The aitools library

Wieger Wesselink

October 2, 2024

1 Introduction

The aitools library is a C++ library that contains a few data structures and algorithms related to AI, see https://github.com/wiegerw/aitools.It currently supports binary decision trees, random forests, probabilistic circuits, generative forests and some algorithms. It was originally intended as a companion library for the paper *Joints in Random Forests*, see [2], but it was never used as such.

This document contains precise mathematical specifications of the data structures and algorithms that are used in the aitools library. This is not a tutorial, so it is assumed that the reader is already familiar with the material. In Appendix A some notation used in the algorithms is explained.

2 Graphs

Let $G = (V, E)$ with $E \subseteq V \times V$ be a directed graph. We write $u \to v$ whenever $(u, v) \in E$. Let $\to^*$ be the reflexive-transitive closure of $\to$. We define

$$\mathsf{pred}(v) = \{u \in V \mid u \to v\}$$

$$\mathsf{succ}(u) = \{v \in V \mid u \to v\}$$

$$\mathsf{desc}(u) = \{v \in V \mid u \to^* v\}$$

A node u is called a leaf or terminal node if $\mathsf{succ}(u) = \emptyset$.

A *topological ordering* of an acyclic graph G is a linear ordering of the vertices V such that for every directed edge $(u, v) \in E$ we have that u comes before v in the ordering. In the context of probabilistic circuits, this ordering is also known as *feedforward order*.

3 Random variables

Let $\mathbf{X} = \{X_1, \ldots, X_m\}$ be a set of random variables or features. We assume that each continuous variable X_i assumes values in some compact set $\mathcal{X}_i \subseteq \mathbb{R}$ and each discrete variable X_i assumes values in $\mathcal{X}_i = \{1, \ldots, K_i\}$, where K_i is the number of states for X_i. The feature space is denoted as

$$\boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times \ldots \mathcal{X}_m.$$

Let $\mathcal{D} = \{x_1, \ldots, x_n\}$ be a data set for the variables $\mathbf{X}$. Elements of $\mathcal{D}$ may have missing values, which we denote with the symbol $\perp$. Hence

$$x_{ij} \in \mathcal{X}_i \cup \{\perp\} \quad (1 \leq i \leq m, 1 \leq j \leq n).$$

In classification problems, there is also an output variable Y (the *class variable*) that assumes values in $\mathcal{Y} = \{1, \ldots, K\}$. A data set for a classification problem consists of pairs of inputs and outputs: $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ with $y_i \in \mathcal{Y}$.

We define the function $\mathsf{project}$ that projects a data set $\mathcal{D}$ on coordinate i as follows:

$$\mathsf{project}(\mathcal{D}, i) = \{x_i \mid x \in \mathcal{D}\}.$$

We define the function ncat that operates on random variables $X_i \in \mathbf{X}$ as follows:

$$\mathsf{ncat}(X_i, \mathcal{D}) = |\mathsf{project}(\mathcal{D}, i)|,$$

In particular we have

$$\mathsf{ncat}(X_i) = \mathsf{ncat}(X_i, \mathcal{X}_i).$$

Furthermore we define the function counts that counts the number of samples in class k as:

$$\mathsf{count}(\mathcal{D}, k) = |\{(x_i, y_i) \in \mathcal{D} \mid y_i = k\}|.$$

3.1 The Kendall rank correlation test

Let $(x_1, y_1), ..., (x_n, y_n)$ be a set of observations of the joint random variables X and Y, such that all the values of (x_i) and (y_i) are unique (the ties are neglected for simplicity). Any pair of observations (x_i, y_i) and (x_j, y_j), where $i < j$, are said to be concordant if the sort order of (x_i, x_j) and (y_i, y_j) agrees: that is, if either both $x_i > x_j$ and $y_i > y_j$ holds or both $x_i < x_j$ and $y_i < y_j$; otherwise they are said to be discordant.

The Kendall τ coefficient is defined as:

$$\tau = \frac{2}{n(n-1)} \sum_{i<j} \operatorname{sgn}(x_i - x_j) \operatorname{sgn}(y_i - y_j),$$

where

$$\operatorname{sgn}(x) = \begin{cases} -1, & \text{x} < 0 \\ 0, & \text{x} = 0 \\ 1, & \text{x} > 0 \end{cases}$$

Let $(X_1, Y_1), \ldots (X_n, Y_n)$ be a sample of n pairs of observations. The rank correlation coefficient τ of Kendall is defined as

$$\tau = 1 - \frac{2K_n}{\binom{n}{2}},$$

where K_n is the number of inversions: the number of pairs $\{(X_i, Y_i), (X_j, Y_j)\}$ such that $X_i < X_j$ and $Y_i > Y_j$ for $i < j$, $i = 1, \ldots n-1$ and $j = 2, \ldots n$.

