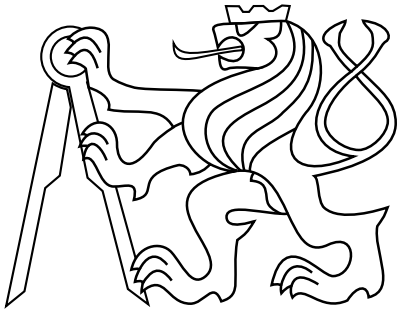
Czech Technical University in Prague

Faculty of Electrical Engineering



# Sum-Product-Set Networks for Density Learning of Tree-Structured Data

**Master's Thesis**

**Bc. Martin Rektoris**

Prague, January 2024

Study programme: Cybernetics and Robotics

**Supervisor: doc. Ing. Václav Šmídl, Ph.D.**
**Supervisor specialist: Ing. Milan Papež, Ph.D.**

# Acknowledgments

I would like to express my gratitude to my supervisors, doc. Ing. Václav Šmídl, Ph.D. and Ing. Milan Papež, Ph.D., especially for all the patience and valuable insights they provided me with while working on this thesis.

The access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" is also gratefully acknowledged.

# MASTER‘S THESIS ASSIGNMENT

## I. Personal and study details

Student's name: **Rektoris Martin** Personal ID number: **483559**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

## II. Master’s thesis details

Master’s thesis title in English:

**Sum-product-set networks for density learning of tree-structured data**

Master’s thesis title in Czech:

**Sum-product-set modely pro učení hustot pravděpodobnosti stromových dat**

Guidelines:

1. Review the state-of-the-art in sum-product networks in the context of density modeling, summarize its advantages and disadvantages. Illustrate their benefits on simple examples.
2. Introduce a probabilistic model of random finite sets, review its properties and demonstrate it on simple examples.
3. Propose an extension of the sum-product networks by an additional node modeling the random finite set. Investigate if the resulting model possesses the same properties as the sum-product networks.
4. Demonstrate the features of the new model on a set of benchmark examples of tree-structured data. Compare the model with competing architectures, such as deep neural networks.

Bibliography / sources:

[1] Poon, H. and Domingos, P., 2011, November. Sum-product networks: A new deep architecture. In 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops) (pp. 689-690). IEEE.
[2] Peharz, R., Tschiatschek, S., Pernkopf, F. and Domingos, P., 2015, February. On theoretical properties of sum-product networks. In Artificial Intelligence and Statistics (pp. 744-752). PMLR.
[3] Vo, B.N., Dam, N., Phung, D., Tran, Q.N. and Vo, B.T., 2018. Model-based learning for point pattern data. Pattern Recognition, 84, pp.136-151.
[4] Pevný, T., Smídl, V., Trapp, M., Poláček, O. and Oberhuber, T., 2020, February. Sum-product-transform networks: Exploiting symmetries using invertible transformations. In International Conference on Probabilistic Graphical Models (pp. 341-352). PMLR.

Name and workplace of master’s thesis supervisor:

**doc. Ing. Václav Šmídl, Ph.D. Artificial Intelligence Center FEE**

Name and workplace of second master’s thesis supervisor or consultant:

**Ing. Milan Papež, Ph.D. Department of Computer Science FEE**

Date of master’s thesis assignment: **08.09.2023** Deadline for master's thesis submission: **09.01.2024**

Assignment valid until:
**by the end of winter semester 2024/2025**

| doc. Ing. Václav Šmídl, Ph.D.<br>Supervisor’s signature | prof. Ing. Michael Šebek, DrSc.<br>Head of department’s signature | prof. Mgr. Petr Páta, Ph.D.<br>Dean’s signature |
|---|---|---|


© ČVUT v Praze, Design: ČVUT v Praze, VIC

## III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

| . | |
|---|---|
| Date of assignment receipt | Student's signature |


© ČVUT v Praze, Design: ČVUT v Praze, VIC

## Declaration

I declare that presented work was developed independently, and that I have listed all sources of information used within, in accordance with the Methodical instructions for observing ethical principles in preparation of university theses.

Date ..... .....

## Abstract

The research and scalability of machine learning research have been accelerated by moving from manual feature engineering to automatic feature extraction. The use of JSON data format in various domains such as cybersecurity, physics or biochemistry development motivated the automated processing of such data. Tree-structured data has previously been shown to generalise the concept of the JSON data format [1]. However, only a mode for discriminative learning of such data has been proposed. In this work, we propose a Sum-Product-Set Network (SPSN), a generative model for tree-structured data based on explicit modelling of its density. We address the challenges of modelling tree-structured data by using the theory of random finite sets. Random finite set theory is combined with a tractable probabilistic model of Sum-Product Networks. The experimental results provide in-depth insights into the strengths and limitations of SPSN in different data domains and highlight the competitiveness of tractable probabilistic models against intractable neural networks.


**Keywords** Sum-Product-Set Networks, Tree-structured data, JSON data format, Probabilistic learning, Density learning, Classification, Clustering, Sum-Product Networks, Multiple-instance learning, Hierarchical multiple-instance learning

## Abstrakt

Výzkum a škálovatelnost výzkumu v oblasti strojového učení se urychlily přechodem od ručního vytváření příznaků k automatické extrakci příznaků. Použití datového formátu JSON v různých oblastech, jako je kybernetická bezpečnost, fyzika nebo biochemie, motivovalo k automatizovanému zpracování i těchto dat. Již dříve se ukázalo, že stromově strukturovaná data zobecňují koncept datového formátu JSON [1]. Byl však navržen pouze způsob diskriminačního učení takových dat. V této práci navrhujeme síť typu SPSN (Sum-Product-Set Network), generativní model pro stromově strukturovaná data založený na explicitním modelování jejich hustoty. Problémy modelování stromově strukturovaných dat řešíme pomocí teorie náhodných konečných množin. Teorie náhodných konečných množin je kombinována s pravděpodobnostním modelem Sum-Product sítí. Experimentální výsledky poskytují hluboký vhled do silných stránek a omezení SPSN v různých datových doménách a zdůrazňují konkurenceschopnost pravděpodobnostních modelů vůči neuronovým sítím.

**Klíčová slova** Sum-Product-Set sítě, Stromově strukturovaná data, JSON datový formát, Pravděpodobnostní učení, Učení hustoty pravděpodobnosti, Klasifikace, Shlukování, Sum-Product sítě, Více instanční učení, Hierarchické více instanční učení

# Contents

# Chapter 1

# Introduction

The use of deep learning has become increasingly dominant in various downstream tasks, notably in computer vision for tasks such as image classification [2], object detection and segmentation [3], or point of interest detection and local feature description [4], and in natural language processing such as machine translation [5, 6]. This surge in adoption has coincided with a shift away from the traditional approach of extensive feature engineering, as machine learning models have increasingly demonstrated the ability to automatically extract relevant features from data.

Firstly, the thesis focuses on developing machine learning models tailored specifically for a particular type of data called tree-structured data. While machine learning has historically concentrated on tensors (like images), sequences (such as language), and, more recently, graphs, this thesis emphasizes JSON files as a data type that has received relatively little attention in the field. Nonetheless, prior successful attempts to model such data have been documented [1].

Secondly, the aim is to leave the dominant data-driven approach in machine learning by proposing a model-based approach rooted in probability theory. Machine learning often involves learning complex functions between observations and desired outputs. We take a step back and explore a model-based strategy. The adoption of probability theory serves as a foundational basis, providing theoretical robustness and soundness to the proposed models. The adoption process includes carefully examining the data's structure and unique characteristics and crafting a model specifically tailored to match the data's structure.

As only discriminative models based on neural networks [1] or distances [7] were proposed for tree-structured data, there is a gap to fill by generative models. Therefore, we aim to develop a generative model of such data.

## 1.1 Structured data

Pevný et al.'s work [8, 9] applied machine learning to hierarchically structured data in cybersecurity, where the dominant approach was extensive feature engineering [10, 11]. Pevný et al.'s hierarchical data model [8], based on a neural network formalism, serves as a generalisation of multiple instance learning [12]. Subsequently, this hierarchical data model found practical application [1] in the automated processing of data stored in the JavaScript Object Notation (JSON) file format [13], which inherently adheres to a similar hierarchical structure, demonstrating the versatility and adaptability of these methods across different data formats and domains.

Mandlík provides a detailed description and extension of Pevný et al.'s approach [8] to hierarchically structured data in his thesis [14]. Mandlík further argues in his thesis that we should build machine learning models directly on structured data. The simple illustration

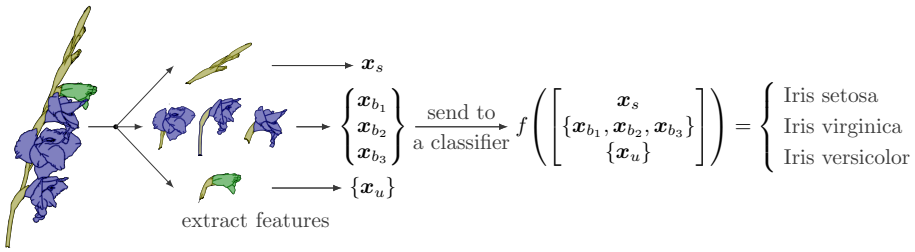



Figure 1.1: Example of iris flower viewed as an instance of tree-structured data. Refer to the source [14] for more details.


```
{
    "ind1": 1, "lumo": -1.246, "inda": 0, "logp": 4.23, "atoms": [
    {
      "element": "c",
        "bonds": [
        {"element": "c", "charge": -0.117, "bond": 7, "atom": 22},
        {"element": "h", "charge":  0.142, "bond": 1, "atom":  3}
      ],
       "charge": -0.117,
         "atom": 22
    },
    ⋮
    {
      "element": "h",
        "bonds": [
        {"element": "c", "charge": -0.117, "bond": 1, "atom": 22}
        ],
       "charge": 0.142,
         "atom": 3
    }
  ]
}
```

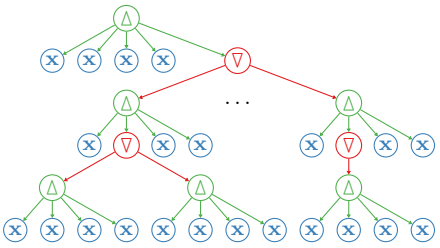



Figure 1.2: (left) Sample of Mutagenesis dataset [19] stored in JSON data format. (right) Leaf-attributed tree (defined later in the Chapter 3) of sample on the left.

is based on a well-known dataset of iris flowers [15]. The main argument is that botanists would classify the flowers used to create the iris dataset by inspecting the whole flower (using structured information), as shown in Figure 1.1, rather than some pre-processed features commonly used in practice. To develop competitive artificial intelligence based on machine learning, we should ideally provide it with the same inputs as humans, which overlaps with the recent shift towards automatic feature extraction.

In this sense, structural data formats such as XML or JSON serve as human-readable ways to store the data and preserve some of its structural information. While the JSON format is widely used in computer security [11, 16, 17], it is not limited to storing data in this domain. It is also used in biochemistry [18], for example, to store information about molecules [19], as shown on the left in Figure 1.2.

## 1.2 Density estimation

Generative modelling is a fundamental aspect of machine learning that has been gaining momentum. Recent advances in this area include realistic image synthesis [20] and text generation using large language models [21]. These models exhibit exceptional expressiveness, generating realistic outputs from probability distribution learned from an excessive amount of existing data. The underlying principle defining the generative model is the probabilistic nature of the model and the probability density it represents.

The knowledge of probability density function is critical to statistical analysis and in-

ference. One can use it to calculate all descriptive statistics approximated from data when the density is unavailable. One of the descriptive statistics includes the highest posterior density region, which, as its name suggests, tries to find a region where a lot of probability density is concentrated. This naturally leads to outlier detection methods based on the assumption that outlying observations lie in areas with low probability density.

The probability density is not only limited to classical analysis but can be used in various machine learning tasks - tasks called supervised, containing data annotations and tasks called unsupervised, not containing annotations. Classification and clustering, respective instances of supervised and unsupervised tasks, can be bridged using latent variable models such as mixture models. Treating the data annotations as a latent variable leads to a natural extension of unsupervised and supervised methods to semi-supervised ones, where the data are only partially annotated. The idea of latent variables generalizes beyond clustering or semi-supervised learning. Latent variables can assist in dealing with missing data. They can be used in imputation models to estimate missing values based on observed data patterns, assuming that the latent variables capture them.

Missing data on the prediction level can be naturally handled by marginal probability densities – missing data values are marginalized out of the probability density. Marginal density plays a role in computing conditional density by utilizing Bayes' rule – extending the concept of prediction beyond predicting only latent variables (data annotations) and data features.

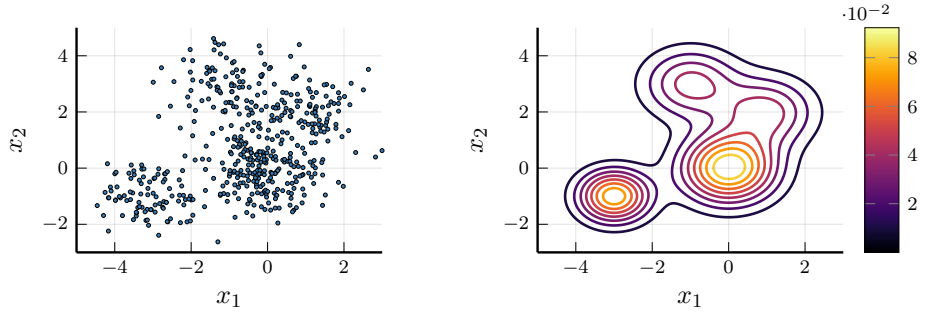



Figure 1.3: (left) Random samples generated from probability distribution described by probability density on the (right).

## 1.3 Structure of the thesis

In this thesis, we propose a generative model for tree-structured data. We are motivated by recent advances in automated JSON data learning, where discriminative methods for classification were proposed. JSON datasets can be realized as a special type of tree-structured data. Since the area is relatively novel, there has yet to be a generative approach for this type of data in its full complexity.

The thesis is organized as follows:

- The first chapter serves as an introduction to probability density, its estimation and use cases. Some modern methods of density estimation are presented. At last, we introduce the probabilistic model of the Sum-Product Network and mention its advantages that revolve around its exact and efficient probabilistic inference.

- The second chapter introduces the studied data (trees) in more detail. Brief prior work on learning special cases of tree-structured data, while special care is taken of probabilistic models of random finite sets. Towards the end of the chapter, the Sum-Product-Set model, an extension of Sum-product networks to leaf-attributed trees, is defined, and its examples and basic capabilities are presented.
- The third chapter defines the experimental tasks which need to be restricted to classification and clustering tasks due to the lack of competitive density estimation approaches of general tree-structured data. Performance measures, datasets used, and experimental setup form the rest of the chapter.
- The fourth chapter presents experimental results and discusses them.
- The last chapter summarizes the achievements of the thesis and explores the possible areas of future work.

# Chapter 2

# Density estimation in $\mathbb{R}^n$

### Construction of probability density

**Probability space** Let us consider a probability space $(\Omega, \sigma(\Omega), \mathbb{P})$, where $\Omega$ represents the sample space, $\sigma(\Omega)$ constitutes a $\sigma$-algebra on $\Omega$ (the event space), and $\mathbb{P} : \Omega \to [0, 1]$ is a measure. If it holds that $\mathbb{P}(\emptyset) = 0$ and $\mathbb{P}(\Omega) = 1$, then $\mathbb{P}$ is referred to as probability measure.

**Random variable** A random variable $X : \Omega \to \mathcal{X}$ is a measurable function between probablity space $(\Omega, \sigma(\Omega), \mathbb{P})$ and measure space $(\mathcal{X}, \sigma(\mathcal{X}), \mu)$. The measurability of $X$ means that $X^{-1}(A) = \{\omega \in \Omega \,|\, X(\omega) \in A\} \in \sigma(\Omega)$ for all $A \in \sigma(\mathcal{X})$. The set $X^{-1}(A)$ is called a pre-image of $A$.

**Probability distribution** We define the distribution $P : \mathcal{X} \to [0, 1]$ of $X$ as $P(A) = \mathbb{P}(X \in A) = \mathbb{P} \circ X^{-1}(A)$ for all $A \in \sigma(\mathcal{X})$, where $P = \mathbb{P} \circ X^{-1}$ is called a push-forward measure. A probability distribution ties together a probability measure and a random variable. It follows that $P(\mathcal{X}) = 1$. Similarly to a probability measure, for the probability distribution, it holds that $P(\emptyset) = 0$ and $P(\mathcal{X}) = 1$.

**Probability density** The Radon-Nikodym theorem [22] states that if a probability distribution $P$ is absolutely continuous with respect to reference measure $\mu$ (defined on measurable space $(\mathcal{X}, \sigma(\mathcal{X}))$), then there exists a measurable function $p : \mathcal{X} \to \mathbb{R}_0^+$, called a probability density, defined as Radon-Nikodym derivative

$$p = \frac{dP}{d\mu}, \tag{2.1}$$

which can be equivalently expressed as a function $p$ satisfying

$$P(A) = \int_A p \, d\mu, \tag{2.2}$$

for all $A \in \sigma(\mathcal{X})$.

When dealing with real-valued variables, $X \in \mathbb{R}^d$, $\mu$ is commonly chosen as the Lebesgue measure [23]. Contrary to that, for discrete-valued random variables $X \in \mathbb{N}$, $\mu$ is chosen as the counting measure [23] and $p$ is then called a probability mass function, not a density function. We will refer to the Radon-Nikodym derivative $p$ simply as probability density regardless of the type of $X$ and the reference measure.

## 2.1 Probabilistic learning

### Estimation of probability density

Suppose we are given a dataset $\mathcal{D} = \{\mathbf{x}_i \in \mathcal{X} \,|\, i = 1 \dots m\}$ where $\mathbf{x}_i \sim p^*(\mathbf{x})$ are presumably independnet and identically distributed (i.i.d) instances of $\mathbf{x}$. However, this true generating (underlying) distribution, $p^*(\mathbf{x})$, is generally unknown. We seek to learn a user-specified density $p(\mathbf{x})$ that approximates the true density $p^*(\mathbf{x})$. An example of probability density and samples generated from it is presented in Figure 2.1.

Throughout this thesis, we will assume parametric density estimation. This setting specifies that the density function is parameterized by $\theta$, an element of the parameter space $\Theta$, a set of all possible configurations of $\theta$. We will be dealing with *proper* densities, meaning that both the structural form of the density $p_\theta$ and the set $\Theta$ are constrained in such a way that for every $\theta \in \Theta$, the density $p_\theta$ is non-negative and integrates to one over the entire space $\mathcal{X}$.

Given this setup, a common approach to density estimation is the maximum likelihood estimation [24]. It proceeds by defining the likelihood function, denoted as $L(\theta|\mathcal{D})$, for the parameters $\theta$, and given the dataset $\mathcal{D}$, which consists of independently and identically distributed (i.i.d.) observations,

$$L(\theta|\mathcal{D}) = \prod_{\mathbf{x}\in\mathcal{D}} p_\theta(\mathbf{x}), \tag{2.3}$$

and the log-likelihood function, denoted as follows:

$$\mathcal{L}(\theta|\mathcal{D}) = \log L(\theta|\mathcal{D}) = \sum_{\mathbf{x}\in\mathcal{D}} \log p_\theta(\mathbf{x}). \tag{2.4}$$

We are interested in finding $\hat{\theta} = \text{argmax}_{\theta\in\Theta}\mathcal{L}(\theta|\mathcal{D})$ which is referred to as the maximum likelihood estimate of $\theta$ based on the dataset $\mathcal{D}$. Both $\mathcal{L}$ and $L$ attain maxima for the same $\theta \in \Theta$, as log is a strictly increasing function.

