (# nodes)
1	0.313 ± 0.23	10168 ± 45798	0.341 ± 0.25	0.363 ± 0.27	1812 ± 812	1168 ± 1242
2	0.362 ± 0.23	287 ± 454	0.341 ± 0.26	0.381 ± 0.26	523 ± 294	92 ± 73

Experiment 2: effect of PUTPUT's step 2. The second step of PUTPUT prunes the input nodes to further lower the aggregated entropy. Table 2 shows the results of PUTPUT applied on the open-source benchmarks. We conclude for the second research question, that the second step significantly decreased the aggregated entropy. Furthermore, it also increased the F_1-score and recall.

Experiment 3: evaluation of PUTPUT. For all datasets, we evaluate using the ground truth labels rather than $\mathcal{E}_{HPR}$. To the best of our knowledge, the only other work that operates in a PU learning scenario and produces an interpretable model that is convertible into a database query, is the work by Goyal et al. [15]. They learn a decision tree (DT) by first enhancing the dataset with negative examples, via either the Rocchio (r) or Likelihood (l) method [3]. In addition to replicating their work, we also extend their idea by considering inductive logic programming (ILP): instead of learning a decision tree, we learn logical rules using RIPPER [9]. Because those methods are not easily convertable into CNF, but are easily convertable into DNF, we apply them on the reliable negatives as the positive class, converting the resulting DNF into CNF by negation. Afterwards, we compare the CNF to the one produced by our method, PUTPUT. To summarize, we evaluate PUTPUT by comparing to both DT and ILP approaches, using either the Rocchio (r) or Likelihood (l) method.

Figure 8 shows the critical difference diagrams of the experiment. A more detailed visualization of the results is presented in Appendix B and C. The most important metric is the F_1-score, as the best aggregated entropy can be trivially achieved with a formula ψ that is always true. This highlights that PUTPUT outperforms other approaches on the *Tunify* data. They failed to learn a meaningful theory, as shown by the mean F_1-score: 0.73 for PUTPUT versus a max mean of 0.125 for l+ilp. On open-source benchmarks, PUTPUT matches r+DT in F_1-score while outperforming in aggregated entropy. These results show that PUTPUT is broadly applicable beyond the *Tunify* use case.

7 Related Work

Probabilistic circuits. Dang et al. proposed several pruning functions for a probabilistic circuit, functions that we utilise within PUTPUT [11]. They used these functions while devising a prune+grow approach to learn more meaningful probabilistic circuits: insignificant parts are pruned before again growing the remaining part. An earlier application of probabilistic circuits in the field of explainable AI is by Wang et al., who used them to find explanations that have a high probability of being correct [35]. Verreet et al. have used probabilistic circuits to explicitly model PU assumptions, improving the identification of reliable negatives [33]. In contrast to our work, the PC structure was not learned on data, nor was it used to learn the final model.

Explainability in AI can be tackled in different ways. (1) By limiting to interpretable models, possibly sacrificing accuracy [23]; (2) By transforming a learned model into a more interpretable one. For example by compiling a Bayesian network classifier into a logical classifier [30]; (3) By transforming part of a model into an interpretable version. This is the approach followed by LIME [26] and SHAP [19] where linear models are generated around a point of interest; (4) By querying the model to identify edge cases or adversarial examples [14]. In this work we focused on the second strategy, such that the resulting model can easily be transformed into a database query, and inspected by a domain expert.

Pattern mining. Finding a description of a given set of examples is a problem setting that occurs in the field of data mining. KRIMP [34] is a pattern mining algorithm that uses the minimum description length principle to find a code table that compresses the data. Unfortunately, converting this code table into a comprehensive logical formula is not trivial. Another example of the use of pattern mining to find descriptions, is constraint-based querying to explore Bayesian networks [2]. The patterns mined in this work are used to answer explorative queries explaining the Bayesian network representation.

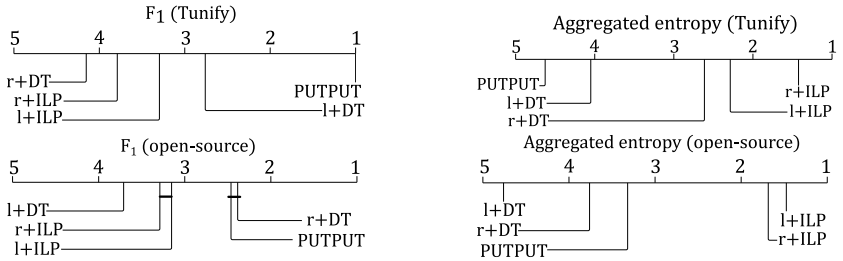


Fig. 8. The critical decision diagrams of experiment 3. Results closer to 1 are better.