4 Probability distributions

4.1 Uniform distribution

A continuous uniform distribution on an interval $[a, b]$ is denoted as $U(a, b)$, and has the following probability density function:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

and cumulative density function

$$F(x) = \begin{cases} 0 & \text{for } x < a \\ \frac{x-a}{b-a} & \text{for } a \leq x \leq b \\ 1 & \text{for } x > b \end{cases}$$

4.2 Normal distribution

A normal or Gaussian distribution $\mathcal{N}(\mu, \sigma)$ is a distribution with the following probability density function:

$$\phi(\mu, \sigma; x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

and cumulative distribution function

$$\Phi(\mu, \sigma; x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-(t-\mu)^2}{2\sigma^2}} dt,$$

The parameter μ is the mean, and σ is the standard deviation of the normal distribution. The function Φ can be expressed in terms of the error function as follows:

$$\Phi(\mu, \sigma; x) = \frac{1}{2}(1 + \operatorname{erf}(\frac{(x-\mu)/\sigma}{\sqrt{2}})).$$

$$\Phi^{-1}(\mu, \sigma; x) = \mu + \sigma\sqrt{2}\operatorname{erf}^{-1}(2x - 1).$$

Sampling a normal distribution can be done by first randomly drawing a value $p \sim U(0, 1)$ and then transforming it to a value x using

$$x = \mu + \sigma \cdot \Phi^{-1}(0, 1; p) = \mu + \sigma \cdot \sqrt{2}\operatorname{erf}^{-1}(2p - 1).$$

4.3 Standard normal distribution

The standard normal distribution is $\mathcal{N}(0, 1)$. It has the following cumulative distribution function:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt,$$

which is closely related to the error function erf:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_{-\infty}^{x} e^{-t^2} dt.$$

We have

$$\Phi^{-1}(x) = \sqrt{2}\operatorname{erf}^{-1}(2x - 1).$$

4.4 Truncated normal distribution

A truncated normal distribution is obtained from a normal distribution $\mathcal{N}(\mu, \sigma)$ by truncating it to an interval $[a, b]$ and then scaling it appropriately, see also [4]. It has the following probability density function:

$$\psi(\mu, \sigma, a, b; x) = \begin{cases} 0 & \text{if } x < a \\ \frac{\phi(\mu,\sigma;x)}{\Phi(\mu,\sigma;b)-\Phi(\mu,\sigma;a)} & \text{if } a \leq x \leq b \\ 0 & \text{if } x > b, \end{cases}$$

where μ and σ are the parameters of the original distribution. The cumulative density function is given by

$$\Psi(\mu, \sigma, a, b; x) = \begin{cases} 0 & \text{if } x < a \\ \frac{\Phi(\mu,\sigma;x)-\Phi(\mu,\sigma;a)}{\Phi(\mu,\sigma;b)-\Phi(\mu,\sigma;a)} & \text{if } a \leq x \leq b \\ 1 & \text{if } x > b. \end{cases}$$

and the inverse by

$$\Psi^{-1}(\mu, \sigma, a, b; x) = \Phi^{-1}(\mu, \sigma;\ \Phi(\mu, \sigma; a) + x \cdot (\Phi(\mu, \sigma; b) - \Phi(\mu, \sigma; a)))$$

Sampling a truncated normal distribution can be done by first drawing a value $p \sim U(0, 1)$ and then transforming it to a value x using

$$x = \Psi^{-1}(\mu, \sigma, a, b; p).$$

4.5 Bernoulli distribution

A categorical distribution is a discrete probability distribution that takes the value 1 with probability p and the value 0 with probability $1 - p$. The probability mass function is given by

$$f(x = i) = p_i.$$

4.6 Categorical distribution

A categorical distribution is a discrete probability distribution that models the possible results of a random variable that can take on one of K possible categories, with the probability of each category separately specified. It is characterized by k probabilities $[p_1, \ldots, p_k]$ with $p_i \geq 0$ and $\sum_{i=1}^{k} p_i = 1$. The probability mass function is given by

$$f(x = i) = \begin{cases} p & \text{if } i = 1 \\ 1 - p & \text{if } i = 0 \end{cases}$$

4.7 Multinomial distribution

A multinomial distribution is a discrete probability distribution that models the probability of counts for each side of a k-sided die rolled n times. It is characterized by k probabilities $[p_1, \ldots, p_k]$ with $p_i \geq 0$ and $\sum_{i=1}^{k} p_i = 1$, and a number of trials n. The probability mass function is given by

$$f(x_1, \ldots, x_n) = \frac{n!}{x_1! \ldots x_k!}\, p_1^{x_1} \ldots p_k^{x_k}.$$

5 Decision trees

A *decision tree* (DT) is a tree $G = (V, E)$ that is used to partition the feature space $\boldsymbol{\mathcal{X}}$ into a number of disjoint subsets. We denote the root of the tree as $\mathsf{root}(G)$. Each node u of the tree is associated with a subset $\boldsymbol{\mathcal{X}}_u \subseteq \boldsymbol{\mathcal{X}}$ of the feature space. We have $\boldsymbol{\mathcal{X}}_{\mathsf{root}(G)} = \boldsymbol{\mathcal{X}}$. Each non-terminal node u is labeled with a *decision tree classifier* $\mathit{split}(u)$ that partitions $\boldsymbol{\mathcal{X}}$ into disjoint subsets $\{U_1, \ldots U_p\}$, and such that $\boldsymbol{\mathcal{X}}_{v_i} = \boldsymbol{\mathcal{X}}_u \cap U_i$, for $v_i \in \mathsf{succ}(u)$. In particular we have that the leaf nodes of G define a partition of the feature space: $\boldsymbol{\mathcal{X}} = \cup\{\boldsymbol{\mathcal{X}}_v \mid v \in V \wedge \mathsf{succ}(v) = \emptyset\}$.

A decision tree is usually defined over a data set $\mathcal{D}$. Each node u is associated with the subset $\mathcal{D}_u = \boldsymbol{\mathcal{X}}_u \cap \mathcal{D}$. We define $\mathsf{count}(u) = |\mathcal{D}_u|$.