Additionally, it has been shown in [25] that maximizing the log-likelihood is asymptotically equivalent to minimizing the Kullback-Leiber (KL) divergence $D_{KL}(p^*(\mathbf{x})||p_\theta(\mathbf{x}))$ between the true (unknown) distribution $p^*(\mathbf{x})$ and our approximation $p_\theta(\mathbf{x})$ with respect to $\theta \in \Theta$ as $|\mathcal{D}| \to \infty$. This ensures that the maximum likelihood estimation gives asymptotically the best approximation to the true distribution in terms of the forward KL divergence.

### Unsupervised learning

Unsupervised learning is a machine learning paradigm where models try to capture some underlying pattern in the data without being provided with the pattern during the training. The datasets used in an unsupervised setting can be written as

$$\mathcal{D}_\mathcal{U} = \{\mathbf{x}_i \in \mathcal{X} \,|\, i = 1 \dots m\}$$

where $\mathbf{x}_i \sim p^*(\mathbf{x})$ are assumably i.i.d. Density learning task falls into this paradigm as it provides the most fundamental insight into the data and its generative process. Nevertheless, unsupervised learning encompasses more additional tasks, such as clustering and outlier detection, where both of these can be formulated in probabilistic setting, making effective use of the density estimation.

**Clustering** In clustering, the primary objective is to partition $\mathcal{D}_\mathcal{U}$ into a beforehand unknown number of subsets, commonly referred to as clusters. The samples from $\mathcal{D}_\mathcal{U}$ assigned to the same cluster should be as similar as possible, and the samples assigned to different clusters should be as distinct as possible.

This challenge can be formulated as the search for a function $f : \mathcal{X} \to \mathcal{C}$ that assigns a cluster label $c \in \mathcal{C} = \{1, \ldots, n_c\}$ to observations $\mathbf{x} \in \mathcal{D}_\mathcal{U}$. $n_c$ signifies the number of clusters, a parameter estimated as part of the model learning or specified by the user. The probabilistic formulation of clustering resorts to the estimation of $p(\mathbf{x})$ while assuming the underlying hidden structure by introducing hidden (latent) variable $c \in \mathcal{C}$. This introduction creates a joint probability density $p(\mathbf{x}, c)$ that $p(\mathbf{x}) = \sum_{c\in\mathcal{C}} p(\mathbf{x}, c) = \sum_{c\in\mathcal{C}} p(\mathbf{x}|c)p(c)$. The terms $p(\mathbf{x}|c)$ represent the probability density of each cluster $c$. Models of this type are also called mixture models (more on this later).

The assignment of a sample $\mathbf{x}$ to a specific cluster $c \in \mathcal{C}$ is determined by the most likely component to have generated $x$. In other words, this is expressed as $f(\mathbf{x}) = \text{argmax}_{c\in C} p(\mathbf{x}, c)$.

**Outlier detection** In the outlier detection, we seek to find so-called *outliers*. Hawkins [26] defines them as follows: "*An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism*". Let us assume that we have the true density $p(\mathbf{x})$ of the data at our disposal. Under this definition, the data generated by $p(\mathbf{x})$ is not likely to appear in areas where $p(\mathbf{x})$ is small and is more likely to appear in areas with bigger valued of $p(\mathbf{x})$. In the outlier detection approach based on probability density, sample $\mathbf{x} \in \mathcal{D}_\mathcal{U}$ is classified as an outlier if its density $p(\mathbf{x})$ falls under some non-negative threshold $\tau$. Given a density $p(\mathbf{x})$, threshold $\tau \geq 0$, set of outliers in dataset $\mathcal{D}_\mathcal{U}$ reads

$$\mathcal{D}_\mathcal{O} = \{\mathbf{x} \in \mathcal{D}_\mathcal{U} \,|\, p(\mathbf{x}) < \tau\}.$$

Setting appropriate threshold $\tau$ can done using the notion of highest density regions, defined later in the subsection about probabilistic inference. The outlier detection task is especially difficult because, realistically, we are rarely provided with the density $p(\mathbf{x})$ and therefore need to estimate it from the data.

### Supevised learning

The classification is one of the most notable tasks that fall under the category of supervised problems. In classification tasks, the training data typically takes the form of

$$\mathcal{D}_\mathcal{L} = \{(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y} \mid i = 1, \ldots, m\},$$

where $(\mathbf{x}_i, y_i)$ are independently, identically distributed according to some true (usually unknown) probability distribution $p^*(\mathbf{x}, y)$. Features $\mathbf{x}$ again originate from the feature space $\mathcal{X}$, often a subset of $\mathbb{R}^d$, and classification target (label) $y$ originates from the target space $\mathcal{Y}$, typically a finite subset of $\mathbb{N}$.

The objective is to learn a function, referred to as a classifier, hypothesis or predictor, denoted as $h : \mathcal{X} \to \mathcal{Y}$. Classification models are commonly categorized into two classes [27] – discriminative and generative models. Discriminative models focus on learning the posterior target distribution $p(y|\mathbf{x})$. The classifier is then commonly formulated as Bayes classifier [28] as follows:

$$h(\mathbf{x}) = \text{argmax}_{y\in\mathcal{Y}} p(y|\mathbf{x}). \tag{2.5}$$

Compared to that, the generative approach seeks to model directly joint distribution $p(\mathbf{x}, y)$ instead of posterior target distribution $p(y|\mathbf{x})$. Since $y$ in $p(\mathbf{x}, y)$ is discrete and takes only finitely many values, marginalizing it is and obtaining $p(\mathbf{x}) = \sum_{y \in \mathcal{Y}} p(\mathbf{x}, y)$ is usually straightforward. Obtaining $p(y|\mathbf{x})$ from $p(\mathbf{x}, y)$ is then also straightforward since $p(y|\mathbf{x}) = \frac{p(\mathbf{x},y)}{p(\mathbf{x})} \propto p(\mathbf{x}, y)$. The hypothesis can be constructed in the same manner, $h(\mathbf{x}) = \text{argmax}_{y \in \mathcal{Y}} p(y|\mathbf{x})$; however, it can be also simplified to $h(\mathbf{x}) = \text{argmax}_{y \in \mathcal{Y}} p(\mathbf{x}, y) = \text{argmax}_{y \in \mathcal{Y}} p(\mathbf{x}|y)p(y)$. For generative models, the task of classification is then decomposed into learning the density of $\mathbf{x}$ conditioned on target variable $y$, $p(\mathbf{x}|y)$, and target $y$ prior distribution, $p(y)$. The feature-only density $p(\mathbf{x})$ can be easily recovered.

## 2.2 Probabilistic inference

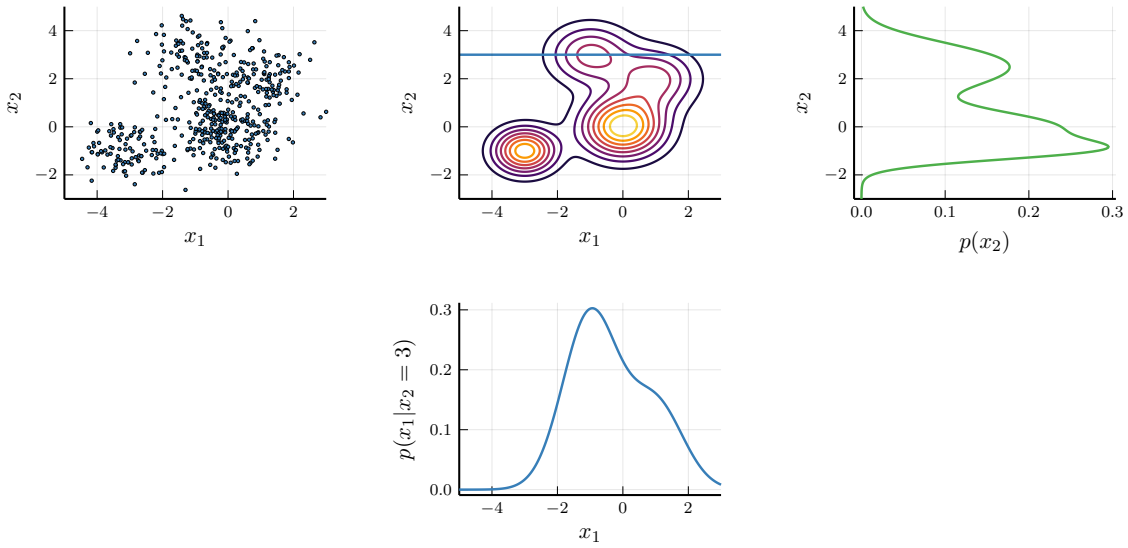



Figure 2.1: An example of some probabilistic queries of a given probability density function $p(\mathbf{x}) = p(x_1, x_2)$: (upper-left) Random samples generated from the distribution $p(\mathbf{x})$ – SAMP query (upper-middle) Contour lines of probability density function $p(\mathbf{x})$, brighter colour means higher density – EVI query, (upper-right) Marginal probability density function $p(x_2)$ – MAR query, (bottom) Conditional probability density function $p(x_1|x_2 = 3)$ – COND query

### Evidence query (EVI)

The evidence query is one of the most fundamental queries. It simply evaluates the density $p(\mathbf{x})$ at $\mathbf{x} \in \mathcal{X}$. The ability of explicit evaluation of $p(\mathbf{x})$ is crucial to perform the statistical tasks mentioned in the introduction, such as classification, clustering or outlier detection. As discussed before, we consider that the density function, $p_\theta(\mathbf{x})$, is parametrized by $\theta$. In this notation, we will refer to the evaluation $p_\theta(\mathbf{x})$ for fixed $\theta$ as evidence evaluation and, and following the classical statistical notion, to $p_\theta(\mathbf{x})$ for fixed $\mathbf{x}$ as likelihood inference. Evaluating likelihood is the first step to performing maximum likelihood estimation of the parameters $\theta$. The sum of log-likelihoods in Equation 2.4 serves as the optimization criterion. It also indicates model quality for different parameters given the same dataset $\mathcal{D}$.

### Marginal query (MAR)

We can think of a marginal query as a partial evidence query. Considering $\mathcal{X} \subseteq \mathbb{R}^n$ as the support of $p(\mathbf{x})$, we want to evaluate a density for a subset of all variables. We split $\mathbf{x}$ into $\mathbf{x}_e$ and $\mathbf{x}_m$ with $e \subset \{1, \ldots, n\}$ and $m = \{1, \ldots, n\} \setminus e$ representing the indexes of observed (partial evidence) and unobserved (missing) parts of $\mathbf{x}$, respectively. For the evaluation of $\mathbf{x}_e$, we marginalize $\mathbf{x}_m$ out of $p(\mathbf{x})$ , which yields

$$p(\mathbf{x}_e) = \int p(\mathbf{x}_e, \mathbf{x}_m)\, d\mathbf{x}_m \,. \tag{2.6}$$

For example, assume that $\mathbf{x} = (x_1, x_2, x_3, x_4)^T \in \mathbb{R}^4$, and that we want to evaluate the partial evidence $\mathbf{x}_e = (x_2, x_3)^T = (1, -2)^T \in \mathbb{R}^2$, meaning values of $x_1$, $x_4$ are unknown. This results in using Equation 2.6 with $\mathbf{x}_e = (1, -2)^T$, $e = \{2, 3\}$, and $m = \{1, 4\}$.

The marginal query allows us to perform the same tasks as the evidence query but now with missing data. We can perform not only inference tasks but also learn from missing data by using the partial evidence $p_\theta(\mathbf{x}_e)$ as the likelihood instead of the full evidence. This is formalized for the classification purposes by Khosravi et al., who used the idea of the expected prediction $\mathrm{EP}(\mathbf{x}_e)$ [29] to tackle the classification with missing features at the prediction time. The expectation of the predictor $p(y|\mathbf{x}_e, \mathbf{x}_m)$ with respect to the conditional density of missing data given observed data, $p(\mathbf{x}_m|\mathbf{x}_e)$, is written as $\mathrm{EP}(\mathbf{x}_e) = \mathbb{E}_{p(\mathbf{x}_m|\mathbf{x}_e)}[p(y|\mathbf{x}_m, \mathbf{x}_e)]$. Eric Wang et al. [30] made use of the exact marginal inference of SPNs to compute the exact expected prediction as follows:

$$\mathrm{EP}(\mathbf{x}_e) = \mathbb{E}_{p(\mathbf{x}_m|\mathbf{x}_e)}[p(y|\mathbf{x}_m, \mathbf{x}_e)] = \int p(y|\mathbf{x}_m, \mathbf{x}_e) p(\mathbf{x}_m|\mathbf{x}_e)\, d\mathbf{x}_m = p(y|\mathbf{x}_e). \tag{2.7}$$

Eric Wang et al. then further used exact computation as a building block for an explanation of classifiers.

As an additional reference, papers [31, 32, 33, 34] consider experiments with marginal queries.

### Conditional query (CON)

The conditional query is simply a query to evaluate the conditional density $p(\mathbf{x}_c|\mathbf{x}_e)$ at $\mathbf{x}_c$ given the partial evidence $\mathbf{x}_e$. We consider $\mathbf{x}$ being split into $\mathbf{x}_e$, $\mathbf{x}_c$, and $\mathbf{x}_m$, where $e \subset \{1, \ldots, n\}$, $c \subset \{1, \ldots, n\}$, $e \cap c = \emptyset$, and $m = \{1, \ldots, n\} \setminus (e \cup c)$. Generally, the ability to compute the conditional query lies in the ability to compute the marginal query as

$$p(\mathbf{x}_c|\mathbf{x}_e) = \frac{p(\mathbf{x}_c, \mathbf{x}_e)}{p(\mathbf{x}_e)} = \frac{\int p(\mathbf{x}_c, \mathbf{x}_e, \mathbf{x}_m)\, d\mathbf{x}_m}{\int\int p(\mathbf{x}_c, \mathbf{x}_e, \mathbf{x}_m)\, d\mathbf{x}_c d\mathbf{x}_m}. \tag{2.8}$$

For example, experiments with conditional distribution can be found in [32].

### Sampling query (SAMP)

The sampling query ensures drawing samples $\mathbf{x}$ from $p(\mathbf{x})$, denoted as $\mathbf{x} \sim p(\mathbf{x})$. The ability to generate random samples is a focal point of current deep generative models. Models like Generative Adversarial Networks(GANs) [35] can generate high-quality random samples

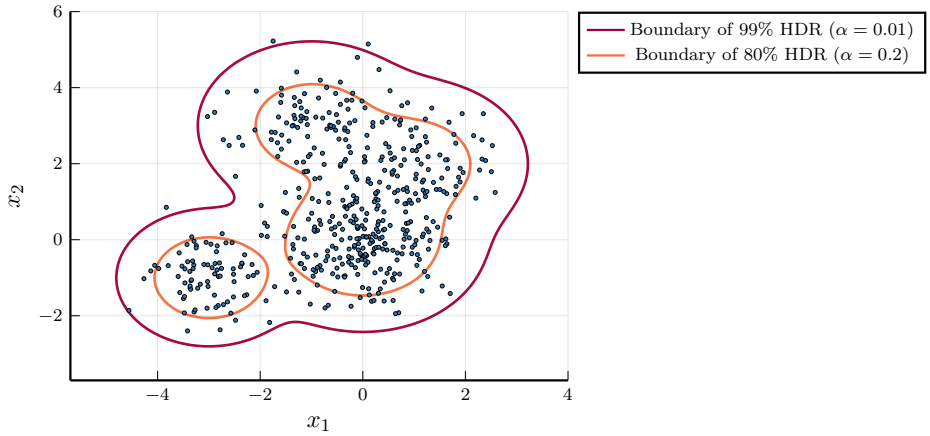



Figure 2.2: An example of boundaries, whose interior defines the highest posterior density region/regions for choices of $\alpha = 0.01$ and $\alpha = 0.2$. The same true density as in figure 2.1, additionally, a few random samples are generated for better visual illustration.

without being able to model the probability density explicitly. This is one of their key advantages in the context of the Monte Carlo sampling since we do not need to resort to the importance sampling.

Monte Carlo methods can be used, for example, for approximating the integrals $\int f(\mathbf{x})p(\mathbf{x})d\mathbf{x}$, where $f : \mathcal{X} \to \mathbb{R}$. The integral can be approximated as $\frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)$ where $\mathbf{x}_i \sim p(\mathbf{x})$.

### Highest density region (HDR)

The task of estimating the highest density region (HDR) is concerned with finding a region $C \subseteq \mathcal{X}$ with a minimal volume such that the probability mass of at least $1 - \alpha$ is concentrated in this region $C$. Formaly, this is realized by solving the following problem:

$$\min_{C \subseteq \mathcal{X}} \quad \lambda(C) \tag{2.9}$$

$$\text{s.t.} \quad \int_C p(\mathbf{x})d\mathbf{x} \geq 1 - \alpha, \tag{2.10}$$

where $\lambda$ is a measure of volume defined on the feature space $\mathcal{X}$ and $\alpha \in (0, 1)$.

The HDR $C$ is, for simplicity, mostly considered to take the form of $C := C_\tau = \{\mathbf{x} \in \mathcal{X} \,|\, p(\mathbf{x}) \geq \tau\}$ [36]. This effectively leads to the simplification of the original optimization problem in Equations 2.9 and 2.10 as follows:

$$\max_{\tau \geq 0} \quad \tau \tag{2.11}$$

$$\text{s.t.} \quad \int_{C_\tau} p(\mathbf{x})d\mathbf{x} \geq 1 - \alpha. \tag{2.12}$$

Note that the threshold $\tau := \tau(\alpha)$ depends on the parameter $\alpha$. An example of HDR and its dependence on $\alpha$ is shown in Figure 2.2

### Tractability

Given a probabilistic model $M$, a set of probabilistic queries $\mathcal{Q}$ is tractable if each query in $\mathcal{Q}$ can be computed **exactly** and with time complexity that is polynomial in the size of the model. By polynomial complexity, we understand that there exists $k \in \mathbb{N}$ such that the complexity is $O(|M|^k)$ for all queries in $\mathcal{Q}$. $|M|$ denotes the size of the model $M$, whose definition is specific to each model. This means intractability might come from two directions: either a given query cannot be computed exactly, or its complexity is greater than polynomial.

| | EVI | MAR | CON | SAMP |
|---|---|---|---|---|
| Variational Autoencoder (VAE) [37] | | | | ✓ |
| Generative Adversarial Network (GAN) [35] | | | | ✓ |
| RealNVP Flow [38] | ✓ | | | ✓ |
| Sum-Product-Network (SPN) [39] | ✓ | ✓ | ✓ | ✓ |

Table 2.1: Tractability table. ✓symbolizes the tractability of a given query (column) for a given model (row).

## 2.3 Prior art

A great amount of density estimation/generative models rely on the notion of latent variables $\mathbf{Z}$ with realizations $\mathbf{z} \in \mathcal{Z}$. They assume that each observation $\mathbf{x}$ has some unknown $\mathbf{z}$ associated with it and the generative process of $\mathbf{x}$ follows as

$$\mathbf{z} \sim p(\mathbf{z}) \,, \tag{2.13}$$

$$\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z}) \,, \tag{2.14}$$

where $p(\mathbf{z})$ is a (usually simple) prior density function over latent variables $\mathbf{z}$ and $p_\theta(\mathbf{x}|\mathbf{z})$ is a density of observations, conditioned on the latent variable. The distributions $p_\theta(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{z})$ are specifically designed by the concrete model.