8 Conclusions

Motivated by a real-world music playlist generation task, we made three key contributions to queryable, interpretable PU learning. First, we are the first to apply probabilistic circuits for learning interpretable models from positive and unlabeled data. Second, we introduced aggregated entropy, a novel description length that measures the ease with which domain experts can inspect and understand the extracted logical formulas. Our primary contribution, however, is PUTPUT, a method that prunes low-probability regions of a probabilistic circuit to facilitate the extraction of interpretable logic formulas. Evaluation shows that PUTPUT outperforms competing methods on the playlist generation use case and that it is broader applicable through experiments on open-source datasets.

Acknowledgments. This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme and from the KU Leuven Research Funds (STG/20/052, iBOF/21/075). LDR is also supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Disclosure of Interests. Authors KV, WP and KJ are employed by Tunify, who have provided the music playlist dataset.

Code and appendix The source code and appendices are found on https://github.com/ML-KULeuven/PUTPUT.

References

1. Aucouturier, J., Pachet, F.: Scaling up music playlist generation. In: Proceedings of the 2002 IEEE International Conference on Multimedia and Expo, ICME. vol. 1, pp. 105–108. IEEE Computer Society (2002). https://doi.org/10.1109/ICME.2002.1035729
2. Babaki, B., Guns, T., Nijssen, S., De Raedt, L.: Constraint-Based Querying for Bayesian Network Exploration. In: IDA. Lecture Notes in Computer Science, vol. 9385, pp. 13–24. Springer (2015)
3. Bekker, J., Davis, J.: Learning from positive and unlabeled data: a survey. Mach. Learn. **109**(4), 719–760 (2020)
4. Bonini, T., Gandini, A.: "first week is editorial, second week is algorithmic": Platform gatekeepers and the platformization of music curation. Social Media + Society **5**(4) (2019). https://doi.org/10.1177/2056305119880006
5. Bonnin, G., Jannach, D.: Automated generation of music playlists: Survey and experiments. ACM Comput. Surv. **47**(2), 26:1–26:35 (2014)
6. Choi, Y., Vergari, A., Van den Broeck, G.: Probabilistic circuits: A unifying framework for tractable probabilistic models. UCLA. URL: http://starai.cs.ucla.edu/papers/ProbCirc20.pdf (2020)
7. Chow, C.K.: On optimum recognition error and reject tradeoff. IEEE Trans. Inf. Theory **16**(1), 41–46 (1970)
8. Cohen, G., Afshar, S., Tapson, J., van Schaik, A.: EMNIST: extending MNIST to handwritten letters. In: IJCNN. pp. 2921–2926. IEEE (2017)