5.1 Decision tree classifiers

This section gives an overview of the supported decision tree classifiers. Currently, all of them are binary splits, i.e. they partition the feature space in two subsets. For each classifier, a function partition-number is defined that assigns a partition to each element x of the feature space. In case of a binary split, the partition numbers are 1 and 2. Furthermore, a function domain is defined that specifies for each variable the part of the domain where it can be nonzero. For the root u of the decision tree, we have $\mathsf{domain}(u, X_i) = \mathcal{X}_i$ for each variable $X_i \in \mathbf{X}$.

Single split

A *single split* $\mathit{SingleSplit}(X_i, v)$ is an axis-aligned split defined by a discrete random variable X_i with domain $\mathcal{X}_i$ and a value $v \in \mathcal{X}_i$. We define

$$\mathsf{partition\text{-}number}(\mathit{SingleSplit}(X_i, v), x) = \begin{cases} 1 & \text{if } x_i = v \\ 2 & \text{otherwise.} \end{cases}$$

$$\begin{aligned} \mathsf{domain}(v_1, X_i) &= \{v\} \\ \mathsf{domain}(v_2, X_i) &= \mathsf{domain}(u, X_i) \setminus \{v\}, \end{aligned}$$

where $\mathsf{succ}(u) = \{v_1, v_2\}$. For other variables X_j $(j \neq i)$, we have $\mathsf{domain}(v_1, X_j) = \mathsf{domain}(v_2, X_j) = \mathsf{domain}(u, X_j)$.

Subset split

A *subset split* $\mathit{SubsetSplit}(X_i, V)$ is an axis-aligned split that is defined by a discrete random variable X_i with domain $\mathcal{X}_i$ and a set of values $V \subseteq \mathcal{X}_i$.

$$\mathsf{partition\text{-}number}(\mathit{SubsetSplit}(X_i, V), x) = \begin{cases} 1 & \text{if } x_i \in V \\ 2 & \text{otherwise.} \end{cases}$$

$$\begin{aligned} \mathsf{domain}(v_1, X_i) &= \mathsf{domain}(u, X_i) \cap V \\ \mathsf{domain}(v_2, X_i) &= \mathsf{domain}(u, X_i) \setminus V, \end{aligned}$$

where $\mathsf{succ}(u) = \{v_1, v_2\}$. For other variables X_j $(j \neq i)$, we have $\mathsf{domain}(v_1, X_j) = \mathsf{domain}(v_2, X_j) = \mathsf{domain}(u, X_j)$.

Threshold split

A *threshold split* $\mathit{ThresholdSplit}(X_i, v)$ is an axis-aligned split defined by a continuous random variable X_i with domain $\mathcal{X}_i$ and value $v \in \mathcal{X}_i$. We define

$$\mathsf{partition\text{-}number}(\mathit{ThresholdSplit}(X_i, v), x) = \begin{cases} 1 & \text{if } x_i < v \\ 2 & \text{otherwise.} \end{cases}$$

$$\begin{aligned} \mathsf{domain}(v_1, X_i) &= \mathsf{domain}(u, X_i) \cap (-\infty, v) \\ \mathsf{domain}(v_2, X_i) &= \mathsf{domain}(u, X_i) \cap [v, +\infty), \end{aligned}$$

where $\mathsf{succ}(u) = \{v_1, v_2\}$. For other variables X_j $(j \neq i)$, we have $\mathsf{domain}(v_1, X_j) = \mathsf{domain}(v_2, X_j) = \mathsf{domain}(u, X_j)$.

Applying a splitting criterion to a data set

When constructing a decision tree, a splitting criterion is used to partition a data set. Let $\mathcal{D}$ be a data set, and let u be a decision tree node with scope X_i, splitting criterion *split*, and with successors $\mathsf{succ}(u) = \{v_1, \ldots, v_p\}$. Then

$$\mathsf{apply\text{-}split}(\mathit{split}, \mathcal{D}) = [\{x \in \mathcal{D} \mid \mathsf{partition\text{-}number}(\mathit{split}, x) = 1\}, \ldots, \{x \in \mathcal{D} \mid \mathsf{partition\text{-}number}(\mathit{split}, x) = p\}].$$

Indicator functions

Each decision tree classifier *split* associated with node u naturally defines an indicator function for each outgoing edge. We define

$$\mathsf{indicator}(u, j) = \mathbb{1}_{\boldsymbol{\mathcal{X}}_j},$$

where

$$\boldsymbol{\mathcal{X}}_j = \{x \in \boldsymbol{\mathcal{X}} \mid \mathsf{partition\text{-}number}(\mathit{split}, x) = j\}.$$

Missing values

When the data set contains missing values for variable X_i, the following approach is taken. When computing the gain of a split, samples x with $x_i = \perp$ are simply ignored. When partitioning a data set using a split, the samples with $x_i = \perp$ are randomly assigned to one of the partitions.

5.2 Executing a decision tree

Let $x \in \boldsymbol{\mathcal{X}}$ be an arbitrary element of the feature space. The algorithms below determines a leaf node to which x corresponds.

Algorithm 1 Executing a decision tree

Input: A decision tree $G = (V, E)$ and a sample $x \in \boldsymbol{\mathcal{X}}$
Output: A leaf node $v \in V$.

```
1: function PREDICT(G = (V, E), x)
2:     u := root(G)
3:     let succ(u) = {v_1, ..., v_p}
4:     while true do
5:         if succ(u) = ∅ then
6:             break
7:         j := partition-number(split(u), x)        ▷ split(u) is the split criterion of node u
8:         u := v_j
9:     return u
```

5.3 Learning a decision tree

In this section we describe an algorithm for learning a decision tree for a classification problem with data set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ that is defined on random variables $\mathbf{X} = \{X_1, \ldots, X_m\}$ and class variable Y.