### Mixture models

Mixture models are models with discrete latent variable $z \in \mathcal{Z} = \{1, \ldots, K\}$. The latent variables $z \sim p(z) = \mathrm{Cat}(z|\pi)$ are considered categorically distributed with parameters $\pi = (\pi_1, \ldots, \pi_K)^T$ as $z$ is $K$-valued categorical variable in this case. $p(\mathbf{x}|z)$ is called mixture component. Common choices for $p(\mathbf{x}|z)$ are distributions from the exponential family. A well-known example is the Gaussian Mixture Model (GMM), with mixture components being Gaussian distributions $p(\mathbf{x}|z) = \mathcal{N}(\mathbf{x}|\mu_z, \Sigma_z)$.

The hierarchical generative process $\mathbf{x} \sim p(\mathbf{x}|z)$ where $z \sim p(z)$ defines probability distribution of $\mathbf{x}$

$$p(\mathbf{x}) = \sum_{z \in \mathcal{Z}} p(\mathbf{x}, z) = \sum_{z \in \mathcal{Z}} p(\mathbf{x}|z)p(z) = \sum_{z=1}^{K} \pi_z \mathcal{N}(\mathbf{x}|\mu_z, \Sigma_z) \,, \tag{2.15}$$

with the last equation 2.15 being specific to GMM. In the case of $p(\mathbf{x}|z)$ being a proper density that can be evaluated, the computation of $p(\mathbf{x})$ is straightforward due to $z$ attaining only a

finite number of values ($K$ specifically). From the Bayes' rule, this allows easy evaluation of latent variable posterior distribution $p(z|\mathbf{x})$.

Similarly to neural networks being universal function approximations, the Gaussian Mixture Model is an asymptotical universal density approximator [40]. This means that as $K \to \infty$, GMM can approximate arbitrary probability density function. However, from a practical point of view, obtaining reasonable density approximation may require a high number of components $K$, making the GMM model computationally expensive.

### Variational Autoencoders

Variational Autoencoder (VAE) [37] assumes $d$-dimensional continous latent variable $z \in \mathcal{Z} \subseteq \mathbb{R}^d$. VAE chooses to model $p(\mathbf{x}|\mathbf{z})$ as multivariate normal distribution called decoder $p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x} \,|\, f_\theta(\mathbf{z}), \sigma^2 I)$, where $f_\theta : \mathcal{Z} \to \mathcal{X}$ is a neural network with parameters $\theta$, $I$ is an identity matrix of corresponding dimensions and $\sigma^2$ is variance, usually treated as hyperparamater. The latent prior distribution is chosen to be isotropic Gaussian distribution $p(\mathbf{z}) = \mathcal{N}(\mathbf{z} \,|\, \mathbf{0}, I)$. Again, generative process $\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$ where $\mathbf{z} \sim p(\mathbf{z})$ defines distribution of $x$ as follows:

$$p(\mathbf{x}) = \int_{\mathcal{Z}} p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int_{\mathcal{Z}} p_\theta(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) d\mathbf{z} \,. \tag{2.16}$$

The density $p(\mathbf{x})$is, however, in general, not tractable to compute in closed form, usually due to the high dimensionality of $\mathbf{z}$ and the highly nonlinear relationship between $\mathbf{z}$ and $\mathbf{x}$ given by $f_\theta$. It follows from Bayes's rule that the latent variable posterior distribution $p(\mathbf{z}|\mathbf{x})$ is then also intractable. VAE solves this problem by introducing variational distribution $q(\mathbf{z}|\mathbf{x})$ in the hope of approximating the true $p(\mathbf{z}|\mathbf{x})$. The variational distribution $q(\mathbf{z}|\mathbf{x})$ is also called decoder and is chosen to be $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z} \,|\, \mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x}))$, meaning Gaussian distribution with mean $\mu_\phi(\mathbf{x})$ and covariance $\Sigma_\phi(\mathbf{x})$ for a given $\mathbf{x}$.

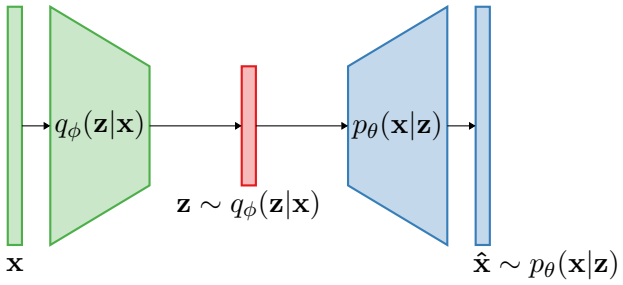



Figure 2.3: VAE model diagram.

The intractability of $p(\mathbf{x})$ makes it difficult to apply maximum likelihood directly. So, instead, the evidence lower bound (ELBO) $\mathcal{L}(x)$ is derived to be the objective of optimization.

$$\log p(\mathbf{x}) \geq \mathcal{L}(\mathbf{x}) = \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) || p(\mathbf{z})) \,. \tag{2.17}$$

The ELBO $\mathcal{L}(x)$ combines reconstruction loss term with regularization in form KL-divergence between decoder distribution and prior latent distribution. It is shown that maximizing ELBO minimizes $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$, giving us the best approximation of $p(\mathbf{z}|\mathbf{x})$ in the terms of reverse KL-divergence. Since the evaluation of full evidence $p(\mathbf{x})$ is intractable, one can directly

use evidence lower bound as its approximation or use tighter Monte Carlo estimate (derived in [37]) for inference tasks involving full evidence evaluation. The work IWAE [41] further investigates tighter log-likelihood lower bounds derived from the importance weighting.

### Normalizing flows

Normalizing flow relates the density of prior latent representation $p_{\mathbf{Z}}(\mathbf{z})$ to the density $p_{\mathbf{X}}(\mathbf{x})$ of a real sample $\mathbf{x}$ using bijective function $g = f^{-1}$, giving $\mathbf{x} = g(\mathbf{z})$ $\big($or $\mathbf{x} = f^{-1}(\mathbf{z})\big)$. Generative process $\mathbf{x} = g(\mathbf{z})$ where $\mathbf{z} \sim p(\mathbf{z})$ is deterministic given a $\mathbf{z}$ and density $p_{\mathbf{Z}}(\mathbf{x})$ is induced by the change of variables formula. All normalizing flow models use this general idea and vary in how they construct the mappings $g : \mathcal{Z} \to \mathcal{X}$ or $f : \mathcal{X} \to \mathcal{Z}$, which are bijective neural networks.

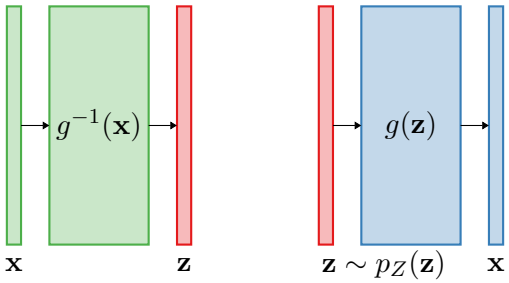



Figure 2.4: Normalizing flow model diagram.

**Change of variables formula** Given realizations of random vectors $\mathbf{x} \in \mathbf{X}$, $\mathbf{z} \in \mathbf{Z}$, known probability density function $p_{\mathbf{Z}}$ of $\mathbf{Z}$ and bijective differentiable mapping $g(\mathbf{Z}) = \mathbf{X}$ with differentiable inverse, then

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}\left(g^{-1}(\mathbf{x})\right) \left| \det \frac{\partial g^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right| . \tag{2.18}$$

The bijective function $g$ is commonly defined in such a form that Jacobian of $g^{-1}$ is easy and efficient to compute. In practice, this means Jacobian of $g^{-1}$ being diagonal, block-diagonal or triangular.

Assume $\mathbf{x} \in \mathcal{X} = \mathbb{R}^D$, the Real NVP flow [38] uses a class of functions $f(\mathbf{x}) = \mathbf{z}$ called affine coupling layer [42] defined as follows

$$\mathbf{z}_{1:d} = \mathbf{x}_{1:d} \tag{2.19}$$

$$\mathbf{z}_{d+1:D} = \exp\left(s(\mathbf{x}_{1:d})\right) \odot \mathbf{x}_{d+1:D} + t\left(\mathbf{x}_{1:d}\right) \tag{2.20}$$

where $\mathbf{z}_{a:b} = (z_a, z_{a+1}, \ldots, z_{b-1}, z_b)^T$, for $a, b \in \mathbb{N}$, $b > a$, $\odot$ is elementwise product and $s$ and $t$ are arbitrary functions $\mathbb{R}^d \to \mathbb{R}^{D-d}$. The parameters of the functions $s$ and $t$ are estimated by maximum likelihood estimation. The loglikelihood reads

$$\log p_{\mathbf{X}}(\mathbf{x}) = \log p_{\mathbf{Z}}\left(f(\mathbf{x})\right) + \log \left| \det \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right| , \tag{2.21}$$

where the second term can be simplified into

$$\log \left| \det \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right| = \sum_{j=d+1}^{D} s(\mathbf{x}_{1:d})_j, \tag{2.22}$$

which follows from the Jacobian of $f(\mathbf{x})$

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \mathbf{I} & \mathbf{O} \\ \frac{\partial \mathbf{z}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \operatorname{diag}(\exp(s(\mathbf{x}_{1:d}))) \end{pmatrix}, \tag{2.23}$$

where **I O** are identity and zero matrices of adequate dimensions, respectively.

## 2.4 Sum-Product Networks

Since this thesis builds upon the concept of Sum-Product Networks (SPNs), we provide their definition in this section. SPNs are deep tractable models. The term "deepness" arises from the structural similarity of SPNs to deep neural networks. We start by introducing a slightly more general class of models called probabilistic circuits (PCs). To wrap up the primary definitions, we formally define SPNs as a special case within the framework of PCs and demonstrate their benefits and effectiveness in solving selected probabilistic inference tasks.

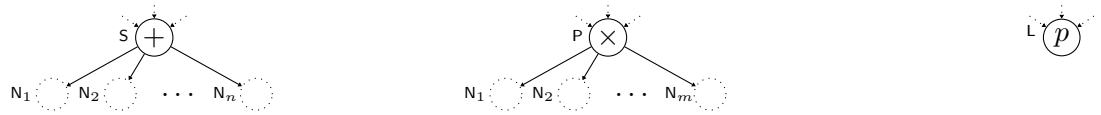
The *computational graph* contains leaf, sum, and product nodes and describes how they are connected and how to proceed in the evaluation of the density. Each node of the computational graph represents a probability density over some subset of random variables $\mathbf{X}$, where the specific subsets are indicated by a *scope function*. Intuitively, PCs represent a probability density over a set of random variables $\mathbf{X}$. The density is computed as a recursive propagation of some elementary probability densities (leaf nodes) using their mixtures (sum nodes) and independent products (product nodes). We resort to more detailed and rigorous definitions of PCs and SPNs in the following text.

### 2.4.1 Definitions

Before defining computational graphs, we begin with a broader insight into graphs and notatin we use. Generally, a graph $G$ constitutes a set of vertices $V$ (also called nodes) and a set of edges $E$. The edges for the undirected graph reads $E \subseteq \binom{V}{2}$ and for the directed $E \subseteq V \times V$. Mainly, we will be dealing with directed graphs, which means that for two nodes $u, v \in V$, we distinguish between $(u, v) \in E$, which symbolizes edge leading from node $v$ to node $u$, and $(v, u) \in E$, which symbolizes edge leading from node $u$ to node $v$. We define more detailed characteristics of the graphs as follows.

- For a node $v \in V$, $\mathbf{pa}(v) = \{u \in V \,|\, (v, u) \in E\}$ denotes the set of parent nodes of node $u$. The root node is a node $v \in V$ for which $\mathbf{pa}(v) = \emptyset$ – in other words, $v$ has no parents.
- For a node $v \in V$, $\mathbf{ch}(v) = \{u \in V \,|\, (u, v) \in E\}$ represents the set of children nodes of node $N$. A leaf node is a node $v \in V$ for which $\mathbf{ch}(v) = \emptyset$ – in other words, $v$ has no children.

**Definition 1** *(Computational graph) The computational graph of a probabilistic circuit denoted as $G$, is a directed acyclic graph, $G = (V, E)$, where $V$ represents a set of computational nodes, and $E \subseteq V \times V$ defines a set of edges connecting these computational nodes. For a node $v \in V$, we define its type $\mathbf{t}(v)$ as $\mathbf{t}(v) \in \{\mathsf{S}, \mathsf{P}, \mathsf{L}\}$, indicating whether a node $v$ is a sum node $\mathsf{S}$, a product node $\mathsf{P}$, or a leaf node $\mathsf{L}$.*




(a) Sum node $\mathsf{S}$ (b) Product node $\mathsf{P}$ (c) Leaf node $\mathsf{L}$

Figure 2.5: Nodes from the computational graph of PC with depicted node types.

**Definition 2** *(Scope function) Given a set of variables $\mathbf{X} = \{X_1, \ldots, X_k\}$, along with the computational graph, $G = (V, E)$, the scope function $\psi$ of $n \in V$ (denoted as $\psi_n$) is a function $\psi : V \to \mathcal{F}(\{1, \ldots, k\})$ that maps the nodes $n \in V$ to the indices of the random variables in $\mathbf{X}$. Furthermore, scope function $\psi$ satisfies following:*

1. *let $v \in V$ be a sum or product node, then $\psi_n = \bigcup_{c \in \boldsymbol{ch}(n)} \psi_c$,*
2. *let $v \in V$ be a root node, then $\psi_v = \{1, \ldots, k\}$.*

The scope function indicates the variables over which node $v$ defines the distribution. A leaf node $v$ represents a user-defined input density function over variables $\mathbf{X}_{\psi_v} \subseteq \mathbf{X}$ and computes density $p(\mathbf{x}_{\psi_v})$. We adopt the same indexing notation as when introducing probabilistic queries in previous chapters. That is that set of indices $m \subseteq \{1, \ldots, d\}$ indexes a $d$-dimensional observation $\mathbf{x} \in \mathcal{X}$ – giving $\mathbf{x}_m$. Similarly, a scope $\psi$ at SPN node $v$, $\psi_v \subseteq \{1, \ldots, d\}$ is used to select corresponding dimensions of $\mathbf{x}$ in the same manner – giving $\mathbf{x}_{\psi_v}$.

The density $p(\mathbf{x})$ of realization of $\mathbf{X}$ is recursively computed by using the computational rules of each node in the computational graph. When traversing the computational graph top down and encountering a product node $v \in V$, we compute:

$$p_v(\mathbf{x}_{\psi_v}) = \prod_{c \in \mathbf{ch}(v)} p_c(\mathbf{x}_{\psi_c}), \tag{2.24}$$

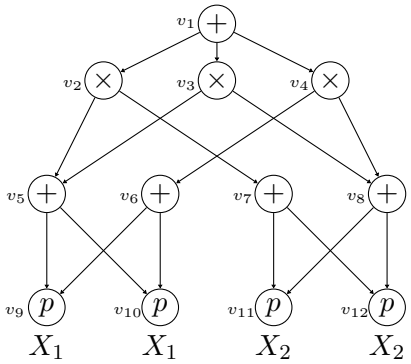
and when encountering a sum node $n \in V$, we calculate:

$$p_v(\mathbf{x}_{\psi_v}) = \sum_{c \in \mathbf{ch}(v)} w_{c,v}\, p_c(\mathbf{x}_{\psi_c}). \tag{2.25}$$

As introduced in the previous equation, each sum node $n \in V$ is associated with one scalar weight $w_{c,v}$ for each of its children $c \in \mathbf{ch}(v)$. All $w_{c,v}$ are non-negative, and for all $v \in V$, it holds that:

$$\sum_{c \in \mathbf{ch}(v)} w_{c,v} = 1, \tag{2.26}$$

meaning that they are locally normalized. The set of all these weights $w$ is denoted as $\theta_{\mathsf{S}}$. Together with a set of parameters for all leaf distributions $\theta_{\mathsf{L}}$, they collectively constitute the parameters $\theta = \theta_{\mathsf{S}} \cup \theta_{\mathsf{L}}$ of a given probabilistic circuit $\mathcal{S} = (G, \psi, \theta)$.




$\mathbf{t}(v) = \mathsf{S}$ for $v \in \{v_1, v_5, v_6, v_7, v_8\}$
$\mathbf{t}(v) = \mathsf{P}$ for $v \in \{v_2, v_3, v_4\}$
$\mathbf{t}(v) = \mathsf{L}$ for $v \in \{v_9, v_{10}, v_{11}, v_{12}\}$
$\psi_{v_9} = \psi_{v_{10}} = \{1\}$
$\psi_{v_{11}} = \psi_{v_{12}} = \{2\}$

Figure 2.6: Example of SPN, representing a distribution over a set of random variables $\mathbf{X} = \{X_1, X_2\}$.

**Definition 3** *(Probabilistic circuit) A probabilistic circuit (PC) with a probability density $p_{\mathbf{X}}$ over a set of random variables $\mathbf{X}$ is a three tuple $\mathcal{S} = (G, \psi, \theta)$, comprising the computational graph $G$, the scope function $\psi$, and the parameters $\theta$.*

**Definition 4** *(Structural constraints) A probabilistic circuit $(G, \theta, \psi)$ is said to be:*

1. ***smooth** if for every Sum Node $n$ it holds that $(\forall\, u, v \in \boldsymbol{ch}(n))(\psi_u = \psi_v)$,*
2. ***decomposable** if for every Product Node $n$ it holds that $(\forall\, u, v \in \boldsymbol{ch}(n))(u \neq v)(\psi_u \cap \psi_v = \emptyset)$.*

**Definition 5** *(Sum-Product Network) A probabilistic circuit satisfying smoothness and decomposability structural constraints is called a Sum-Product Network (SPN).*

The smoothness constraint simplifies equation 2.25 into

$$p_v(\mathbf{x}_{\psi_v}) = \sum_{c \in \mathbf{ch}(v)} w_{c,v}\, p_c(\mathbf{x}_{\psi_v})\ . \tag{2.27}$$

being a convex combination of distribution over the same scope (variables). Effectively, with smoothness, the sum node represents a valid mixture distribution as seen from the introduction to mixture models 2.3. This also implicitly defines a hidden categorical variable associated with each sum node. Peharz et al. noticed the same fact in [43]. Compared to the sum node, decomposability makes a product node implement a product of independent children distributions, creating a factorization of joint distributions.

Not only do structural constraints give a probabilistic circuit clearer semantics and interpretation, but they also simplify the computation of some probabilistic queries and make them tractable [44]. The work of Choi et al. explores additional structural constraints of probabilistic circuits and their impact on the tractability of given queries; refer to [45] for a more detailed description. The next part of the thesis explains how to compute the probabilistic queries introduced in Section 2.2.

### 2.4.2 Inference and tractability in SPNs

In SPNs, we consider leaves to be tractable distributions, concretely being tractable for evidence, marginal and sampling queries. Usually leaf distributions are usually chosen from the exponential family [46, 47, 48].