9. Cohen, W.W.: Fast effective rule induction. In: Twelfth International Conference on Machine Learning. pp. 115–123. Morgan Kaufmann (1995)
10. Dang, M., Khosravi, P., Liang, Y., Vergari, A., Van den Broeck, G.: Juice: A julia package for logic and probabilistic circuits. In: AAAI. pp. 16020–16023. AAAI Press (2021)
11. Dang, M., Liu, A., Van den Broeck, G.: Sparse probabilistic circuits via pruning and growing. In: NeurIPS (2022)
12. Defferrard, M., Benzi, K., Vandergheynst, P., Bresson, X.: FMA: A dataset for music analysis. In: 18th International Society for Music Information Retrieval Conference (ISMIR) (2017), https://arxiv.org/abs/1612.01840
13. Deng, L.: The MNIST database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Process. Mag. **29**(6), 141–142 (2012)
14. Devos, L., Meert, W., Davis, J.: Versatile verification of tree ensembles. In: ICML. Proceedings of Machine Learning Research, vol. 139, pp. 2654–2664. PMLR (2021)
15. Goyal, K., Meert, W., Blockeel, H., Van Wolputte, E., Vanderstraeten, K., Pijpops, W., Jaspers, K.: Automatic generation of product concepts from positive examples, with an application to music streaming. In: BNAIC/BENELEARN. Communications in Computer and Information Science, vol. 1805, pp. 47–64. Springer (2022)
16. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE **40**(9), 1098–1101 (1952). https://doi.org/10.1109/JRPROC.1952.273898
17. Li, T., Wang, C., Ma, Y., Ortal, P., Zhao, Q., Stenger, B., Hirate, Y.: Learning classifiers on positive and unlabeled data with policy gradient. In: ICDM. pp. 399–408. IEEE (2019)
18. Liu, A., Van den Broeck, G.: Tractable regularization of probabilistic circuits. In: NeurIPS. pp. 3558–3570 (2021)
19. Lundberg, S.M., Lee, S.: A unified approach to interpreting model predictions. In: NIPS. pp. 4765–4774 (2017)
20. Michalski, R.S.: A theory and methodology of inductive learning. Artif. Intell. **20**(2), 111–161 (1983)
21. Mordelet, F., Vert, J.: A bagging SVM to learn from positive and unlabeled examples. Pattern Recognit. Lett. **37**, 201–209 (2014)
22. Muggleton, S.H.: Duce, an oracle-based approach to constructive induction. In: IJCAI. pp. 287–292. Morgan Kaufmann (1987)
23. Murdoch, W., Singh, C., Kumbier, K., Abbasi Asl, R., Yu, B.: Definitions, methods, and applications in interpretable machine learning. Proceedings of the National Academy of Sciences **116**, 201900654 (10 2019). https://doi.org/10.1073/pnas.1900654116
24. Pichl, M., Zangerle, E., Specht, G.: Understanding Playlist Creation on Music Streaming Platforms. In: IEEE International Symposium on Multimedia, ISM. pp. 475–480. IEEE Computer Society (2016). https://doi.org/10.1109/ISM.2016.0107
25. Pronzato, L., Wynn, H., Zhigljavsky, A.: A generalized golden-section algorithm for line search. IMA Journal of Mathematical Control and Information **15** (06 1998). https://doi.org/10.1093/imamci/15.2.185
26. Ribeiro, M.T., Singh, S., Guestrin, C.: "why should I trust you?": Explaining the predictions of any classifier. In: KDD. pp. 1135–1144. ACM (2016)
27. Rosenberg, G., Brubaker, J.K., Schuetz, M.J.A., Salton, G., Zhu, Z., Zhu, E.Y., Kadıoğlu, S., Borujeni, S.E., Katzgraber, H.G.: Explainable artificial intelligence using expressive boolean formulas. Machine Learning and Knowledge Extraction **5**(4), 1760–1795 (2023). https://doi.org/10.3390/make5040086, https://www.mdpi.com/2504-4990/5/4/86

28. Sakurai, K., Togo, R., Ogawa, T., Haseyama, M.: Controllable music playlist generation based on knowledge graph and reinforcement learning. Sensors **22**(10), 3722 (2022)
29. Samer, M., Szeider, S.: Algorithms for propositional model counting. J. Discrete Algorithms **8**(1), 50–64 (2010)
30. Shih, A., Choi, A., Darwiche, A.: A Symbolic Approach to Explaining Bayesian Network Classifiers. In: IJCAI. pp. 5103–5111. ijcai.org (2018)
31. UCI MLR: Mushroom. UCI Machine Learning Repository (1987), doi: 10.24432/C5959T
32. UCI MLR: Molecular Biology (Splice-junction Gene Sequences). UCI Machine Learning Repository (1992), doi: 10.24432/C5M888
33. Verreet, V., De Raedt, L., Bekker, J.: Modeling PU learning using probabilistic logic programming. Mach. Learn. **113**(3), 1351–1372 (2024)
34. Vreeken, J., van Leeuwen, M., Siebes, A.: Krimp: mining itemsets that compress. Data Min. Knowl. Discov. **23**(1), 169–214 (2011)
35. Wang, E., Khosravi, P., Van den Broeck, G.: Probabilistic sufficient explanations. In: IJCAI. pp. 3082–3088. ijcai.org (2021)
36. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. CoRR **abs/1708.07747** (2017)
37. Yuan, Y., Bai, F.: Absolute value inequality svm for the pu learning problem. Mathematics **12**, 1454 (05 2024). https://doi.org/10.3390/math12101454
38. Zhang, H., Chen, Q., Zou, Y., Pan, Y., Wang, J., Stevenson, M.: Document set expansion with positive-unlabeled learning: A density estimation-based approach. CoRR **abs/2401.11145** (2024)