Impurity measures

There are several *impurity measures* of a data set $\mathcal{D}$ that can be used to select a suitable split. The *Gini index* and the *cross-entropy* are two impurity measures on a data set that are defined as follows (see also [3]):

$$\mathsf{gini\text{-}index}(\mathcal{D}) = \sum_{k=1}^{K} p_k(1 - p_k) = 1 - \sum_{k=1}^{K} p_k^2$$

$$\mathsf{cross\text{-}entropy}(\mathcal{D}) = -\sum_{k=1}^{K} p_k \log(p_k)$$

where $p_k = \mathsf{count}(\mathcal{D}, k) \,/\, |\mathcal{D}|$, i.e. the proportion of samples of class k in $\mathcal{D}$. Furthermore we define

$$\mathsf{purity}(\mathcal{D}) = \max_{1 \le k \le K} p_k.$$

Choosing a best split

Suppose we have a data set $\mathcal{D}$, and a splitter that splits $\mathcal{D}$ into $\{\mathcal{D}_1, \ldots, \mathcal{D}_p\}$. Given an impurity measure *imp*, the quality of the split is then measured by

$$\mathsf{gain}(\mathcal{D}, \{\mathcal{D}_1, \ldots, \mathcal{D}_p\}, imp) = imp(\mathcal{D}) - \sum_{i=1}^{p} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} imp(\mathcal{D}_i).$$

A splitter with the maximum gain is selected. In practice we use the simplified expression below to determine the maximum:

$$\mathsf{gain}_1(\{\mathcal{D}_1, \ldots, \mathcal{D}_p\}, imp) = -\sum_{i=1}^{p} |\mathcal{D}_i| imp(\mathcal{D}_i).$$

This function gives the same results, but is more efficient to compute.

Algorithm 2 Learning a decision tree

Input: A training set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ that is defined on random variables $\mathbf{X} = \{X_1, \ldots, X_m\}$ and class variable Y; the maximum number of split variables M, $(1 \leq M \leq m)$; a family of split functions *Splits*; a function *gain* to measure the quality of a split; a criterion *stop* for ending the recursion.
Output: A decision tree (V, E).

```
1:  function LearnDecisionTree(D, M, Splits, gain, stop)
2:      choose u_0 ∈ V                          ▷ u_0 is a fresh node, taken from a universal set of nodes V
3:      V := {u_0}
4:      E := ∅
5:      D_{u_0} := D
6:      todo := {u_0}
7:      while todo ≠ ∅ do
8:          choose u ∈ todo
9:          todo := todo \ {u}
10:         if stop(D_u) then
11:             continue
12:         Z := random-sample(X, M)            ▷ Z ⊆ X is a random subset of size M
13:         A := argmax_{split ∈ Splits(D_u, Z)} gain(apply-split(split, D_u))
14:         if A = ∅ then continue
                                                ▷ A = ∅ means that no suitable split is found
15:         choose split ∈ A
16:         for D ∈ apply-split(split, D_u) do
17:             choose v ∈ V \ V                ▷ v is a fresh node
18:             V := V ∪ {v}
19:             E := E ∪ {(u, v)}
20:             todo := todo ∪ {v}
21:             D_v := D
22:     return (V, E)
```

Split families

The family of split functions $\mathit{Splits}(\mathcal{D}, \boldsymbol{Z})$ that is considered can for example be chosen as follows:

$$\mathit{Splits}(\mathcal{D}, \mathbf{Z}) = \begin{array}{l} \{\mathit{SingleSplit}(X_i, v) \mid X_i \in \boldsymbol{Z} \wedge \mathsf{ncat}(X_i) \leq 5 \wedge v \in \mathsf{project}(\mathcal{D}, i)\} \cup \\ \{\mathit{ThresholdSplit}(X_i, v) \mid X_i \in \boldsymbol{Z} \wedge \mathsf{ncat}(X_i) > 5 \wedge v \in \mathsf{project}(\mathcal{D}, i)\}. \end{array}$$

.

Stop criterion

A possible stop criterion is

$$stop(\mathcal{D}_u) = |\mathcal{D}_u| \leq \text{min-samples-leaf} \vee \text{mis-classification}(\mathcal{D}_u) \leq 0.01 \vee \text{depth}(u) > \text{max-depth}$$

where min-samples-leaf is a user defined constant and depth(u) is the distance between the root of the decision tree and u.

6 Random forests

A random forest is a union of decision trees $\{G_i = (V_i, E_i)\}_{i \in I}$ that are all defined on the same feature space $\boldsymbol{\mathcal{X}}$. The trees are disjoint, i.e. $V_i \cap V_j = \emptyset$ for $i \neq j$.

6.1 Learning a random forest

The following algorithm is used to learn a random forest from a training set $\mathcal{D} = \{x_1, \ldots, x_n\}$ that is defined on random variables $\mathbf{X} = \{X_1, \ldots, X_m\}$.