#### Evidence query

Computing evidence query is straightforward as the computational graph of SPN directly encodes a way to compute a density over the full set of random variables $\mathbf{X}$ at full evidence $\mathbf{x}$. We start at the root node $v$, for which $\mathbf{x} = \mathbf{x}_{\psi_v}$, and we evaluate $p_v(\mathbf{x}_{\psi_v})$ recursively as follows

$$p_v(\mathbf{x}_{\psi_v}) = \begin{cases} \sum_{c \in \mathbf{ch}(v)} w_{c,v}\, p_c(\mathbf{x}_{\psi_v}), & \text{if } \mathbf{t}(v) = \mathsf{S}, \\ \prod_{c \in \mathbf{ch}(v)} p_c(\mathbf{x}_{\psi_c}), & \text{if } \mathbf{t}(v) = \mathsf{P}, \\ p(\mathbf{x}_{\psi_v}), & \text{if } \mathbf{t}(v) = \mathsf{L}. \end{cases} \tag{2.28}$$

In other words, at first, we traverse a computational graph top-down, making use of computational rules for sum and product nodes, essentially spitting scopes, until we reach leaf nodes. Distributions at leaves are evaluated at corresponding parts of observations. Then, we propagate values of leaf distributions using the same sum and product node computational rules until we reach the root node, which, as a result, encodes a value of distribution at $\mathbf{x}$.

For the evaluation, we need to traverse a computational graph top-down, propagating and splitting scopes, where we need to apply recursion $|E|$ times, once for each edge in the computational graph. Then, we assume that corresponding evidence can be evaluated at each leaf with constant complexity $C$, which is a reasonable assumption for many distributions (especially exponential family). Propagating the values of leaf distributions bottom-up to root also takes the complexity of the number of edges $|E|$. This gives the total complexity of $O(2|E| + C|L|)$, where $|L|$ denotes the number of leaf nodes in the computational graph. This leads to complexity $O(|E|)$ – linear complexity in the number of edges in the big O notation since $|L| \leq |E|$. It is important to point out that the size of the edge set may be exponential in some SPN hyperparameters (such as depth); see, for example, [47].

#### Marginal query

Marginal query evaluates a marginal probability distribution. This is performed similarly to a full evidence query, except that variables corresponding to scope $m$ are marginalized. Given a root node $v \in V$, we can say $m \subseteq \psi_v$ as $\mathbf{x} = \mathbf{x}_{\psi_v}$. Marginal query is computed by recursive evaluation of the following

$$\int p_v(\mathbf{x}_{\psi_v}) d\mathbf{x}_m = \begin{cases} \sum_{c \in \mathbf{ch}(v)} w_{c,v} \int p_c(\mathbf{x}_{\psi_v}) d\mathbf{x}_m, & \text{if } \mathbf{t}(v) = \mathsf{S}, \\ \prod_{c \in \mathbf{ch}(v)} \int p_c(\mathbf{x}_{\psi_c}) d\mathbf{x}_{\psi_c \cap m}, & \text{if } \mathbf{t}(v) = \mathsf{P}, \\ \int p(\mathbf{x}_{\psi_v}) d\mathbf{x}_m, & \text{if } \mathbf{t}(v) = \mathsf{L}, \end{cases} \tag{2.29}$$

which essentially says that integration is pushed down to leaf nodes. The intersection of scope $\psi_c$ of product node child $c$ and marginalization scope $m$ reflects the fact that the product node implements scope splitting, and integration needs to be correctly propagated into its children. We further define corner case if $m = \emptyset$ as

$$\int p_v(\mathbf{x}_{\psi_v}) d\mathbf{x}_m \coloneqq p_v(\mathbf{x}_{\psi_v}). \tag{2.30}$$

Also, it is easy to say that if $m = \psi_v$ then

$$\int p_v(\mathbf{x}_{\psi_v}) d\mathbf{x}_m = \int p_v(\mathbf{x}_m) d\mathbf{x}_m = 1. \tag{2.31}$$

In SPNs, for marginal queries to be tractable, we assume that leaves with multivariate distribution can be marginalized in a constant time. When the SPN contains only univariate distributions, marginalization becomes particularly easy – we simply set values of leaf distribution corresponding to marginalized RVs to one. This results in the same complexity $O(|E|)$ as with full evidence query.

#### Conditional query

Computing a conditional query reduces the computation of the ratio of two marginal queries. Given that marginal queries are tractable and have complexity $O(|E|)$, any conditional query can also be easily computed in $O(|E|)$.

#### Sampling query

Sampling in SPNs follows the same philosophy as previously mentioned queries, that is, traversing the computational graph top-down, performing query at the leaf level and traversing the graph back bottom-up. In sampling, this is sometimes referred to as ancestral sampling. It proceeds as follows: if $v \in V$ is sum node, $\mathbf{t}(v) = \mathsf{S}$, then

$$\mathbf{x}_{\psi_v} \sim p_c(\mathbf{x}_{\psi_v}), \quad c \sim \text{Categorical}(c|\mathbf{w}_v), \tag{2.32}$$

where $\mathbf{w}_v$ is a vector of weights corresponding to children on sum node $v$. If $v \in V$ is product node, $\mathbf{t}(v) = \mathsf{P}$, then $\mathbf{x}_{\psi_v}$ is constructed from the samples of children $c \in \mathbf{ch}(v)$

$$\mathbf{x}_{\psi_c} \sim p_c(\mathbf{x}_{\psi_c}) \tag{2.33}$$

as by definition of scope function, $\psi_v = \bigcup_{c \in \mathbf{ch}(v)} \psi_c$ and as by decomposability structural constraint, scopes $\psi_c$ are pairwise disjoint. If $v \in V$ in a leaf, $\mathbf{t}(v) = \mathsf{P}$, then we perform generating random samples as defined for a given distribution.

For tractability of the sampling query, it is crucial to generate samples at leaves with constant complexity. Then computing sampling query also has the complexity of $O(|E|)$.

### 2.4.3 Development of SPNs

Sum-product networks are based on the work of arithmetic circuits [49]. This fact is reflected in the first mentioned work on SPNs [39] by Poon and Domingos, where the notation is similar to that of arithmetic circuits due to the use of indicator functions as leaf distributions. Peharz et al. [44] showed that the leaf distributions can be chosen as arbitrary tractable distributions. The prominence of SPNs was motivated by exact inference for certain queries with linear complexity in the concrete SPN's size.

Research in SPNs features not only learning their parameters given a fixed structure [47, 50] but also joint learning of their structure as well as parameters from data [51, 31, 52, 53, 48]. Structure learning means that the computational graph is expanded during the learning

phase. The intuitive approach presented by LearnSPN [31] relies on recursively partitioning the current scope into approximately independent subsets of variables – inducing a product node and grouping similar observations based on clustering – inducing a sum node. While numerous papers have considered structure learning, more recently, automatic learning algorithms have been slowly abandoned in favour of models with fixed and dense structures such as RAT-SPN [47] and EinSumNet [50]. Peharz et al. showed in [47, 50] that these dense-structure models performed surprisingly well even compared to models with complex structure learning algorithms such as ID-SPN [52].

Advances in scalability and expressiveness have been a significant focus of SPN research. This has brought to light tensorised SPN architectures - the aforementioned RAT-SPN [47] and EinSumNet [50] - which allow SPNs to scale more easily to the tasks performed by state-of-the-art machine learning models. The tensorised architecture makes the similarity of SPNs to neural networks more obvious. However, the field of neural networks is more mature – there are many tools available for neural networks, for example, allowing more accessible training, regularisation, or calibration of classification confidence. The field of SPNs is less advanced, but there has been some focus on it. For example, the works [48, 54] use a Bayesian setting for learning SPN parameters - allowing regularisation by placing prior distributions over the SPN parameters. Dropout in SPNs for the purpose of regularisation has been investigated in [47]. Furthermore, the issue of calibration of SPNs has been addressed by the introduction of tractable dropout [55] as an analytical counterpart to Monte Carlo dropout used in neural networks.

Learning other than homogeneous tabular data has also been discussed in the field of SPNs. The paper Mixed SPN [53] introduces complex distributions over hybrid domains - containing discrete and continuous variables - based on SPN with piecewise polynomial distributions. ABDA [54] tackles the problem of distribution over a hybrid domain by automatically choosing the appropriate distribution (from the exponential family) for given variables.

There have been efforts to link SPNs to existing density estimation models such as normalising flows [56, 57] or variational autoencoders [58]. Pevný et al. introduced the Sum-Product-Transform network - by extending the computational graph of the SPN to include a transformation node that realises a change of the variable formula - effectively combining SPNs and flow models. Tan et al. [58] introduced a hierarchical mixture model over low-dimensional VAE models - effectively creating a combination of SPNs and variational autoencoders. However, these combinations lead to improved expressiveness of SPNs at the cost of losing some tractable queries of SPNs – losing some of the advantages of SPN.

# Chapter 3

# Density estimation of tree-structured data

## 3.1 Tree-structured data

The tree-structure data are nicely defined in the work about HMIL framework [14]; the data are called HMIL samples. The introduction of such samples is motivated by JSON data format. The JSON files consist of some elementary data (vectors or strings), objects (dictionaries or tuples) and lists (sets.)


(a) JSON dictionary.

(b) JSON list.

Figure 3.1: An example of elementary building blocks of JSON data format.

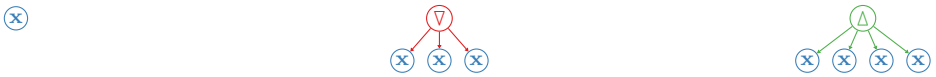
The simplest instances of JSON samples, elementary data, can be seen in Figure 3.1 (a) and (b) in the blue colour. Samples of this form can be described as instances $\mathbf{x}$ of some feature space $\mathcal{X}$, most commonly a subset of $\mathbb{R}^d$. The atomic data refer to the most developed topic of interest in machine learning - it covers a broad scope of possible feature spaces, expressing various modalities. Some examples of these modalities can be boolean values, arbitrary real values or human-readable text. Some concrete examples of feature spaces are $\mathcal{X} = \{0,1\}^d$, $\mathcal{X} = \{0,1,2,\ldots,k\}^d$ for $k \in \mathbb{N}$, $\mathcal{X} = \mathbb{R}^d$ or $\mathcal{X} =$ set of all text strings.

Figure 3.1 (a) shows another more complex example of JSON. Here, the sample consists of the dictionary (tuple of key-value pairs) with several atomic elements data. Such sample is mathematically expressed as ordered tuple $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \in \mathcal{X}_1 \times \mathcal{X}_2 \times \mathcal{X}_3 \times \mathcal{X}_4$, where $\mathcal{X}_i$ represents the feature space for each element of $\mathbf{x}$.

In Figure 3.1 (b), we encounter a different instance of JSON. Here, a sample is a set of JSON objects. The list can alternatively be perceived as a finite set. It can be represented as $b = \{T_1, \ldots, T_k\}$, where $T_i$ are arbitrary JSON samples. In $b$, its elements $T_i$ are not only random, but the cardinality of $b$ itself is also random.

For the purposes of this thesis, we define an alternative formulation of HMIL samples, which we call a leaf-attributed tree.

**Definition 6** *(Leaf-attributed tree) Let $T = (V, E, D)$ be a rooted directed tree with a set of nodes $V$, edges $V$ and attributes $D$. $T$ is equipped with a node type function $\boldsymbol{t}(v) \in \{\mathsf{H}, \mathsf{O}, \mathsf{A}\}$ that tells whether a node $v \in V$ is a heterogeneous node $\mathsf{H}$, homogeneous node $\mathsf{O}$, or an atomic node $\mathsf{A}$. Moreover*



(a) (b) (c)

Figure 3.2: Examples of simple leaf-attributed trees. In (a), the tree represents a vector $\mathbf{x} \in \S$. In (b), the tree is set $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ modelling a JSON list. In (c), the tree is tuple $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_1)$ modelling a JSON object. Here, $\triangle$ is the heterogeneous node (green colour), $\triangledown$ is the homogeneous node (red colour), and $\mathbf{x}$ is the leaf node (blue colour).

1. *if $n \in V$ is an atomic node* $\mathsf{A}$*, then $n$ is attributed with feature $\mathbf{x} = D_n \in \mathcal{X}_n \subseteq \mathbb{R}^{d_n}$, additionally $\boldsymbol{ch}(n) = \emptyset$,*
2. *if $n \in V$ is a homogeneous node* $\mathsf{O}$*, then for all $u, v \in \boldsymbol{ch}(n)$ it holds that $\boldsymbol{t}(u) = \boldsymbol{t}(v)$.*

Heterogeneous nodes $\mathsf{H}$ possess no restrictions on their children. In fact, its children may be an arbitrary leaf-attributed tree, potentially completely different for each child. In the works [1, 7], authors consider edges $(u, v)$, leaving heterogeneous nodes $v$ provided with string labels $k_u$ carrying semantic meaning about the tree rooted at $u$. We will omit this label for the sake of reducing clutter in the visual representation.


```
{
    "ind1": 1, "lumo": -1.246, "inda": 0, "logp": 4.23, "atoms": [
    {
      "element": "c",
        "bonds": [
        {"element": "c", "charge": -0.117, "bond": 7, "atom": 22},
        {"element": "h", "charge":  0.142, "bond": 1, "atom":  3}
      ],
       "charge": -0.117,
         "atom": 22
    },
    ⋮
    {
      "element": "h",
        "bonds": [
        {"element": "c", "charge": -0.117, "bond": 1, "atom": 22}
        ],
       "charge": 0.142,
         "atom": 3
    }
  ]
}
```


(a) (b)

Figure 3.3: (a) An example of Mutagenesis dataset in JSON data format. (b) Leaf-attributed tree, Definition 6, of sample depicted in (a) along with schema (black dashed line), Definition 7, of the given sample.

Definition 6 describes the category of data we will be dealing with, but it is not enough to ensure regularity among children on homogeneous nodes needed for building a model with the ability to effectively generalize to unseen data as described in [1, 14]. To ease the definition of regular data samples, we define the concept of *schema*.

**Definition 7** *(Schema) Given a leaf-attributed tree $T = (V, E, D)$, its schema $S = (V_S, E_S)$ is a subtree of $(V, E)$, meaning that $V_S \subseteq V$ and $E_S \subseteq E$. A subtree $S$ is constructed such by recursively traversing $(V, E)$, beginning at root node $n \in V$, for a homogeneous node $n \in V$, $S$ contains only one (arbitrary) children node of $n$, for a heterogeneous node $n \in V$, $S$ contains all of the children nodes of $v$.*

One leaf-attributed tree can have multiple schemas given by choice of selecting one of the children in homogeneous nodes in the Definition 7 where traversing the tree. Because of that, we further consider only trees satisfying the following regularity assumption.

**Assumption 1** *(Regularity assumption) We consider leaf-attributed trees $T$ that satisfy the following*

1. *$T$ contains only a single root node,*
2. *all schemas of $T$ are the same (isomorphic).*

Figure 3.4 illustrates the difference between a tree satisfying the regularity assumption and not satisfying the assumption.

(a) (b)

Figure 3.4: (a) A tree satisfying the assumption 1, (b) A tree not satisfying the second condition of the Assumption 1.

The main goal of the thesis is to build a notion of density function $p(T)$ over the leaf-attributed trees with fixed schema. In other words, we will estimate $p(T)$ from datasets taking form of $\mathcal{D} = \{T_i \in \mathcal{T} \,|\, i = 1, \ldots, m\}$, where all of the leaf-attributed trees $T_i$ share the **same** schema.

## 3.2 Prior art

### 3.2.1 Set data

In this part, we consider prior art of models for observations of type $\phi = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\} \in \mathcal{F}(\mathcal{X})$. In other words, each observation is a finite set, with possibly variable cardinality $k$. In literature, these observations appear in the context of, for example, multiple instance learning [19], point cloud processing [59] or group anomaly detection [60]. In the context of multiple instance learning, $\phi$ is called a *bag*, and its elements *instances*.

As mentioned in the introduction of the thesis, Pevný et al. introduced a form of neural network [8] for discriminative learning (classification) of multiple instance problems. The main point lies in introducing differentiable and permutation invariant function $f(\phi)$ parametrized by neural networks. This function consists of three elementary building blocks, which are an instance-level transformation $f_I : \mathcal{X} \to \mathbb{R}^m$, elementwise aggregation function $g : \mathcal{F}(\mathbb{R}^m) \to \mathbb{R}^m$ and bag-level transformation $f_B : \mathbb{R}^m \to \mathbb{R}^c$. Altogether, $f(\phi)$ can be written as

$$f(\phi) = f_B(g(\{f_I(\mathbf{x}) \,|\, \mathbf{x} \in \phi\})). \tag{3.1}$$

Please refer to Figure 3.5 for an example. Various aggregation functions are commonly employed, with common choices being elementwise mean and maximum. Notably, Tomczak et al.[61] introduced an approach of using self-attention[62] as an aggregation function. Furthermore, in the model definition, it's possible to stack multiple aggregation functions together,

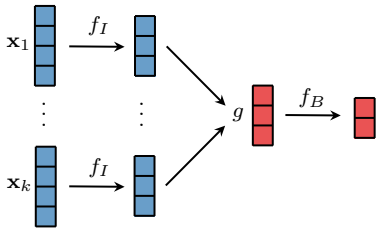



Figure 3.5: An example of *Bag Model* for multiple instance data. The input $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ is a set of $k$ instances $\mathbf{x}_i \in \mathbb{R}^3$. Each instance is transformed by instance transformation $f_I : \mathbb{R}^4 \to \mathbb{R}^3$ and then aggregated elementwise aggregation function $g : \mathcal{F}(\mathbb{R}^3) \to \mathbb{R}^3$. After that, bag transformation $f_B : \mathbb{R}^3 \to \mathbb{R}^2$.

concatenating their outputs. Consequently, this necessitates adjusting the input dimension of the bag-level transformation $f_B$.

The Neural Statistician paper [59] introduces a model that combines neural networks with probabilistic modelling to enable the learning and generation of set data $\phi$. It proposes a framework where a neural network learns to summarize data observations $\phi$ into a fixed-size set of statistics, which are then used to reconstruct the data. The model employs a variational autoencoder (VAE), which allows for generating new data points $\phi$.

Vo et al. [63] introduced a framework for model-based learning from point pattern data based on the random finite set theory [64]. It leverages likelihood functions derived from this theory to extend various learning tasks, such as classification, novelty detection, and clustering. Since Vo et al.'s approach and random finite set theory are crucial for this thesis, we will feature more details in the next section.

Muandet et al. [65] represent each set $\phi$ as empirical distributions, which are then used to build empirical kernels. This approach facilitates an extension of Support vector machines called Support measure machines.

Zhang et al. [66] use Hausdorf distances between sets $\phi_1$ and $\phi_2$ to cluster multiple instance data using the k-medoids algorithm. For non-empty $\phi_1$ and $\phi_2$, maximal Hausdorf distance is defined as

$$d_{H_{\max}}(\phi_1, \phi_2) = \max\{\max_{\mathbf{x}\in\phi_1} \min_{\mathbf{y}\in\phi_2} d(\mathbf{x}, \mathbf{y}), \max_{\mathbf{y}\in\phi_2} \min_{\mathbf{x}\in\phi_1} d(\mathbf{x}, \mathbf{y})\}, \tag{3.2}$$

minimal Hausdorf distance can be written as

$$d_{H_{\min}}(\phi_1, \phi_2) = \min_{\mathbf{y}\in\phi_2} \min_{\mathbf{x}\in\phi_1} d(\mathbf{x}, \mathbf{y}), \tag{3.3}$$

and average Hausdorf distance is defined as

$$d_{H_{\text{ave}}}(\phi_1, \phi_2) = \frac{\sum_{\mathbf{x}\in\phi_1} \min_{\mathbf{y}\in\phi_2} d(\mathbf{x}, \mathbf{y}) + \sum_{\mathbf{y}\in\phi_2} \min_{\mathbf{x}\in\phi_1} d(\mathbf{y}, \mathbf{x})}{|\phi_1| + |\phi_2|}. \tag{3.4}$$

Squared L-2 norm is usually used as distance on distance level [66, 67], $d(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||_2^2$, then unnormalized average Hausdorf distance coincides with the Chamfer distance [67] used in pointcloud processing. Both minimal and average are not metrics in the mathematical sense as they do not satisfy the triangle inequality axiom of metric, see [66].

### 3.2.2 Heterogeneous data

In this part, we consider prior art of models for observations of type $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_k) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_k$. In other words, each observation is a tuple of $k$ elements $\mathbf{x}_i \in \mathcal{X}_i$, where $k$ is fixed. Each element possibly resides in different features space $\mathcal{X}_i$, representing a different modality.

Mandlík [14] tackles this problem in the HMIL framework by defining the so-called *Product Model*. It consists of one function for each element of $\mathbf{x}$ – $f_1, \ldots, f_k$ where $f_i : \mathcal{X}_i \to \mathbb{R}^{m_i}$. Moreover, he defines a function acting on the concatenation of outputs of $f_i$; it is denoted as $f_P : \mathbb{R}^m \to \mathbb{R}^c$ where $m = \sum_{i=1}^{k} m_i$. The whole procedure of applying *Product Model* on $\mathbf{x}$ is written as

$$f(\mathbf{x}) = f_P \left( \Big\|_{i=1}^{k} f_i(\mathbf{x}_i) \right), \tag{3.5}$$

where $\|_{i=1}^{k}$ represents concatenation of $k$ vectors, creating an $m$ dimensional vector. See an example in Figure 3.6.

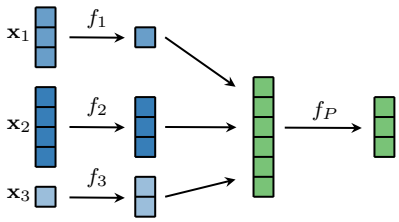



Figure 3.6: An example of *Product Model* for heterogeneous data. The input is a tuple $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ where $\mathbf{x}_1 \in \mathbb{R}^3$, $\mathbf{x}_2 \in \mathbb{R}^4$ and $\mathbf{x}_3 \in \mathbb{R}$. Each element of tuple is transfrom its corresponding function $f_1 : \mathbb{R}^3 \to \mathbb{R}$, $f_2 : \mathbb{R}^4 \to \mathbb{R}^3$ and $f_3 : \mathbb{R} \to \mathbb{R}^2$. The transformed elements of tuples are concatenated together into one vector $\mathbf{x}_c \in \mathbb{R}^6$, and finally, the function $f_P : \mathbb{R}^6 \to \mathbb{R}^3$ is applied.

The probabilistic model for heterogeneous data consists mostly of choosing appropriate base distribution for different models of modality [68, 53, 54].

### 3.2.3 Tree-structured data

HMIL framework is tailored directly for tree-structured data; essentially, it creates a vector embedding of each node by iteration from leaves of the tree and iteratively applying *Bag models* (Figure 3.5) on homogenous nodes and *Product models* (Figure 3.6) on heterogeneous nodes. Similarly, the tree can traversed using TreeLSTM [69] with little tweaking as was shown in [70]. A similar solution, utilizing LSTM models, was proposed by Woof et al. [13]

There has also been research in distance-based approaches for tree-structured data. For example, tree edit distance [71] is a metric used to quantify the dissimilarity or similarity between two tree structures by measuring the minimum cost of transforming one tree into another through a sequence of edit operations. These operations typically include node insertion, deletion, substitution, or relabeling. Šopík et al. [7] developed a version of tree edit distance compatible with HMIL framework [14].

## 3.3 Random finite sets

Let us call $\mathcal{X}$ a feature space, then a realization $\phi$ of a random finite set takes the form of

$$\phi = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\},\ \mathbf{x}_i \in \mathcal{X}, \tag{3.6}$$

where the elements $\mathbf{x}_i$ are random, and the cardinality $\mathsf{card}(\phi) = |\phi| = n$ of $\phi$ is random as well. This gives us an idea that an RFS can be jointly characterized (i) by a cardinality distribution $p(n)$, describing the random nature in the cardinality of $\phi$, and (ii) by a feature distribution $p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ on $\mathcal{X}^n$, describing the randomness of elements of $\phi$ in $\mathcal{X}$ given $\phi$ has cardinality $n$.

Let $\mathcal{F}(\mathcal{X})$ be a set of all subsets of $\mathcal{X}$, also called a powerset. Then, $\phi$ takes values in the power set $\mathcal{F}(\mathcal{X})$, meaning $\phi$ is interpreted as a subset of $\mathcal{X}$. For example, considering $\mathcal{X} = \mathbb{N}$, realizations of random finite sets can look as $\phi_1 = \{2, 3\}$, $\phi_2 = \emptyset$ or $\phi_3 = \{1, 8, 5, 4, 18\}$. Another example with $\mathcal{X} = \mathbb{R}^2$ is shown in Figure 3.7.

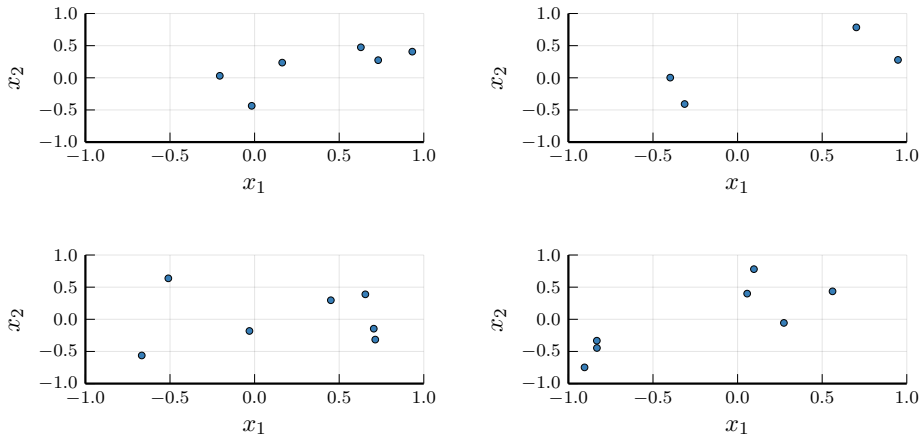



Figure 3.7: Each of four plots depicts one realization of the RFS with the features space $\mathcal{X} = \mathbb{R}^2$. Note that each realization (meaning a set) may contain a different number of elements. The spatial positions of elements also differ among different realizations.

**Random finite set** Similarly to classical random variables, a random finite set is defined as a measurable mapping, $\Gamma : \Omega \to \mathcal{F}(\mathcal{X})$, between measure spaces – namely a probability space $(\Omega, \sigma(\Omega), \mathbb{P})$, and a measurable space $(\mathcal{F}(\mathcal{X}), \sigma(\mathcal{F}(\mathcal{X})), \mu)$. Throughout the rest of the thesis, we will be using the term RFS for the realizations $\Gamma(\omega) \in \mathcal{F}(\mathcal{X})$ for $\omega \in \Omega$ as well as for the random process itself.

**Probability distribution** The probability distribution $P$ of $\mathcal{H} \in \sigma(\mathcal{F}(\mathcal{X}))$ is again defined using the push-forward measure as $P(\mathcal{H}) = \mathbb{P}(\Gamma \in \mathcal{H}) = \mathbb{P} \circ \Gamma^{-1}(\mathcal{H})$, also expressed as $P(\mathcal{H}) = \mathbb{P}(\{\omega \in \Omega \,|\, \Gamma(\omega) \in \mathcal{H}\})$.

**Reference measure** To further investigate the properties of RFSs, we need to build the notion of the reference measure $\mu$ on the measurable space $(\mathcal{F}(\mathcal{X}), \sigma(\mathcal{F}(\mathcal{X})))$. Let $\mathcal{X}^i$ represents $i$-th cartesian product of the feature space $\mathcal{X}$ considering $\mathcal{X}^0 := \emptyset$. Let $\lambda$ be the Lebesgue

measure defined on the measurable space $(\mathcal{X}, \sigma(\mathcal{X}))$. Let $\lambda^i$ be the product Lebesgue measure defined on the product measurable space $(\mathcal{X}^i, \sigma(\mathcal{X}^i))$ for all $i \in \mathbb{N}_0$. Then, the reference measure on $(\mathcal{F}(\mathcal{X}), \sigma(\mathcal{F}(\mathcal{X})))$ is commonly defined [72, 64, 63] as

$$\mu(\mathcal{H}) = \sum_{i=0}^{\infty} \frac{1}{i!U^i} \int_{\mathcal{X}^i} \mathbf{1}_{\mathcal{H}}(\{\mathbf{x}_1, \ldots, \mathbf{x}_i\}) \lambda^i(d\mathbf{x}_1, \ldots, d\mathbf{x}_i) \tag{3.7}$$

for all measurable $\mathcal{H} \subseteq \mathcal{F}(\mathcal{X})$. We consider the units of measurement of $\lambda$ as $\iota$ and of $\lambda^i$ as $\iota^i$. Importantly, given $k, l \in \mathbb{N}$, $K \in \sigma(\mathcal{X}^k)$ and $L \in \sigma(\mathcal{X}^l)$, the sum $\lambda^k(K) + \lambda^l(L)$ is undefined for $k \neq l$ due to the mismatch between the units of measurements. To fix that, the constant $U$ represents the intensity measure of unit volume on $\mathcal{X}$ and has the same units of measurement as $\lambda$. It ensures that the summands in Equation 3.7 are unitless, and the sum is well-defined. In this sense, the resulting reference measure defined in Equation 3.7 is unitless.

The approach to modelling the random finite sets took two paths – one considers the measure-theoretic integral and the standard notion of the probability density as the "modelling" tools [72], and, in contrast, the other adopts the set integral and the belief function [73]. These approaches are shown to be numerically equivalent if $U = 1\iota$ and to differ in the definition of the reference measure [72, 74]. In this work, we consider the first approach (using the measure-theoretic integral and probability density), employing the probability density to align ourselves with the terminology of classical probability densities defined on $\mathbb{R}^n$.

**Measure-theoretic integral** For any measurable function $f : \mathcal{F}(\mathcal{X}) \to \mathbb{R}$ its integral is defined with respect to reference measure $\mu$ (Equation 3.7) as

$$\int_{\mathcal{H}} f(\phi)\mu(d\phi) = \sum_{i=0}^{\infty} \frac{1}{i!U^i} \int_{\mathcal{X}^i} \mathbf{1}_{\mathcal{H}}(\{\mathbf{x}_1, \ldots, \mathbf{x}_i\}) f(\{\mathbf{x}_1, \ldots, \mathbf{x}_i\}) \lambda^i(d\mathbf{x}_1, \ldots, d\mathbf{x}_i) \tag{3.8}$$

for all measurable $\mathcal{H} \subseteq \mathcal{F}(\mathcal{X})$. We will use a shorter notation for the integral over the whole domain $\mathcal{F}(\mathcal{X})$ as follows:

$$\int f(\phi)\mu(d\phi) \coloneqq \int_{\mathcal{F}(\mathcal{X})} f(\phi)\mu(d\phi). \tag{3.9}$$

The introduction of the integral allows the definition of other measures, inevitably leading to the idea of the relationship between the probability distribution and the base measure. Also, as the definition of the integral stems from the definition of the base measure, the reference measure can be straightforwardly recovered from Equation 3.8 as $\mu(\mathcal{H}) = \int_{\mathcal{H}} 1 \cdot \mu(d\phi)$. That is because setting $f(\phi) = 1$ in Equation 3.8 gives directly the reference measure from Equation 3.7.

**Probability density function** The density function of an RFS, $\phi \in \mathcal{F}(\mathcal{X})$, is defined as the Radon-Nikodym derivative of the probability distribution $P$ with respect to the reference measure $\mu$ (Equation 3.7). In other words, it is a function $p : \mathcal{F}(\mathcal{X}) \to \mathbb{R}^+$ satisfying

$$P(\mathcal{H}) = \int_{\mathcal{H}} p(\phi)\mu(d\phi) \tag{3.10}$$

for all measurable $\mathcal{H} \subseteq \mathcal{F}(\mathcal{X})$. For $\phi \in \mathcal{F}(\mathcal{X})$, the quantity $p(\phi)\mu(d\phi)$ represents the probability that the elements of $\phi$ lie in their infinitesimally small surroundings. The probability density of an RFS, $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, with respect to the reference measure $\mu$ can be written [64, 63] as

$$p(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}) = n!U^n p(n) p(\mathbf{x}_1, \ldots, \mathbf{x}_n), \tag{3.11}$$

where $p(n)$ is the cardinality distribution evaluated at $n$, $p(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is the premutation-invariant joint feature distribution evaluated at the elements of the RFS given the cardinality $n$. The quantity $U^n$ ensures that the probability density of the RFS is unitless, and $n!$ accounts for all possible permutations of the set given the cardinality.

It can be easily verified by computing $\int p(\phi)\mu(d\phi)$ that the probability density, as defined in Equation 3.11, is properly normalized.

$$\begin{aligned}
\int p(\phi)\mu(d\phi) &= \sum_{i=0}^{\infty} \frac{1}{i!U^i} \int_{\mathcal{X}^i} p(\{\mathbf{x}_1, \ldots, \mathbf{x}_i\})\lambda^i(d\mathbf{x}_1, \ldots, d\mathbf{x}_i) \\
&= \sum_{i=0}^{\infty} \frac{1}{i!U^i} \int_{\mathcal{X}^i} i!U^i p(i) p(\mathbf{x}_1, \ldots, \mathbf{x}_i)\lambda^i(d\mathbf{x}_1, \ldots, d\mathbf{x}_i) \\
&= \sum_{i=0}^{\infty} p(i) \underbrace{\int_{\mathcal{X}^i} p(\mathbf{x}_1, \ldots, \mathbf{x}_i)\lambda^i(d\mathbf{x}_1, \ldots, d\mathbf{x}_i)}_{=1} \\
&= \sum_{i=0}^{\infty} p(i) \\
&= 1.
\end{aligned}$$

**Expectation** Naturally, introducing the integral and the probability density allows the definition of the expected values. RFSs are a bit different since it is unclear what $\mathbb{E}[\phi]$ should look like. It is because there is no definition of addition and subtraction of sets, and, therefore, we define the function $g : \mathcal{F}(\mathcal{X}) \to \mathbb{R}$, projecting the RFSs to the reals. We can define the expected value of the function $g$ as follows:

$$\mathbb{E}[g(\phi)] = \int g(\phi)p(\phi)\mu(d\phi), \tag{3.12}$$

where $p$ is the probability density of RFSs. Computing the expected values boils down to computing the integrals (as with classical random variables) over $g(\phi)p(\phi)$. We usually consider computing the expectation over the whole domain of the probability density. We can, however, calculate the expected values over the measurable set $\mathcal{H} \subseteq \mathcal{F}(\mathcal{X})$ simply by changing the integrand $g(\phi)p(\phi)$ to $\mathbf{1}_{\mathcal{H}}(\phi)g(\phi)p(\phi)$.

**Cardinality distribution** For $n, k \in \mathbb{Z}$, the Kronecker delta $\delta_n(k)$ is defined as

$$\delta_n(k) = \begin{cases} 1, & \text{if } n = k, \\ 0, & \text{otherwise.} \end{cases} \tag{3.13}$$

Then, the cardinality distribution of an RFS can take the following form:

$$p(|\phi| = n) = p(n) = \mathbb{E}[\delta_n(|\phi|)]. \tag{3.14}$$

Using the probability density of an RFS (Equation 3.11), the relationship between $\mathbb{E}[\delta_n(|\phi|)]$ and $p(n)$ is illustrated by utilising the definition of the expected value

$$\mathbb{E}[\delta_n(|\phi|)] = \sum_{i=0}^{\infty} \frac{1}{i!U^i} \int_{\mathcal{X}^i} \delta_n(i) p(\{\mathbf{x}_1, \ldots, \mathbf{x}_i\}) \lambda^i(d\mathbf{x}_1, \ldots, d\mathbf{x}_i) \tag{3.15}$$

$$= \frac{1}{n!U^n} \int_{\mathcal{X}^n} p(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}) \lambda^n(d\mathbf{x}_1, \ldots, d\mathbf{x}_n) \tag{3.16}$$

$$= \frac{1}{n!U^n} \int_{\mathcal{X}^n} n!U^n p(n) p(\mathbf{x}_1, \ldots, \mathbf{x}_n) \lambda^n(d\mathbf{x}_1, \ldots, d\mathbf{x}_n) \tag{3.17}$$

$$= p(n) \int_{\mathcal{X}^n} p(\mathbf{x}_1, \ldots, \mathbf{x}_n) \lambda^n(d\mathbf{x}_1, \ldots, d\mathbf{x}_n) \tag{3.18}$$

$$= p(n). \tag{3.19}$$

**IID cluster model** Assuming the elements of $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ are independent and identically distributed simplifies the joint feature density as follows: $p(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{i=1}^{n} p(\mathbf{x}_i)$. The class of models that assumes this independence among the elements of an RFS is called the IID cluster model [63], and its probability density can be written as follows:

$$p(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}) = p(n) n! U^n \prod_{i=1}^{n} p(\mathbf{x}_i). \tag{3.20}$$

Some models consider dependencies between the elements but are usually too complex, and the proper normalization of the probability density is intractable. An example of such a model may be the finite Gibbs model [75].

## 3.4 Sum-Product-Set Networks

This thesis proposes a Sum-Product-Set Network (SPSN) model for density estimation of tree-structured data (leaf-attributed trees). SPSNs act as an extension of SPNs from tabular data to structured data. The extension is done by introducing a new node in the computational graph. The inspiration for an extension comes from Sum-Product-Transform Networks (SPTN) [56], which introduced a transformation node for additional expressivity of SPN. In our case, the newly added node, called a set node, has a different purpose – that is, to model homogeneous nodes in the leaf-attributed tree employing the theory of the random finite sets. The following section is concerned with the definition of the Sum-Product-Set Network.

**Definition 8** *(Extended computational graph) The computational graph, denoted as $G$, is a directed acyclic graph, $G = (V, E)$, where $V$ represents a set of computational nodes, and $E \subseteq V \times V$ defines the set of edges connecting these computational nodes. Additionaly, for a node $v \in V$, we define its type $\mathbf{t}(v)$ as $\mathbf{t}(v) \in \{\mathsf{S}, \mathsf{P}, \mathsf{L}, \mathsf{B}\}$, indicating whether node $N$ is a sum node $\mathsf{S}$, a product node $\mathsf{P}$, a set node $\mathsf{B}$, or a leaf node $\mathsf{L}$. A set node $v \in V$ has exactly two children $\mathbf{ch}(v) = \{c_v, f_v\}$, where $c_v$ and $f_v$ are called cardinality and feature children, respectivelly.*

A newly introduced set node computes probability density, given its children, by utilizing the IID cluster model from RFS theory. The choice of cardinality distribution is up to the user's choice.

**Definition 9** *(Extended scope function) Given a set of variables* $\mathbf{X} = \{X_1, \ldots, X_k\}$ *along with computational graph* $G = (V, E)$*. Exteded, scope function* $\psi$ *of* $n \in V$ *denoted as* $\psi_n$ *is a function* $\psi : V \to \mathcal{F}(\{1, \ldots, k\})$ *that maps extended computational graph nodes* $n \in V$ *to indices of random variables in* $\mathbf{X}$*. Furthermore, scope function* $\psi$ *satisfies following:*

1. *let* $v \in V$ *be a sum, product node or set node, then* $\psi_n = \bigcup_{c \in \boldsymbol{ch}(n)} \psi_c$*,*
2. *let* $v \in V$ *be a root node, then* $\psi_v = \{1, \ldots, k\}$*.*

**Definition 10** *(Sum-Product-Set Network) A 3-tuple* $(G, \psi, \theta)$*, where* $G$ *is an extended computational graph,* $\psi$ *is an extended scope function,* $\theta$ *are parameters of computational nodes in* $G$*, and computational graph satisfies decomposability and completeness structural constraints, is called a Sum-Product-Set Network (SPSN).*

**Building SPSN** The construction of SPSN begins by extracting schema from the leaf-attributed tree (Figure 3.8 (a)). The structure of SPSN follows directly from the extracted schema. The simplest SPSN, which we call Naïve SPSN, can be constructed as follows. We traverse the schema starting at the root node and expand the initially empty computational graph $G$ of SPSN recursively as follows. The procedure illustrated in Algorithm 1 takes as an input schema $S$. $E$, $V$ represent nodes and edges of computational graph $G$, respectively.