Algorithm 3 Learning a random forest

Input: A training set $\mathcal{D} = \{x_1, \ldots, x_n\}$ that is defined on random variables $\mathbf{X} = \{X_1, \ldots, X_m\}$; a function *sample* that draws a sample of $\mathcal{D}$; a fraction $p \in [0, 1]$ that determines the size of the sample; the number d of decision trees in the forest; the maximum number of split variables M, $(1 \leq M \leq m)$; a family of split functions *Splits*; a function *gain* to measure the quality of a split; a criterion *stop* for ending the recursion.
Output: A random forest

1: **function** LearnRandomForest($\mathcal{D}, \mathbf{X}$, *sample*, p, d, M, *Splits*, *gain*, *stop*)
2: $\quad$ *result* $:= \emptyset$
3: $\quad$ **for** $1 \leq i \leq d$ **do**
4: $\quad\quad$ $\mathcal{D}' :=$ *sample*$(\mathcal{D}, p)$ $\quad\quad$ ▷ $|\mathcal{D}'| \simeq p|\mathcal{D}|$
5: $\quad\quad$ *result* $:=$ *result* $\cup$ LearnDecisionTree($\mathcal{D}', M$, *Splits*, *gain*, *stop*)
6: $\quad$ **return** *result*

7 Probabilistic circuits

A *probabilistic circuit* (PC) is a rooted directed acyclic graph $G = (V, E)$. Each terminal node (also called *distribution node*) represents a multivariate probabilistic distribution. We denote the root of the PC G as $\mathsf{root}(G)$. There are two types of non-terminal nodes: sum nodes and product nodes, and there are many different kinds of terminal nodes, see also [1]. For each node $u \in V$ an evaluation function $\mathsf{evi}(u, x)$ is defined that is used to compute the probability of the node, where $x \in \boldsymbol{\mathcal{X}}$ is a given sample.

7.1 Definitions

We denote the set of random variables that correspond to terminal node v as $\mathsf{scope}(v)$. This definition is extended to a non-terminal node u using

$$\mathsf{scope}(u) = \cup\{\mathsf{scope}(v) \mid v \in \mathsf{desc}(u) \wedge \mathsf{succ}(v) = \emptyset\}.$$

An *unnormalized distribution* is any nonnegative function $\Phi(x)$ where $\exists x : \Phi(x) > 0$.

For each random variable X_i and each $t \in \mathcal{X}_i$ we define an *indicator variable* $\lambda_{X_i=t}$, which is a function on $\boldsymbol{\mathcal{X}}$ defined as

$$\lambda_{X_i=t}(x) = \begin{cases} 1 & \text{if } x_i = t \\ 0 & \text{otherwise.} \end{cases}$$

Definition 1 *(Network Polynomial) Let Φ be an unnormalized probability distribution over random variables $\mathbf{X}$ with finitely many states and $\boldsymbol{\lambda}$ their indicator variables. The network polynomial f_Φ of Φ is defined as*

$$f_\Phi(\boldsymbol{\lambda}) := \sum_{x \in \boldsymbol{\mathcal{X}}} \Phi(x) \prod_{\lambda_{X_i=t} \in \boldsymbol{\lambda}} \lambda_{X_i=t}.$$

Definition 2 *(SPN Distribution) Let $\mathcal{S}$ be a Sum-product network over $\mathbf{X}$. The distribution represented by $\mathcal{S}$ is defined as*

$$P_{\mathcal{S}}(x) := \frac{\mathit{evi}(\mathit{root}(\mathcal{S}), x)}{\sum_{x' \in \boldsymbol{\mathcal{X}}} \mathit{evi}(\mathit{root}(\mathcal{S}), x')}.$$

Definition 3 *(Smoothness a.k.a. completeness) A sum node u is called called* smooth *if its children have the same scope: $\mathit{scope}(v) = \mathit{scope}(w)$ for any $v, w \in \mathit{succ}(u)$. A PC $\mathcal{P}$ is called smooth if every sum node in $\mathcal{P}$ is smooth.*

Definition 4 *(Consistency) A product node u is called called* consistent *if $\forall v, w \in \mathit{succ}(u), v \neq w$ it holds that $\lambda_{X=x} \in \mathit{desc}(v) \Rightarrow \forall x' \neq x : \lambda_{X=x'} \notin \mathit{desc}(w)$.*

Definition 5 *(Decomposability) A product node u is called* decomposable *if its children have non-overlapping scopes: $\mathit{scope}(v) \cap \mathit{scope}(w) = \emptyset$ for any $v, w \in \mathit{succ}(u)$ with $v \neq w$. A PC is called decomposable if all its product nodes are decomposable.*

Definition 6 *(Shared random variables) A random variable X is a* shared *random variable of product node u if there are $v, w \in \mathit{succ}(u)$ with $v \neq w$ such that $X \in \mathit{scope}(v) \cap \mathit{scope}(w)$.*

Definition 7 *(Locally normalized) A sum node u is called called* locally normalized *if the weights of its outgoing edges sum to 1. A PC $\mathcal{P}$ is called locally normalized if every sum node in $\mathcal{P}$ is locally normalized.*

Definition 8 *(Deterministic) A PC is called* deterministic *if it holds that for each complete sample $x \in \boldsymbol{\mathcal{X}}$, each sum node has at most one non-zero child.*

7.2 Nodes in a probabilistic circuit

This section gives an overview of the nodes in a PC that are supported. We use the convention that for every node u the function evi returns the value 1 for missing data (denoted as $\perp$):

$$\mathsf{evi}(u, \perp) = 1.$$

Sum nodes

A sum node $u = \mathit{Sum}([w_1, \ldots, w_p])$ is a non-terminal node with $[w_1, \ldots, w_p]$ the edge weights of the outgoing edges. Let $\mathsf{succ}(u) = \{v_1, \ldots, v_p\}$. We have $0 \leq w_i$ $(1 \leq i \leq p)$.