**Algorithm 1** Construct Naïve SPSN

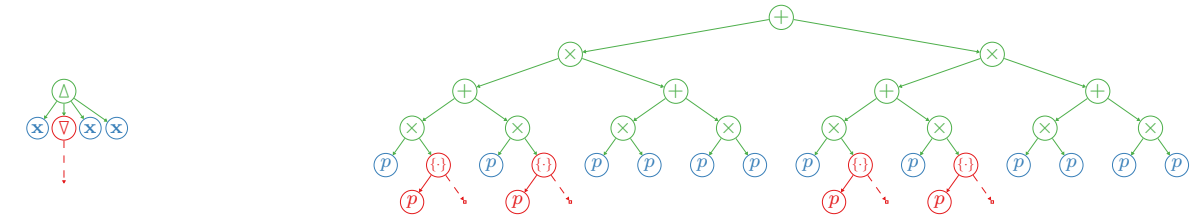
**procedure** `build_SPSN`$(S, E, V, v)$
- $s \leftarrow$ `get_root`$(S)$ ▷ Get root node $s$ from schema $S$
- **if** $\mathbf{t}(s) = \mathsf{H}$ **then** ▷ If $s$ is heterogeneous node
  - $u \leftarrow$ `create_unique_node`
  - $\mathbf{t}(u) \leftarrow \mathsf{P}$ ▷ Create a unique product node $u$
  - $V \leftarrow V \cup \{u\}$
  - $E \leftarrow E \cup \{(u, v)\}$
  - **for all** $c \in \mathbf{ch}(s)$ **do**
    - `build_SPSN`$(S_c, E, V, u)$
  - **end for**
- **else if** $\mathbf{t}(s) = \mathsf{O}$ **then** ▷ If $s$ is homogeneous node
  - $u \leftarrow$ `create_unique_node`
  - $\mathbf{t}(u) \leftarrow \mathsf{B}$ ▷ Create a unique set node $u$
  - $u_c \leftarrow$ `create_unique_node`
  - $\mathbf{t}(u_c) \leftarrow \mathsf{L}$ ▷ Create a unique cardinality leaf node $u_c$
  - $V \leftarrow V \cup \{u, u_c\}$
  - $E \leftarrow E \cup \{(u, v), (u_c, u)\}$
  - `build_SPSN`$(S_{\mathbf{ch}(s)}, E, V, u)$ ▷ Homogeneous node has a single child
- **else if** $\mathbf{t}(s) = \mathsf{A}$ **then** ▷ If $s$ is atomic node
  - $l \leftarrow$ `create_unique_node`
  - $\mathbf{t}(l) \leftarrow (L)$ ▷ Create a unique leaf node $l$
  - $V \leftarrow V \cup \{l\}$
  - $E \leftarrow E \cup \{(l, v)\}$
- **end if**

Naïve SPSN is the simplest form of a probabilistic model of tree-structured data. It assumes the independence of children of heterogeneous nodes. More complex SPSN follow

from Naïve SPSN, but heterogeneous are modelled similarly as in SPNs by alternating between sum nodes and produce nodes. A part of a more complex SPSN can be seen in Figure 3.9.

(a) Tree+schema (b) Naïve SPSN

Figure 3.8: (a) An example of a leaf-attributed tree along with its schema. (b) Naïve SPSN constructed from the schema of the sample in (a)



(a) Schema (b) SPSN

Figure 3.9: An example of a more complex SPSN.

**Inference** Inference in SPSNs is almost the same as in SPNs. Additionally, the newly added set nodes compute the probability density of homogeneous nodes (also interpreted as sets). Assume $v \in V$ is a set node, and $T = \{T_1, \ldots, T_k\}$ is a leaf-attributed tree rooted at the homogeneous node. Then, the set node computes

$$p_v(T) = p_{c_v}(|T|)|T|! \prod_{i=1}^{|T|} p_{f_v}(T_i), \tag{3.21}$$

where $p_{c_v}$ is cardinality distribution of node $v$ and $p_{f_v}$ is feature distribution of node $v$. Similarly, sampling is extended for set node $v \in V$ by first, sampling the cardinality from $k \sim p_{c_v}(|T|)$ and secondly sampling $k$ subtrees from $T_i \sim p_{f_v}(T_i)$. Putting all together results in $T = \{T_1 \ldots T_k\}$. Marginalization follows from the fact that SPNs are a special case of SPSNs. SPNs can be tractably marginalized, and the addition of a set node does not violate this property as marginalization can be propagated from sets to their instances as the instances are assumed to be i.i.d.

# Chapter 4

# Experiments

The title of this thesis reads Sum-Product-Set Networks for Density Learning of Tree-Structured Data. In the previous chapter, we introduced a probabilistic model for density estimation of tree-structured data based on the Sum-Product network. However, to the best of our knowledge, no other probabilistic approach in its whole generality for tree-structured data has not been proposed before. This fact introduces inconvenience – what competing models to choose for comparison as there is a density estimation model for this task. Thus, we resort to comparing tasks that can be solved using SPSN and where prior work exists.

Therefore, in the experiments, we demonstrate the following matters in the context of the goals of the thesis.

1. We show simple examples of SPNs applied to classification problems of observations $\mathbf{x} \in \mathcal{X} = \mathbb{R}^d$. We use the advantages of SPNs to demonstrate their capability to deal effectively with missing values at prediction time.
2. We demonstrate predictive capabilities of SPSNs for random finite sets $\phi \in \mathcal{F}(\mathbb{R}^d)$ on problems of classification and clustering with various feature probability densities.
3. Finally, we show the capabilities of full SPSN on datasets consisting of leaf-attributed trees. The analysis is performed on classification and clustering problems, as well.

In all classification problems, we take the calibration of classifiers into consideration.

## 4.1 Performace metrics

This section serves as a brief introduction to the performance measures used in the thesis. We focus on metrics for classification and clustering tasks as they are the main points of experimental comparison, as stated before.

### Classification

For the evaluation of classification, we assume true labels $Y = (y_1, \dots y_n)$ and predicted labels $\hat{Y} = (\hat{y}_1, \dots, \hat{y}_n)$ for corresponding observations $(\mathbf{x}_1, \dots, \mathbf{x}_n)$. We suppose that $y_i, \hat{y}_i \in \mathcal{Y}$, where $\mathcal{Y} = \{1, \dots, c\}$, meaning we are performing classification to $c$ classes.

**Confusion matrix** A confusion matrix for multiclass classification provides a detailed breakdown of a model's predictions across multiple classes. It is a square matrix where each row corresponds to the true classes, and each column corresponds to the predicted classes. Suppose we have a multiclass classification problem with $c = |\mathcal{Y}|$ classes. We denote a confusion matrix as $C \in \mathbb{N}_0^{c \times c}$. The diagonal elements $C_{ii}$ of the matrix $C$ represent the number of

| | 1 | $\cdots$ | $c$ |
|---|---|---|---|
| 1 | $C_{11}$ | $\cdots$ | $C_{1c}$ |
| $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| $c$ | $C_{c1}$ | $\cdots$ | $C_{cc}$ |

Table 4.1: Confusion matrix for classification into $c$ classes.

instances correctly predicted for class $i$. The entries $C_{ij}$ in column $j$ and row $i$ for $i \neq j$ count the number of instances that belong to class $i$ but were incorrectly classified as class $j$.

The confusion matrix serves as a good summary of classification. However, more commonly used metrics are based on their summarization, such as accuracy or balanced accuracy.

**Accuracy (ACC)** The classical accuracy is one of the most common classification measures. It represents an average number of matches between the model's prediction and the true labels. It can be defined directly on the level of predictions and true labels or using a confusion matrix as follows

$$\mathrm{ACC} = \frac{1}{n}\sum_{i=1}^{n}[\![y_i = \hat{y}_i]\!] = \frac{1}{\sum_{i=1}^{c}\sum_{j=1}^{c} C_{ij}}\sum_{i=1}^{c} C_{ii}. \tag{4.1}$$

The accuracy gives values in the $[0, 1]$ range where 1 represents the perfect classification result.

The standard accuracy is not appropriate when facing a big class imbalance. Image we have $c = 2$ classes, but the first class accounts for 99% of observations and the second only for 1%. Then, one could achieve 99% accuracy by predicting the first class, and it can lead to misleading results. The ideal case is to inspect confusion matrices directly, which is infeasible if comparing a large number of different models or datasets.

**Balanced accuracy (BACC)** Balanced accuracy [76] aims to solve the issue of class imbalance. It does so by computing accuracies for each class separately, and it takes an average of class accuracies. It can be expressed using the confusion matrix as follows:

$$\mathrm{BACC} = \frac{1}{c}\sum_{i=1}^{c}\frac{C_{ii}}{\sum_{j=1}^{c} C_{ij}}. \tag{4.2}$$

Again, BACC gives values in the $[0, 1]$ range where 1 represents the perfect classification result.

However, considering the mentioned $99:1$ class imbalance. When predicting the first class only, balanced accuracy gives the result of 50% compared to 99% of classical accuracy. This makes BACC more suitable for classification in imbalanced datasets. Moreover, when the classes are perfectly balanced, BACC and ACC coincide.

**Other performance metrics** Other performance metrics include other summarizations of confusion matrix; for example, a popular one is Matthews correlation coefficient [77]. Then, in binary classification case ($c = 2$), one can define more metrics that are not based on predicted labels $\hat{Y}$ but on some classification scores $\hat{S} = (\hat{s}_1, \ldots, \hat{s}_k)$, where the prediction label $\hat{y}_i$ is recovered based on thresholding score $\hat{s}_i$ by threshold $\tau$. Receiver characteristic operating (ROC) or precision-recall (PR) curves study the effect of changing $\tau$.

### Clustering

Evaluation of the clustering performance is slightly more challenging than evaluation classification. This is given by the fact that clustering is an unsupervised task and that the number of clusters is mostly regarded as unknown. Since the task is unsupervised, the identified cluster may be correct up to permutation. Consider ground truth labels $Y = (1, 1, 2, 2)$, and clustering result $\hat{Y} = (2, 2, 1, 1)$, in this case, classical accuracy would be 0%. However, swapping labels 1 and 2 produces 100% accuracy. Motivated by this simple example, clustering performance metrics should measure agreement between clustering prediction and true labels considering the label permutation of clustering prediction.

A commonly used measure, called Rand index (RI) [78] is computed by calculating true positives (pairs of points that are in the same cluster in both the true and predicted clusterings) and true negatives (pairs of points that are in different clusters in both the true and predicted clusterings) against all possible pairs of points. RI can be seen as computing accuracy on pairs of points induced by clustering and true labels, which is invariant to renaming clusters (permuting cluster labels). There is a variation of the Rand index, called the Adjusted Rand Index (ARI) [79], which corrects against random clustering assignment. This correction helps prevent the high index values that can occur with a random assignment of clusters (especially with unbalanced true cluster labels).

Another standard measure is the Silhouette coefficient (SC) [80]. It calculates the average distance between data points within the same cluster compared to the distance to points in the nearest neighbouring cluster. The problem with SC is that it is, by its definition, restricted to clustering methods based on distances or spaces where the distance between two observations is defined.

**Unsupervised accuracy (UACC)** This thesis features an extension of supervised accuracy to unsupervised problems. For example, this extension called Unsupervised accuracy (UACC) was used in [81]. It assumes that the number of clusters is the same as the number of ground truth classes. UACC can be defined as

$$\mathrm{UACC} = \max_{\pi} \frac{1}{N} \sum_{j=1}^{n} [\![ y_j = \pi(\hat{y_j}) ]\!] \tag{4.3}$$

where $\pi : \mathcal{Y} \to \mathcal{Y}$ is permutation function of cluster assignments. Evaluation of UACC can be done by enumerating the classical accuracy of all possible $c!$ permutations and choosing the maximal value of those, which can be computationally expensive for a large number of clusters. However, the problem of computing UACC can be more effectively solved by the Hungarian matching algorithm [82] with complexity $O(c^3)$.

### Calibration

A well-calibrated classifier should predict accurate labels and provide probabilities that reliably reflect the true chance of those labels. For instance, if a model predicts a 75% chance of $\mathbf{x}$ being of class $y$, it should actually be class $y$ about 75% of the time among observations with a predicted 75% probability. Calibration is especially crucial in applications where accurate probability estimates are necessary, such as computer security or medical diagnoses.

Naeini et al. [83] define an Expected calibration error (ECE). The ECE quantifies the discrepancy between predicted probabilities and the actual outcomes. It measures the average

difference between predicted probabilities and the true accuracy within different probability bins. A lower ECE signifies better calibration.

Suppose $Y = (y_1, \dots y_n)$ constitute true labels, $\hat{Y} = (\hat{y}_1, \dots, \hat{y}_n)$ are predicted labels and $\hat{P} = (\hat{p}_1, \dots, \hat{p}_n)$ are corresponding prediction confidences. Let $B_m$ be the set of observations whose prediction confidence falls into the interval $(\frac{m-1}{M}, \frac{m}{M}]$ for $m = 1, \dots, M$, where $M$ reads the number of bins $M \in \mathbb{N}$. ECE can be written as

$$\mathrm{ECE} = \frac{1}{n} \sum_{i=m}^{M} |B_m| \cdot |\mathrm{ACC}(B_m) - \mathrm{CONF}(B_m)|, \tag{4.4}$$

where $\mathrm{ACC}(B_m)$ is accuracy of observation in $B_m$ and $\mathrm{CONF}(B_m)$ is average confidence of prediction of observations in $B_m$. ECE gives values in the interval $[0, 1]$, where 0 signifies the perfect calibration.

## 4.2 Datasets

### Tabular data

We use thirteen tabular datasets from the UCI dataset repository [84] for classification purposes. The observations are vectors $\mathbf{x} \in \mathbb{R}^d$. The summary of statics of tabular datasets is shown in Table 4.2.

| name | $d$ | $n_{\mathrm{data}}$ | $c$ |
|---|---|---|---|
| blood | 4 | 747 | 2 |
| breast | 9 | 683 | 2 |
| climate | 18 | 540 | 2 |
| ecoli | 7 | 336 | 8 |
| glass | 9 | 214 | 6 |
| iris | 4 | 150 | 3 |
| leaf | 14 | 340 | 30 |
| olive | 9 | 572 | 3 |
| parkinsons | 22 | 195 | 2 |
| seeds | 7 | 210 | 3 |
| vehicle | 18 | 846 | 4 |
| vertebral_column | 6 | 620 | 4 |
| wine | 13 | 178 | 3 |

Table 4.2: Statistics of tabular datasets. Each row represents one dataset $\mathcal{D}$ with its name, dimension $d$ of is features $\mathbf{x} \in \mathbb{R}^d$, number of observations $n_{\mathrm{data}} = |\mathcal{D}|$ and number of classes $c$.

### Multiple-instance problems

For demonstration purposes of SPSN and the model of random finite sets, we utilise the same twenty datasets used by Pevný et al. [8], which are available at https://github.com/pevnak/MIProblems. These datasets contain observations $\phi \in \mathcal{F}(\mathbb{R}^d)$, where $d$ is instance dimension. The summary of statics of multiple-instance datasets is shown in Table 4.3.

| dataset | $n_{\text{bags}}$ | $n_{\text{inst}}$ | $d$ | $c$ | ratio |
|---|---|---|---|---|---|
| brown_creeper | 548 | 10232 | 38 | 2 | 64 : 36 |
| corel_african | 2000 | 7947 | 9 | 2 | 95 : 5 |
| corel_beach | 2000 | 7947 | 9 | 2 | 95 : 5 |
| elephant | 200 | 1391 | 230 | 2 | 50 : 50 |
| fox | 200 | 1320 | 230 | 2 | 50 : 50 |
| musk_1 | 92 | 476 | 166 | 2 | 49 : 51 |
| musk_2 | 102 | 6598 | 166 | 2 | 62 : 38 |
| mutagenesis_1 | 188 | 10486 | 7 | 2 | 66 : 34 |
| mutagenesis_2 | 42 | 2132 | 7 | 2 | 69 : 31 |
| newsgroups_1 | 100 | 5443 | 200 | 2 | 50 : 50 |
| newsgroups_2 | 100 | 3094 | 200 | 2 | 50 : 50 |
| newsgroups_3 | 100 | 5175 | 200 | 2 | 50 : 50 |
| protein | 193 | 26611 | 9 | 2 | 87 : 13 |
| tiger | 200 | 1220 | 230 | 2 | 50 : 50 |
| ucsb_breast_cancer | 58 | 2002 | 708 | 2 | 55 : 45 |
| web_1 | 75 | 2212 | 5863 | 2 | 72 : 28 |
| web_2 | 75 | 2219 | 6519 | 2 | 76 : 24 |
| web_3 | 75 | 2514 | 6306 | 2 | 81 : 19 |
| web_4 | 75 | 2291 | 6059 | 2 | 73 : 27 |
| winter_wren | 548 | 10232 | 38 | 2 | 80 : 20 |

Table 4.3: Statistics of multiple-instance problems datasets. Each row represents one dataset $\mathcal{D}$ with its name, number of observations (bags) $n_{\text{bags}} = |\mathcal{D}|$, number of instances $n_{\text{inst}} = \sum_{\phi \in \mathcal{D}} |\phi|$, dimension $d$ of instances, number of classes $c$ and the ratio of number class labels.

### Hierarchical multiple-instance problems

Full capabilities of SPSN are used on datasets, which we refer to as Hierarchical multiple-instance problems, following the HMIL paradigm [14]. The datasets come from CTU Relational datasets database [85] and are available at https://relational.fit.cvut.cz/. We downloaded thirteen suitable datasets and saved them in JSON file format. We selected only one-hop neighbourhoods of relational datasets that contain cycles in their database schema. The summary of statics of multiple-instance datasets is shown in Table 4.4. Each JSON dataset has its schema extracted by JsonGrinder.jl library [86].

## 4.3 Models

### Tabular data

Table 4.5 shows models used in the experiments with tabular data.