$$\mathsf{evi}(u, x) = \sum_{i=1}^{m} w_i \, \mathsf{evi}(v_i, x)$$

Sum split nodes

A sum split node $u = \mathit{SumSplit}([w_1, \ldots, w_p], \mathit{split})$ is an extension of a sum node with a decision tree classifier *split*. Note that this is not a standard node of probabilistic circuits, but it is used by generative forests that are discussed in section 8. The decision tree qualifier *split* is used to ensure that only one of the successors of u can have a non-zero evaluation:

$$\mathsf{evi}(u, x) = w_j \cdot \mathsf{evi}(v_j, x) \qquad \text{with } j = \mathsf{partition\text{-}number}(\mathit{split}, x).$$

Product nodes

A product node $u = \mathit{Product}()$ is a non-terminal node. Let $\mathsf{succ}(u) = \{v_1, \ldots, v_p\}$. We have

$$\mathsf{evi}(u, x) = \prod_{i=1}^{m} \mathsf{evi}(v_i, x)$$

Normal nodes

A normal node $u = \mathit{Normal}(X_i, \mu, \sigma)$ is a terminal node that models a normal distribution $\mathcal{N}(\mu, \sigma)$ for random variable X_i. We have

$$\mathsf{evi}(u, x) = \phi(\mu, \sigma; x_i).$$

Truncated normal nodes

A truncated normal node $u = \mathit{TruncatedNormal}(X_i, \mu, \sigma, a, b)$ is a terminal node that models a truncated normal distribution $\mathcal{N}(\mu, \sigma)$ on an interval $[a, b]$ for random variable X_i. We have

$$\mathsf{evi}(u, x) = \psi(\mu, \sigma, a, b; x_i).$$

Categorical nodes

A categorical node $u = \mathit{Categorical}(X_i, [p_1, \ldots, p_k])$ is a terminal node that models a categorical distribution with probabilities $[p_1, \ldots, p_k]$ for random variable X_i. It is only defined for integer values $x \in [1, \ldots, k]$. We have

$$\mathsf{evi}(u, x) = p_{x_i}$$

Equal nodes

An equal node $u = \mathit{Equal}(X_i, t)$ is a degenerate case of a categorical distribution with probability 1 for a single integer value $i \in \mathbb{N}$ for random variable X_i. It should only be evaluated for integer values $x \in \mathbb{N}$.

$$\mathsf{evi}(u, x) = \begin{cases} 1 & \text{if } x_i = t \\ 0 & \text{otherwise} \end{cases}$$

Not equal nodes

A not-equal node $u = \mathit{NotEqual}(X_i, t)$ is a degenerate case of a categorical distribution with probability 1 for integer values not equal to a value $i \in \mathbb{N}$ for random variable X_i. It should only be evaluated for integer values $x \in \mathbb{N}$.

$$\mathsf{evi}(u, x) = \begin{cases} 1 & \text{if } x_i \neq t \\ 0 & \text{otherwise} \end{cases}$$

Less nodes

A less node $u = \mathit{Less}(X_i, t)$ evaluates to 1 for values less than a given threshold t for random variable X_i.

$$\mathsf{evi}(u, x) = \begin{cases} 1 & \text{if } x_i < t \\ 0 & \text{otherwise} \end{cases}$$

Greater-equal nodes

A greater node $u = \mathit{GreaterEqual}(X_i, t)$ evaluates to 1 for values greater than or equal to a given threshold t for random variable X_i.

$$\mathsf{evi}(u, x) = \begin{cases} 1 & \text{if } x_i \geq t \\ 0 & \text{otherwise} \end{cases}$$

Subset nodes

A subset node $u = \mathit{Subset}(X_i, V)$ evaluates to 1 for values in a set V for random variable X_i.

$$\mathsf{evi}(u, x) = \begin{cases} 1 & \text{if } x_i \in V \\ 0 & \text{otherwise} \end{cases}$$

7.3 Inference

Algorithm 4 Iterative computation of an EVI query

Input: A probabilistic circuit $G = (V, E)$ over random variables $\mathbf{X}$ and a complete state $x \in \boldsymbol{\mathcal{X}}$.
Output: A probability.

function EVI-ITERATIVE(G, x)
  $U :=$ topological-ordering(G)
  $c := \{\mapsto\}$ ▷ $c \subset U \times \mathbb{R}$ is an empty mapping
  for $u \in U$ **do**
    let succ(u) $= [v_1, \ldots, v_p]$
    if $u = \mathit{Sum}([w_1, \ldots, w_p])$ **then**
      $c[u] := \sum_{j=1}^{p} w_j \cdot c[v_j]$
    else if $u = \mathit{SumSplit}([w_1, \ldots, w_p], \mathit{split})$ **then**
      $j :=$ partition-number(split, x)
      $c[u] := w_j \cdot c[v_j]$
    else if $u = \mathit{Product}()$ **then**
      $c[u] := \prod_{j=1}^{p} c[v_j]$
    else ▷ u is a terminal node
      $c[u] :=$ evi(u, x)
  return $c[U[-1]]$ ▷ The last node $U[-1]$ is the root of the PC

Algorithm 5 Recursive computation of an EVI query

Input: A probabilistic circuit $G = (V, E)$ over random variables $\mathbf{X}$ and a complete state $x \in \boldsymbol{\mathcal{X}}$.
Output: A boolean value.

function EVI-RECURSIVE(G, x)
  return evi(root(G), x)

7.4 Sampling

We start at the root node. If we have a sum node, we pick one of the children by sampling from the categorical distribution defined by the weights of the sum node—intuitively, we will pick the child with largest weight more often—and sample from it. If we have a product node, we sample from all of its children. If we have a distribution node, we just sample from its distribution and write the value into the corresponding variable in our sample.

In this algorithm we start of with an empty vector $\boldsymbol{x}$, and write values to it once we reach a leaf. Note that we sample each variable only once, so the set of scopes of the leaves the function reaches will form a partition of the scope.

Algorithm 6 Sampling

Input: The root node u of the PC one wants to sample from, a vector $\boldsymbol{x}$ where we store the sample.
Output: A sample x.

function SAMPLE($u, \boldsymbol{x}$)
  let $\mathsf{succ}(u) = [v_1, \ldots, v_p]$
  if $u = \mathit{Sum}([w_1, \ldots, w_p])$ **then** ▷ If sum node.
    $j \sim \mathit{Categorical}(.|[w_1, \ldots, w_p])$ ▷ Choose a child according to distribution given by the weights.
    SAMPLE($v_j, \boldsymbol{x}$) ▷ Sample from the chosen child.
  else if $u = \mathit{Product}()$ **then** ▷ If product node.
    for $v_j \in \mathsf{succ}(u)$ **do** ▷ Sample from every child.
      SAMPLE($v_j, \boldsymbol{x}$)
  else ▷ A leaf node with a distribution parameterised by θ.
    $\boldsymbol{x}[\mathsf{scope}(u)] \sim \mathit{Distribution}(.|\theta)$

7.5 Learning a sum-product network

Algorithm 7 Learning a Probabilistic Circuit with LearnSPN

Input: A training set $\mathcal{D} = \{x_1, \ldots, x_n\}$ that is defined on a set random variables $\mathcal{X}$; A function *isplit* that splits the set of variables $\mathcal{X}$ into independent sets; A function *cluster* that splits the instances in $\mathcal{D}$ into K clusters.
Output: A Probabilistic Circuit (V, E).

function LEARNSPN($\mathcal{D}, \mathcal{X}$, *isplit*, *cluster*, K)
  if $|\mathcal{X}| = 1$ **then** ▷ we have a single variable in the scope.
    return univariate distribution node fitted on $\mathcal{D}$.
  else
    $\mathcal{X}_1, \ldots, \mathcal{X}_J = \mathit{isplit}(\mathcal{D}, \mathcal{X})$.
    if $J > 1$ **then** ▷ we found at least one independence relationship.
      return $\prod_j \mathit{LearnSPN}(\mathcal{D}, \mathcal{X}_j, \mathit{isplit}, \mathit{cluster}, K)$ ▷ add a prod. node with one child for each $\mathcal{X}_j$.
    else
      $\mathcal{D}_1, \ldots, \mathcal{D}_K = \mathit{cluster}(\mathcal{D}, \mathcal{X}, K)$.
      return $\sum_k \mathit{LearnSPN}(\mathcal{D}_k, \mathcal{X}, \mathit{isplit}, \mathit{cluster}, K)$ ▷ add a sum node with one child for each $\mathcal{D}_k$.

The *isplit* function needs to identify whether there is any independence relationship in a set of variables. This typically means iterating through each possible pair $(X_i, X_j) \in \mathcal{X}$ and running an independence test: Kendall (X_i and X_j continuous), chi-square (X_i and X_j discrete), or Kruskal (one continuous, one discrete). The *cluster* function is usually just K-means with K=2.

8 Generative forests

A generative forest (GeF) is a probabilistic circuit that is derived from a random forest. The structure of the underlying forest is preserved, and also the decision tree classifiers (splitters) are encoded in the PC.

8.1 Converting a random forest to a generative forest

A decision tree $G = (V, E)$ defined over a data set $\mathcal{D}$ can be straightforwardly converted into a generative forest G' as follows, see also [2]. The graph $G' = (V', E')$ contains an isomorphic copy of G. Each non-terminal node $u \in V$ with decision tree classifier *split* is transformed into a sum split node $\mathit{SumSplit}([w_1, \ldots, w_p], \mathit{split})$, with $w_j = |\mathcal{D}_{v_j}|/|\mathcal{D}_u|$ for all $v_j \in \mathsf{succ}(u)$. Each leaf node $u \in V$ is transformed into a small PC that fits a distribution to the samples in $\mathcal{D}_u$. By default u is transformed into a tree $G_u = (V_u, E_u)$ that consists of a product node with an outgoing edge for each random variable X_i as follows:

$$\begin{array}{lcll} V_u & = & \{u, v_1, \ldots, v_m\} & \\ E_u & = & \{(u, v_1), \ldots, (u, v_m)\} & \\ u & = & \mathit{Product}() & \\ v_j & = & \begin{cases} \mathsf{fit\text{-}normal}(u, X_i) & \text{if } X_i \text{ is continuous} \\ \mathsf{fit\text{-}categorical}(u, X_i, \alpha) & \text{otherwise} \end{cases} & \end{array}$$

For a continuous variable X_i we define

$$\mathsf{fit\text{-}normal}(u, X_i) = \begin{cases} \mathit{TruncatedNormal}(X_i, \mathsf{mean}(D_i), \mathsf{stddev}(D_i), a, b) & \text{if } |D_i| > 0 \\ \mathit{TruncatedNormal}(X_i, 0, 1, a, b) & \text{otherwise,} \end{cases}$$

where $D_i = \{x_i \mid x \in \mathcal{D}_u\} \setminus \{\bot\}$, and where a and b are chosen such that $\mathsf{domain}(u, X_i) \subseteq [a, b]$.