For classification with missing values using neural network-based classifiers (mlp), we consider a feature vector $\mathbf{x} \in \mathbb{R}^d$, $d \in \mathbb{N}$, with corresponding binary mask $\mathbf{b} \in \{0,1\}^d$, with $b_i = 1$, $i \in \{1, \ldots, D\}$, indicating that feature element $x_i$ at position $i$ is missing. Missing values are tackled by imputation used in [14]. Imputed feature vector $\tilde{\mathbf{x}}$ element $\tilde{x}_i$ reads

| name | $n_{\text{data}}$ | $c$ | $n_{\mathsf{O}}$ | $n_{\mathsf{H}}$ | $n_{\mathsf{A}}$ | $\bar{n}_{\mathsf{O}}$ | $\bar{n}_{\mathsf{H}}$ | $\bar{n}_{\mathsf{A}}$ |
|---|---|---|---|---|---|---|---|---|
| mutagenesis | 188 | 2 | 5081 | 15567 | 57375 | 27 | 83 | 305 |
| genes | 862 | 15 | 3544 | 17941 | 125712 | 4 | 21 | 146 |
| cora | 2708 | 7 | 16274 | 13566 | 247663 | 6 | 5 | 91 |
| citeseer | 3312 | 6 | 16071 | 12759 | 411929 | 5 | 4 | 124 |
| webkp | 877 | 5 | 4970 | 4093 | 396890 | 6 | 5 | 453 |
| world | 239 | 7 | 478 | 5302 | 18296 | 2 | 22 | 77 |
| chess | 295 | 3 | 10325 | 590 | 53593 | 35 | 2 | 182 |
| uw_cse | 278 | 4 | 782 | 782 | 3128 | 3 | 3 | 11 |
| hepatitis | 500 | 2 | 1500 | 7008 | 65039 | 3 | 14 | 130 |
| pubmed_diabetes | 19717 | 3 | 236503 | 108393 | 10896008 | 12 | 5 | 553 |
| ftp | 30000 | 3 | 30000 | 96491 | 392455 | 1 | 3 | 13 |
| ptc | 343 | 2 | 18953 | 55279 | 128184 | 55 | 161 | 374 |
| premier_league | 380 | 3 | 760 | 10749 | 2084929 | 2 | 28 | 5487 |

Table 4.4: Statistics of hierarchical multiple-instance problems datasets. Each row represents one dataset $\mathcal{D}$ with its name, number of observations $n_{\text{data}} = |\mathcal{D}|$ and the number of classes $c$. The quantities $n_{\mathsf{O}}$, $n_{\mathsf{H}}$ and $n_{\mathsf{A}}$ represent the number of homogeneous, heterogeneous and atomic nodes in the entire dataset, respectively. $\bar{n}_{\mathsf{O}}$, $\bar{n}_{\mathsf{H}}$ are $\bar{n}_{\mathsf{A}}$ the corresponding average numbers of nodes per one observation.

| name | description | hyperparameters |
|---|---|---|
| ratspn | RAT-SPN [47] | Table A.1 |
| mlp | Multi-layer perceptron | Table A.2 |
| mlp+mcd | Multi-layer perceptron with Monte Carlo dropout [87] | Table A.2 + (MCD rate 0.2) |
| lin | Perceptron | none |

Table 4.5

$\tilde{x}_i = (1 - b_i)x_i + b_i\psi_i$, where $\psi_i$ is an element of static (i.e., the same for all $\mathbf{x}$) but learnable imputing vector $\psi \in \mathbb{R}^d$.

- *mlp 5* reads Multilayer perceptron trained with randomly sampled missing values (5% for each batch) at the input, $b_i \sim \text{Bernoulli}(0.05)$,
- *mlp uniform* reads Multilayer perceptron trained with randomly sampled missing values (uniformly sampled missing rate for each batch) at the input, $b_i \sim \text{Bernoulli}(p)$, $p \sim \mathcal{U}(0, 0.9)$.

### Multiple instance problems

#### Hierarchical multiple instance problems

## 4.4 Experimental setup

In all experiments, consider splits of the dataset in three parts - training, validation and testing dataset, where 64% of observation of the original dataset accounts for the training part,

| name | description | hyperparameters |
|---|---|---|
| spsn+gmm | SPSN with GMM leaf distribution | Table A.3 |
| spsn+spn | SPSN with SPN leaf distribution | Table A.5 |
| spsn+relanvp | SPSN with RealNVP flow leaf distribution | Table A.4 |
| hmil | HMIL [14] | Table A.6 |
| hmil+mcd | HMIL [14] with Monte Carlo dropout [87] | Table A.6 (+ MCD rate 0.2) |

Table 4.6: Classification methods used on multiple instance datasets.

| name | description | hyperparameters |
|---|---|---|
| spsn+gmm | SPSN with GMM leaf distribution | Table A.3 |
| spsn+relanvp | SPSN with RealNVP flow leaf distribution | Table A.4 |
| k-medoids | k-medoids with Hausdorf distance [66] | $\{d_{H_{\text{ave}}}, d_{H_{\text{max}}}, d_{H_{\text{min}}}\}$ |

Table 4.7: Clustering methods used on multiple instance datasets.

16% for validation and 20% is left out for testing. All datasets are split using random stratified partitioning, meaning the class distribution ratio is the same across training, validation, and testing. The splits are controlled by fixed random seeds for easier reproduction of the results. The best configuration of hyperparameters is selected using 5-fold cross-validation.

We implemented models containing keywords *spsn*, *gmm*, *spn*, *realnvp* ourselves. For *ratspn*, we used the original implementation available at https://github.com/cambridge-mlg/RAT-SPN. For the *hmil* model, we used the implementation available at https://github.com/CTUAvastLab/Mill.jl. BAMIC [66] version of *k-medoids* is also our implementation, along with variations of Hausdorf distances. Lastly, authors of tree edit distance [7] provided us with its implementation, which cannot be openly shared.

The experimental pipeline and all methods implemented by ourselves are coded in the Julia programming language [88]. All experiments are run on a computational cluster with 46 × Intel Xeon Scalable Gold 6146 CPUs and scheduled using Slurm job manager. One Slurm job consists of running one model on one dataset and for one hyperparameter setup. The run time of one job was restricted to 1 day of computation time with computational resources of 1 CPU core and 16 GB RAM.

| name | description | hyperparameters |
|---|---|---|
| spsn | SPSN | Table A.7 |
| hmil | HMIL | Table A.8 |

Table 4.8: Classification methods used on hierarchical multiple instance datasets.

| name | description | hyperparameters |
|---|---|---|
| spsn | SPSN | Table A.7 |
| k-medoids | k-medoids with tree edit distance [7] | none |

Table 4.9: Clustering methods used on hierarchical multiple instance datasets.

# Chapter 5

# Results

For a more convenient comparison of methods presented in the result tables, we deploy the average ranking [89] on each dataset (lower is better).

## 5.1 Tabular data

The experiments performed on tabular data serve as a check that SPN models can be seen as competitive approaches to classification compared to more expressive models based on neural networks. As can be seen in Table 5.1, *spsn* is on par with *mlp*. Both of these models, however, fall short in comparison to *mlp+mcd*, which improves the classification and calibration results. It has to be mentioned that the difference between the methods on some datasets is slight and mostly falls within a standard deviation of presented results. Further significance testing would have to be performed for a more precise conclusion. The small difference in results can also be accounted for by the small sizes of the datasets and their simplicity, illustrated by the fact that the *lin* model is consistently relatively competitive to other methods.

Probabilistic models are intuitively believed to be better calibrated than their non-probabilistic counterparts. This idea was disproven in [55] and is supported by the results of our experiments as shown in Table 5.2. The mentioned results might lower the anticipation of SPSN's excellent performance.

In terms of robustness to missing values, *mlp 5* seems to be inferior *mlp uniform*, which was to be expected because *mlp uniform* learns to impute the missing values better. However, *rat-spn* performs as good or better, except iris, ecoli or parkinsons datasets. This is a good result, given that *rat-spn* can naturally handle missing data (using marginalization) without additional tweaking.

| dataset | mlp | ratspn | linear | mlp+mcd |
|---|---|---|---|---|
| blood | **0.79±0.03** | **0.79±0.04** | 0.76±0.04 | **0.79±0.04** |
| breast | 0.95±0.02 | **0.97±0.02** | 0.96±0.02 | **0.97±0.02** |
| climate | 0.96±0.02 | **0.97±0.01** | 0.96±0.01 | 0.96±0.02 |
| ecoli | **0.87±0.03** | 0.85±0.02 | 0.86±0.03 | 0.86±0.05 |
| glass | 0.71±0.07 | 0.70±0.05 | 0.60±0.04 | **0.73±0.08** |
| iris | **0.96±0.03** | 0.89±0.05 | 0.87±0.02 | **0.96±0.03** |
| leaf | 0.70±0.07 | 0.55±0.07 | 0.64±0.06 | **0.75±0.06** |
| olive | **1.00±0.00** | **1.00±0.00** | **1.00±0.00** | **1.00±0.00** |
| parkinsons | 0.91±0.02 | **0.93±0.04** | 0.84±0.03 | 0.91±0.04 |
| seeds | 0.93±0.02 | **0.95±0.02** | 0.93±0.03 | 0.94±0.03 |
| vehicle | 0.82±0.03 | 0.76±0.03 | 0.75±0.03 | **0.83±0.02** |
| vertebral_column | 0.53±0.04 | **0.57±0.07** | 0.53±0.05 | 0.54±0.02 |
| wine | **0.97±0.05** | **0.97±0.05** | **0.97±0.05** | **0.97±0.02** |
| rank | 1.92 | 1.92 | 2.92 | **1.38** |

Table 5.1: Testing dataset accuracy (ACC) of classification tabular data. The best model is chosen according to the best ACC achieved on the validation dataset.

| dataset | mlp | ratspn | linear | mlp+mcd |
|---|---|---|---|---|
| blood | 0.06±0.01 | **0.05±0.01** | 0.06±0.02 | **0.05±0.02** |
| breast | 0.04±0.02 | **0.02±0.01** | **0.02±0.01** | 0.03±0.02 |
| climate | **0.04±0.01** | **0.04±0.01** | 0.06±0.02 | **0.04±0.02** |
| ecoli | **0.07±0.03** | 0.13±0.01 | 0.10±0.02 | 0.08±0.03 |
| glass | 0.26±0.06 | 0.21±0.06 | 0.15±0.04 | **0.12±0.06** |
| iris | **0.04±0.02** | 0.25±0.04 | 0.15±0.02 | **0.04±0.01** |
| leaf | 0.20±0.06 | 0.16±0.05 | 0.21±0.05 | **0.12±0.05** |
| olive | **0.00±0.00** | 0.01±0.00 | 0.02±0.00 | **0.00±0.00** |
| parkinsons | **0.08±0.02** | 0.10±0.01 | 0.11±0.03 | **0.08±0.04** |
| seeds | 0.07±0.02 | 0.12±0.02 | 0.12±0.02 | **0.05±0.02** |
| vehicle | 0.09±0.05 | **0.06±0.01** | **0.06±0.03** | 0.10±0.02 |
| vertebral_column | 0.12±0.04 | 0.10±0.03 | 0.13±0.03 | **0.08±0.03** |
| wine | 0.05±0.02 | 0.12±0.01 | 0.06±0.01 | **0.03±0.02** |
| rank | 2.23 | 2.46 | 3.00 | **1.46** |

Table 5.2: Testing dataset expected calibration error (ECE) of classification tabular data. The best model is chosen according to the best ACC achieved on the validation dataset.

## 5.2 Multiple-instance problems

### Classification

The experiments performed on multiple instance data present a step up from tabular data and aim to test the probabilistic model of random finite sets. From Table 5.3, *spsn+spn* shows a promise compared to the established baseline model *hmil*. As with tabular data, the combination of *hmil* with Monte Carlo dropout generally yields the best result in BACC, which illustrates the strength of this approach, which was, however, redemption by higher compu-

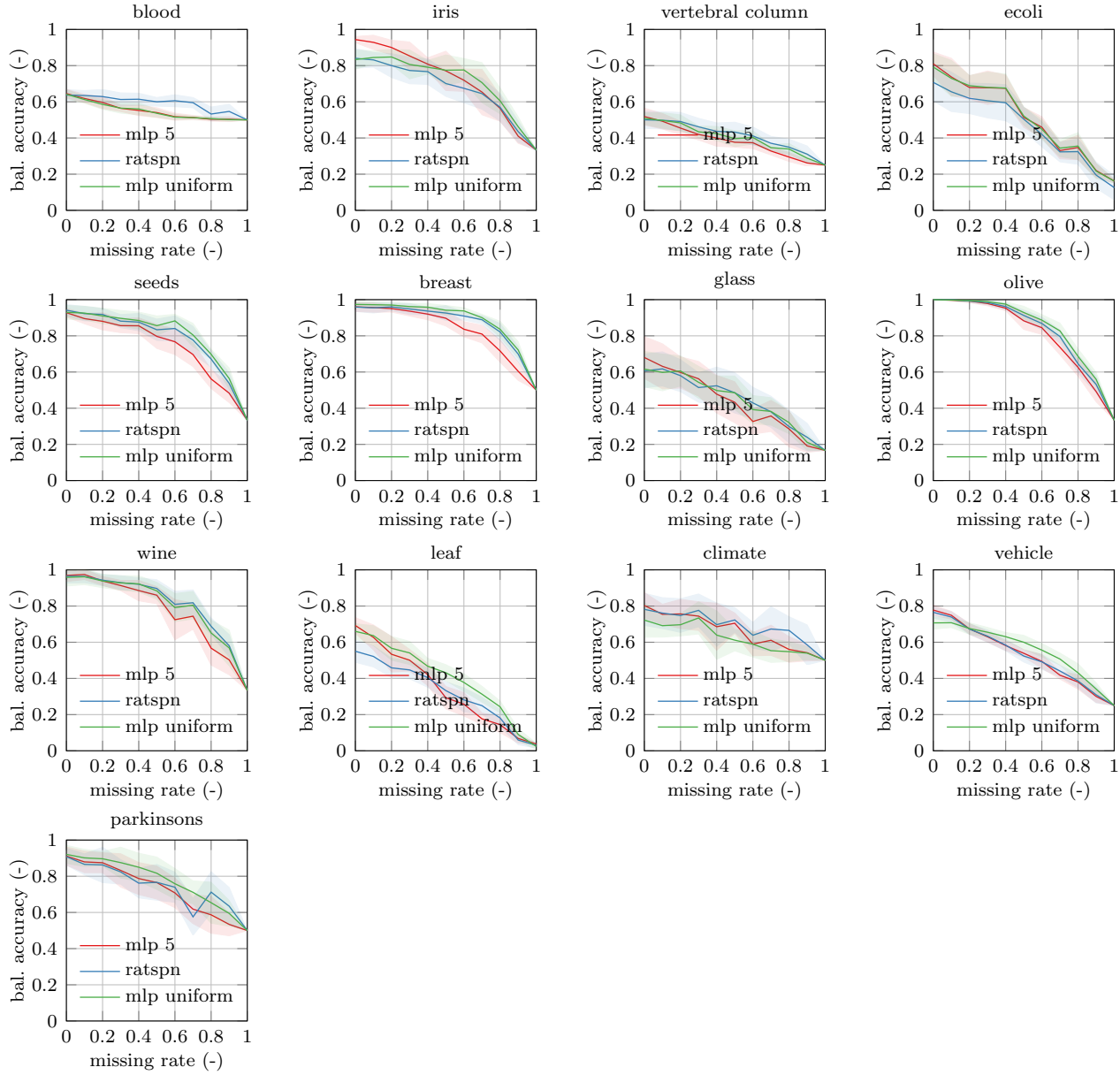



Figure 5.1: Dependency of testing dataset balanced accuracy (BACC) of classification tabular data with changing ratio of missing values at prediction time. The best model is chosen according to the best BACC achieved on the validation dataset (at 0% missing rate), and the same model is used to evaluate data with different ratios of missing values.

tational cost and slower convergence. Of three *spsn* approaches, *spsn+realnvp* performed the worst. We believe the reason is the usual overfitting of normalizing flow models, which we observed during the experiments, even though we tried to implicitly regularize the flow by reducing its size. Contrary to that, *spsn+gmm* is believed to perform poorly due to a lack of expressivity. Arguments for interpretation of calibration error results follow the ones used for BACC, except that *spsn+spn* was better calibrated than the baseline *hmil* in this case.

Additionally, the ablation study of covariance type of *spsn+gmm*. Authors of [47] stated

that choosing the unit matrix as the covariance matrix of Gaussian distribution gave the best result. This was not proven in our case, and the diagonal covariance matrix yielded the best results in terms of both BACC and ECE. Not even the regularization of the covariance matrix helped to achieve better or more stable results. See tables 5.5 and 5.6.

| dataset | spsn+gmm | spsn+spn | spsn+realnvp | hmil | hmil+mcd |
|---|---|---|---|---|---|
| brown_creeper | **0.94±0.01** | **0.94±0.02** | 0.93±0.01 | **0.94±0.02** | **0.94±0.01** |
| corel_african | 0.78±0.07 | 0.79±0.05 | 0.78±0.06 | 0.78±0.05 | **0.81±0.04** |
| corel_beach | 0.88±0.05 | 0.88±0.06 | 0.90±0.03 | 0.91±0.04 | **0.93±0.06** |
| elephant | 0.82±0.04 | 0.83±0.06 | 0.74±0.08 | 0.83±0.05 | **0.85±0.04** |
| fox | 0.54±0.15 | 0.59±0.10 | 0.48±0.04 | 0.55±0.15 | **0.60±0.11** |
| musk_1 | 0.80±0.13 | 0.84±0.09 | **0.89±0.06** | 0.82±0.14 | 0.88±0.10 |
| musk_2 | **0.85±0.06** | 0.75±0.08 | 0.78±0.07 | 0.79±0.09 | 0.83±0.06 |
| mutagenesis_1 | **0.87±0.07** | 0.83±0.08 | 0.83±0.04 | 0.80±0.06 | 0.81±0.07 |
| mutagenesis_2 | 0.66±0.23 | 0.69±0.20 | **0.78±0.10** | 0.69±0.10 | 0.60±0.20 |
| newsgroups_1 | 0.77±0.08 | **0.81±0.10** | 0.68±0.12 | 0.78±0.08 | 0.77±0.09 |
| newsgroups_2 | 0.57±0.19 | 0.53±0.10 | 0.60±0.11 | 0.65±0.08 | **0.66±0.11** |
| newsgroups_3 | **0.71±0.09** | 0.65±0.07 | 0.58±0.06 | 0.67±0.14 | 0.65±0.05 |
| protein | 0.64±0.10 | 0.64±0.13 | 0.65±0.09 | 0.67±0.09 | **0.75±0.08** |
| tiger | 0.75±0.04 | **0.77±0.05** | 0.67±0.03 | **0.77±0.13** | **0.77±0.03** |
| ucsb_breast_cancer | 0.75±0.11 | 0.79±0.06 | 0.79±0.04 | **0.82±0.10** | 0.75±0.11 |
| web_1 | 0.65±0.12 | **0.82±0.07** | 0.66±0.12 | 0.66±0.10 | 0.62±0.10 |
| web_2 | 0.49±0.18 | **0.86±0.14** | 0.38±0.07 | 0.53±0.17 | 0.43±0.05 |
| web_3 | 0.53±0.09 | **0.97±0.03** | 0.62±0.14 | 0.55±0.15 | 0.55±0.11 |
| web_4 | 0.57±0.07 | 0.62±0.08 | 0.70±0.08 | 0.64±0.12 | **0.72±0.20** |
| winter_wren | 0.95±0.02 | 0.96±0.02 | 0.92±0.05 | 0.95±0.04 | **0.97±0.02** |
| rank | 3.35 | 2.40 | 3.40 | 2.40 | **2.25** |

Table 5.3: Testing dataset balanced accuracy (BACC) of classification multiple-instance data. The best model is chosen according to the best BACC achieved on the validation dataset.

#### Clustering

Clustering and its evaluation are difficult tasks, even on tabular data. The whole procedure becomes even more complex with structured data. In clustering experiments on multiple instance data, *spsn+gmm* resulted in the best performance as observed in Table 5.7. Again, the poor performance of *spsn+realnvp* might be attributed to its overfitting. It is notable that *spsn+gmm* beat distance-based method *k-medoids* as the distance-based methods tend to be highly competitive on these tasks.