For a categorical variable X_i and a given Laplace smoothing factor α we define

$$\mathsf{fit\text{-}categorical}(u, X_i, \alpha) = \mathit{Categorical}(X_i, [p_1, \ldots, p_{\mathsf{ncat}(X_i)}]),$$

where

$$p_k = \frac{|\{x \in \mathcal{D}_u \mid x_i = k\}| + \alpha}{|\{x \in \mathcal{D}_u \mid x_i \neq \bot\}| + \alpha \cdot \mathsf{ncat}(X_i)}.$$

If the number of samples $\mathcal{D}_u$ is larges (say ≥ 30), then the node u can be transformed into a sum-product network using the LearnSPN algorithm, see section 7.5.

A random forest $\{G_i = (V_i, E_i)\}_{i=1}^N$ is converted to a generative forest by first converting each decision tree G_i into a PC G_i', and then add a new root node $r = \mathit{Sum}([\frac{1}{N}, \ldots, \frac{1}{N}])$ with outgoing edges $(r, \mathsf{root}(G_i'))$ for $i = 1 \ldots N$.

8.2 Converting a generative forest to a probabilistic circuit

A generative forest can be converted to an equivalent PC by expanding the sum split nodes. Node $u = \mathit{SumSplit}([w_1, \ldots, w_p], \mathit{split})$ is replaced by $u = \mathit{Sum}([w_1, \ldots, w_p])$. Each edge (u, v_j) is replaced by a subgraph (V_u, E_u) with

$$\begin{array}{lcl} V_u & = & \{u, y_j, z_j, v_j\} \\ E_u & = & \{(u, y_j), (y_j, v_j), (y_j, z_j)\} \\ y_j & = & \mathit{Product}() \\ z_j & = & \mathsf{make\text{-}indicator\text{-}node}(u, j), \end{array}$$

where y_j and z_j are fresh nodes, and

$$
\begin{aligned}
\text{make-indicator-node}(\mathit{SingleSplit}(X_i, v), 0) &= \mathit{Equal}(X_i, v) \\
\text{make-indicator-node}(\mathit{SingleSplit}(X_i, v), 1) &= \mathit{NotEqual}(X_i, v) \\
\text{make-indicator-node}(\mathit{SubsetSplit}(X_i, V), 0) &= \mathit{Subset}(X_i, V) \\
\text{make-indicator-node}(\mathit{SubsetSplit}(X_i, V), 1) &= \mathit{Subset}(X_i, V^C) \\
\text{make-indicator-node}(\mathit{ThresholdSplit}(X_i, v), 0) &= \mathit{Less}(X_i, v) \\
\text{make-indicator-node}(\mathit{ThresholdSplit}(X_i, v), 1) &= \mathit{GreaterEqual}(X_i, v)
\end{aligned}
$$

A Pseudocode

This section describes the pseudocode conventions and data structures that are used in this document.

A.1 Attributes

In many algorithms objects are manipulated that have various attributes. For example a node u in a graph might be assigned the label a, and a common way to denote this is using the statement $u.label := a$. We do not allow this kind of object oriented notation. Instead our way to handle this is to introduce a global mapping *label*, and to write $label[u] := a$ instead. This is done to keep the amount of concepts used in the pseudocode as small as possible. In an actual implementation it may be very inefficient to store these attributes in a separate mapping. But we consider it obvious that the implementer has the freedom to choose an efficient way to store a mapping.

A.2 Lists

Let l and m be lists, and i, j natural numbers. We use the following notations:

Expression	Meaning	Precondition
$[\,]$	The empty list	
$[a, b, c]$	The list with elements a, b and c	
$\|l\|$	The number of elements in l	
$l[i]$	The element at position i	$0 \leq i < \|l\|$
$l[-i]$	The element at position $\|l\| - i$	$0 < i \leq \|l\|$
$l[i:j]$	The sublist $[l[i], \ldots, l[j-1]]$	$0 \leq i \leq j < \|l\|$
$l[i:]$	$l[i : \|l\| - 1]$	$0 \leq i \leq \|l\|$
$l[:i]$	$l[0 : i]$	$0 \leq i \leq \|l\|$
$l +\!\!+ m$	The concatenation of l and m	
$a \in l$	$\exists i : 0 \leq i < \|l\| : l[i] = a$	
$\mathsf{index}(l, a)$	The smallest value i such that $l[i] = a$	$a \in l$

Table 1: List operations

A.3 Mappings

We define a mapping as a set of (key, value) pairs. Let $m \subseteq V \times W$ be a mapping, and let $v \in V$ and $w \in W$ be two values. We use the following notations:

Expression	Meaning	Precondition
$\{\mapsto\}$	The empty mapping	
$\mathsf{keys}(m)$	$\{v \in V \mid \exists w \in W : (v, w) \in m\}$	
$\mathsf{values}(m)$	$\{w \in W \mid \exists v \in V : (v, w) \in m\}$	
$m[v]$	The value $w \in W$ such that $(v, w) \in m$	$v \in \mathsf{keys}(m)$
$v \in m$	$v \in \mathsf{keys}(m)$	

Table 2: Map operations

References

[1] YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. oct 2020.

[2] Alvaro H. C. Correia, Robert Peharz, and Cassio de Campos. Joints in random forests, 2020.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer New York, 2009.

[4] J.Burkardt. The truncated normal distribution. Technical report, Florida State University (FSU), Florida, USA, 2014.