## 5.3 Hierarchical multiple-instance problems

### 5.3.1 Classification

Hierarchical multiple-instance datasets are the most difficult classification problems we face in this thesis. They also demonstrate the ability to scale SPSNs to tree-structured data in

| dataset | spsn+gmm | spsn+spn | spsn+realnvp | hmil | hmil+mcd |
|---|---|---|---|---|---|
| brown_creeper | 0.05±0.01 | 0.05±0.01 | 0.06±0.01 | 0.05±0.02 | **0.04±0.01** |
| corel_african | 0.03±0.01 | 0.03±0.01 | 0.03±0.01 | 0.03±0.01 | **0.02±0.01** |
| corel_beach | **0.01±0.00** | **0.01±0.00** | **0.01±0.00** | 0.02±0.00 | **0.01±0.00** |
| elephant | 0.16±0.03 | 0.16±0.06 | 0.26±0.08 | 0.17±0.06 | **0.14±0.04** |
| fox | 0.46±0.14 | 0.37±0.10 | 0.51±0.04 | 0.43±0.14 | **0.24±0.09** |
| musk_1 | 0.20±0.13 | 0.15±0.09 | **0.11±0.06** | 0.17±0.14 | 0.12±0.10 |
| musk_2 | **0.17±0.07** | 0.27±0.08 | 0.21±0.06 | 0.21±0.08 | 0.18±0.06 |
| mutagenesis_1 | **0.08±0.05** | 0.10±0.04 | 0.13±0.03 | 0.16±0.03 | 0.11±0.05 |
| mutagenesis_2 | 0.30±0.19 | 0.25±0.13 | **0.21±0.12** | 0.27±0.10 | 0.31±0.14 |
| newsgroups_1 | 0.24±0.09 | **0.19±0.10** | 0.32±0.11 | 0.23±0.08 | 0.20±0.10 |
| newsgroups_2 | 0.43±0.19 | 0.47±0.10 | 0.40±0.11 | 0.34±0.08 | **0.31±0.11** |
| newsgroups_3 | 0.29±0.09 | 0.35±0.07 | 0.42±0.06 | 0.33±0.13 | **0.19±0.13** |
| protein | 0.15±0.06 | 0.13±0.05 | 0.14±0.04 | 0.14±0.03 | **0.10±0.02** |
| tiger | 0.24±0.04 | 0.22±0.05 | 0.33±0.03 | 0.23±0.13 | **0.21±0.04** |
| ucsb_breast_cancer | 0.25±0.10 | 0.23±0.07 | 0.22±0.05 | **0.20±0.06** | 0.21±0.10 |
| web_1 | 0.32±0.10 | **0.21±0.09** | 0.33±0.14 | 0.31±0.05 | 0.30±0.07 |
| web_2 | 0.45±0.18 | **0.11±0.08** | 0.43±0.10 | 0.25±0.04 | 0.30±0.04 |
| web_3 | 0.40±0.08 | **0.05±0.06** | 0.28±0.07 | 0.36±0.17 | 0.25±0.06 |
| web_4 | 0.34±0.10 | 0.23±0.04 | **0.21±0.06** | 0.33±0.12 | 0.23±0.17 |
| winter_wren | 0.03±0.01 | 0.03±0.01 | 0.05±0.03 | 0.03±0.02 | **0.02±0.01** |
| rank | 3.35 | 2.30 | 3.45 | 3.05 | **1.75** |

Table 5.4: Testing dataset expected calibration error (ECE) of classification multiple-instance data. The best model is chosen according to the best BACC achieved on the validation dataset.

their full hierarchy. Table 5.8 suggests that SPSNs present a competitive approach compared to neural networks. *spsn* significantly lacks on the world dataset, which contains a lot of observations in the form of text (strings), which are easier to learn using neural networks. In terms of calibration, *hmil* model appears to be better calibrated. The cause of this phenomenon is the subject of further investigation in future work.

### 5.3.2 Clustering

As mentioned in experimental results on multiple-instance problems, clustering problems are hard. It becomes even harder with more complex data, such as hierarchical multiple-instance problems. The results are present in Table 5.10. The table illustrates that *spsn* performs better than *k-medoids* in terms of UACC. However, some fields in the table are filled with *nan*, and those say that the model training or evaluation was not finished. In all cases, it was due to a restricted computational budget, especially RAM. For *spsn*, the problem arises when the number of clusters is large since, for clustering, we use a mixture of SPSNs. For the genes dataset, which has 15 clusters, this essentially means evaluating 15 independent SPSNs. The problem with *k-medoids* is that it requires the computation of pairwise distances between all the observations on training data, which becomes quickly infeasible. Dataset pubmed_diabetes and ftp both have more than ten thousand observations, which confirms our findings.

No clear preference for clustering methods can be given. The ambiguity in evaluating clustering results caused by possible permutation of clusters and low values of UACC makes

| dataset | diag | scalar | unit | reg. diag | reg. scalar |
|---|---|---|---|---|---|
| brown_creeper | **0.95±0.01** | 0.92±0.03 | **0.95±0.02** | 0.94±0.03 | **0.95±0.01** |
| corel_african | **0.83±0.05** | 0.81±0.06 | 0.79±0.06 | 0.80±0.06 | 0.82±0.04 |
| corel_beach | 0.89±0.05 | 0.89±0.04 | 0.88±0.05 | 0.84±0.07 | **0.91±0.04** |
| elephant | 0.81±0.05 | **0.82±0.04** | 0.80±0.05 | 0.71±0.04 | **0.82±0.03** |
| fox | **0.59±0.11** | 0.56±0.05 | 0.51±0.10 | 0.54±0.15 | 0.49±0.04 |
| musk_1 | 0.80±0.13 | 0.78±0.07 | 0.83±0.09 | **0.87±0.06** | 0.83±0.09 |
| musk_2 | **0.85±0.04** | **0.85±0.06** | 0.79±0.05 | 0.80±0.05 | 0.83±0.07 |
| mutagenesis_1 | 0.83±0.08 | 0.82±0.08 | **0.87±0.07** | 0.81±0.08 | 0.84±0.05 |
| mutagenesis_2 | 0.72±0.20 | 0.73±0.16 | **0.75±0.14** | 0.61±0.20 | 0.63±0.16 |
| newsgroups_1 | **0.82±0.14** | 0.77±0.08 | 0.68±0.12 | 0.70±0.08 | 0.76±0.07 |
| newsgroups_2 | **0.75±0.08** | 0.61±0.14 | 0.50±0.11 | 0.61±0.08 | 0.57±0.19 |
| newsgroups_3 | 0.69±0.10 | **0.71±0.09** | 0.66±0.08 | 0.60±0.12 | 0.49±0.10 |
| protein | **0.66±0.08** | 0.65±0.12 | 0.64±0.15 | 0.64±0.10 | 0.65±0.13 |
| tiger | 0.71±0.08 | **0.75±0.04** | 0.73±0.07 | 0.72±0.06 | 0.74±0.04 |
| ucsb_breast_cancer | **0.83±0.12** | 0.71±0.09 | 0.81±0.08 | **0.83±0.13** | **0.83±0.06** |
| web_1 | 0.51±0.11 | **0.65±0.12** | 0.54±0.12 | 0.52±0.06 | 0.55±0.04 |
| web_2 | 0.45±0.14 | 0.43±0.21 | **0.49±0.18** | 0.43±0.16 | 0.46±0.19 |
| web_3 | 0.53±0.09 | 0.53±0.07 | **0.60±0.10** | 0.49±0.11 | 0.51±0.10 |
| web_4 | 0.55±0.15 | 0.57±0.07 | 0.59±0.10 | 0.57±0.04 | **0.63±0.10** |
| winter_wren | **0.98±0.02** | 0.93±0.03 | 0.95±0.03 | 0.97±0.01 | 0.95±0.03 |
| rank | **2.30** | 2.65 | 3.05 | 3.65 | 2.50 |

Table 5.5: Ablation study for covariance type of *spsn+gmm* model on classification multiple-instance problems. Testing dataset balanced accuracy (BACC) of classification multiple-instance data is presented. The best model is chosen according to the best BACC achieved on the validation dataset.

drawing clear decisions unreliable. However, from the methods‘ definition and issues mentioned, *k-medoids* can be deployed on data with a bigger estimated number of clusters and fewer observations. On the other hand, *spsn* handled a higher number of observations quite well, and its problems arise for a bigger number of clusters.

| dataset | diag | scalar | unit | reg. diag | reg. scalar |
|---|---|---|---|---|---|
| brown_creeper | **0.04±0.01** | 0.07±0.02 | **0.04±0.01** | 0.05±0.02 | 0.05±0.01 |
| corel_african | **0.01±0.00** | 0.02±0.01 | 0.02±0.00 | 0.02±0.01 | 0.05±0.01 |
| corel_beach | **0.01±0.01** | 0.02±0.00 | **0.01±0.00** | 0.02±0.00 | 0.04±0.01 |
| elephant | 0.19±0.05 | **0.16±0.03** | 0.19±0.04 | 0.28±0.04 | 0.19±0.03 |
| fox | **0.41±0.11** | 0.43±0.06 | 0.49±0.09 | 0.46±0.14 | 0.50±0.04 |
| musk_1 | 0.20±0.13 | 0.22±0.07 | 0.17±0.09 | **0.13±0.06** | 0.15±0.08 |
| musk_2 | **0.15±0.04** | 0.17±0.07 | 0.21±0.05 | 0.21±0.07 | 0.18±0.08 |
| mutagenesis_1 | 0.10±0.04 | 0.10±0.02 | **0.08±0.05** | 0.15±0.06 | 0.12±0.04 |
| mutagenesis_2 | 0.26±0.19 | **0.25±0.14** | **0.25±0.12** | 0.32±0.17 | 0.30±0.14 |
| newsgroups_1 | **0.18±0.15** | 0.24±0.09 | 0.30±0.10 | 0.30±0.08 | 0.24±0.07 |
| newsgroups_2 | **0.25±0.08** | 0.39±0.15 | 0.49±0.11 | 0.39±0.08 | 0.43±0.19 |
| newsgroups_3 | 0.31±0.10 | **0.29±0.09** | 0.35±0.09 | 0.40±0.12 | 0.50±0.09 |
| protein | **0.15±0.03** | **0.15±0.03** | 0.21±0.09 | **0.15±0.06** | 0.22±0.05 |
| tiger | 0.28±0.08 | **0.24±0.04** | 0.27±0.07 | 0.28±0.05 | 0.26±0.04 |
| ucsb_breast_cancer | 0.18±0.12 | 0.25±0.06 | 0.22±0.07 | **0.17±0.13** | 0.20±0.07 |
| web_1 | 0.39±0.09 | **0.32±0.10** | 0.37±0.10 | 0.41±0.09 | 0.43±0.04 |
| web_2 | **0.40±0.11** | 0.47±0.18 | 0.45±0.18 | 0.46±0.14 | 0.47±0.15 |
| web_3 | 0.40±0.08 | 0.35±0.09 | **0.24±0.04** | 0.29±0.08 | 0.35±0.09 |
| web_4 | 0.37±0.17 | 0.34±0.10 | **0.29±0.07** | 0.31±0.10 | 0.33±0.12 |
| winter_wren | **0.03±0.02** | **0.03±0.00** | 0.06±0.03 | **0.03±0.02** | 0.07±0.03 |
| rank | **2.10** | 2.40 | 2.60 | 2.95 | 3.70 |

Table 5.6: Ablation study for covariance type of *spsn+gmm* model on classification multiple-instance problems. Testing dataset expected calibration error (ECE) of classification multiple-instance data is presented. The best model is chosen according to the best BACC achieved on the validation dataset.

| dataset | spsn+gmm | spsn+realnvp | k-medoids |
|---|---|---|---|
| brown_creeper | **0.82±0.07** | 0.65±0.00 | 0.69±0.06 |
| corel_african | 0.91±0.00 | **0.92±0.00** | 0.59±0.10 |
| corel_beach | **0.92±0.00** | **0.92±0.00** | 0.70±0.06 |
| elephant | **0.61±0.09** | 0.49±0.10 | 0.58±0.09 |
| fox | 0.53±0.03 | **0.56±0.01** | **0.56±0.07** |
| musk_1 | **0.60±0.05** | 0.53±0.09 | 0.59±0.05 |
| musk_2 | 0.60±0.07 | **0.61±0.13** | 0.56±0.04 |
| mutagenesis_1 | **0.65±0.04** | 0.63±0.00 | 0.60±0.07 |
| mutagenesis_2 | **0.80±0.07** | 0.60±0.22 | 0.65±0.06 |
| newsgroups_1 | **0.76±0.09** | 0.47±0.14 | 0.70±0.06 |
| newsgroups_2 | **0.63±0.06** | 0.54±0.08 | 0.61±0.04 |
| newsgroups_3 | 0.56±0.07 | **0.57±0.08** | 0.52±0.03 |
| protein | **0.95±0.00** | **0.95±0.00** | 0.78±0.07 |
| tiger | **0.67±0.05** | 0.52±0.08 | 0.65±0.13 |
| ucsb_breast_cancer | **0.70±0.10** | 0.38±0.07 | 0.68±0.11 |
| web_1 | 0.63±0.09 | 0.60±0.12 | **0.67±0.00** |
| web_2 | 0.63±0.10 | 0.65±0.17 | **0.80±0.00** |
| web_3 | **0.80±0.00** | 0.73±0.09 | **0.80±0.00** |
| web_4 | 0.73±0.13 | **0.76±0.04** | 0.71±0.06 |
| winter_wren | 0.78±0.00 | 0.78±0.00 | **0.84±0.05** |
| rank | **1.50** | 2.15 | 2.10 |

Table 5.7: Testing dataset unsupervised accuracy (UACC) of clustering multiple-instance data. The best model is chosen according to the best UACC achieved on the validation dataset, and the permutation of predicted clusters on the validation data is stored and used to evaluate testing data.

| dataset | spsn | hmil |
|---|---|---|
| chess | 0.31±0.05 | **0.35±0.04** |
| citeseer | **0.71±0.01** | 0.69±0.01 |
| cora | **0.85±0.01** | 0.81±0.01 |
| ftp | **0.43±0.09** | 0.40±0.05 |
| genes | 0.79±0.05 | **0.82±0.02** |
| hepatitis | **0.91±0.03** | 0.90±0.02 |
| mutagenesis | **0.88±0.04** | **0.88±0.05** |
| ptc | **0.57±0.05** | 0.56±0.07 |
| pubmed_diabetes | 0.88±0.01 | **0.89±0.01** |
| uw_cse | 0.78±0.06 | **0.79±0.08** |
| webkp | 0.63±0.03 | **0.67±0.05** |
| world | 0.73±0.11 | **0.87±0.06** |
| rank | 1.50 | **1.42** |

Table 5.8: Testing dataset balanced accuracy (BACC) of classification hierarchical multiple-instance data. The best model is chosen according to the best BACC achieved on the validation dataset.

| dataset | spsn | hmil |
|---|---|---|
| chess | 0.65±0.05 | **0.32±0.21** |
| citeseer | 0.22±0.01 | **0.14±0.07** |
| cora | 0.12±0.01 | **0.08±0.05** |
| ftp | 0.50±0.06 | **0.05±0.02** |
| genes | 0.07±0.01 | **0.01±0.00** |
| hepatitis | **0.09±0.03** | **0.09±0.03** |
| mutagenesis | 0.11±0.03 | **0.09±0.04** |
| ptc | **0.28±0.06** | 0.34±0.09 |
| pubmed_diabetes | 0.05±0.01 | **0.03±0.01** |
| uw_cse | **0.10±0.04** | **0.10±0.05** |
| webkp | 0.23±0.02 | **0.08±0.03** |
| world | 0.30±0.10 | **0.09±0.04** |
| rank | 1.75 | **1.08** |

Table 5.9: Testing dataset expected calibration error (ECE) of classification hierarchical multiple-instance data. The best model is chosen according to the best BACC achieved on the validation dataset.

| dataset | spsn | k-medoids |
|---|---|---|
| chess | **0.38±0.00** | 0.35±0.05 |
| citeseer | 0.21±0.01 | **0.27±0.03** |
| cora | **0.33±0.07** | 0.25±0.02 |
| hepatitis | **0.75±0.05** | 0.49±0.07 |
| mutagenesis | 0.75±0.00 | **0.76±0.02** |
| premier_league | **0.41±0.00** | 0.35±0.08 |
| ptc | **0.57±0.00** | 0.43±0.00 |
| uw_cse | **0.46±0.16** | 0.43±0.15 |
| webkp | **0.49±0.01** | 0.33±0.06 |
| world | 0.20±0.02 | **0.24±0.04** |
| ftp | 0.50±0.00 | *nan* |
| pubmed_diabetes | 0.57±0.00 | *nan* |
| genes | *nan* | 0.21±0.02 |

Table 5.10: Testing dataset unsupervised accuracy (UACC) of clustering hierarchical multiple-instance data. The best model is chosen according to the best UACC achieved on the validation dataset, and the permutation of predicted clusters on the validation data is stored and used to evaluate testing data.

# Chapter 6

# Conclusion

The thesis aimed to propose a density estimation model tailored for tree-structured data as an extension of Sum-Product Networks (SPNs). The defined goal of the thesis was fulfilled. We further summarize the achieved results in the following text.

We provided an overview of classical probability density estimation methods, estimation, and practical applications. WE resented contemporary methods for estimating density. It concluded by introducing the Sum-Product Network (SPN) probabilistic model, emphasizing its unique advantages, particularly its ability to perform exact and efficient probabilistic inference.

The following chapter delved deeper into the data studied – trees. It provided a detailed exploration of tree-structured data, discussing prior research on learning from such structures. Notably, it highlighted the probabilistic model of random finite sets and introduced the Sum-Product-Set model, an extension of Sum-Product Networks designed for leaf-attributed trees.

In the subsequent sections, the experimental scope was specified, emphasizing classification and clustering tasks due to the lack of competitive density estimation methods for general tree-structured data. This segment provided detailed insights into performance metrics, dataset selection, and the setup necessary for empirical analysis. Following this, the empirical findings from the predefined tasks were rigorously presented and discussed, thoroughly analyzing the outcomes of classification and clustering experiments conducted within the predefined framework.

The initial set of experiments centered on tabular data analysis, primarily comparing the effectiveness of Sum-Product Networks (SPNs) against neural networks, specifically Multi-Layer Perceptrons (MLPs), in classification tasks Results indicated that both SPN and MLP models exhibited competitive performance. Notably, the MLP combined with Monte Carlo Dropout (MCD) showed slightly superior classification results. Additionally, the belief that probabilistic models possess better calibration than non-probabilistic counterparts was challenged, with the experiment's results showing no clear advantage in calibration metrics. Moreover, SPN models showcased promising results in handling missing data while performing equally well or even better than MLP models.

Moving onto experiments with multiple instance data, the focus shifted to evaluating SPSN models for random finite sets. Results suggested that SPN models displayed potential in classification tasks compared to established baseline models like the Heterogeneous Multi-Instance Learning (HMIL) model. However, combinations of HMIL with Monte Carlo Dropout exhibited superior performance, mimicking the results from tabular data. Among the combination of SPSN approaches tested, SPSN with RealNVP flow leaves faced challenges such as overfitting issues, impacting their performance negatively. Furthermore, an in-depth analysis of the covariance type of the SPSN with the Gaussian Mixture Model (GMM) leaves highlighted that a diagonal covariance matrix yielded the best results, both in terms of classification accuracy and calibration error. In clustering experiments conducted on multiple

instance data, SPSN with Gaussian Mixture Model (GMM) emerged as the most effective model, outperforming distance-based methods. The proposed method *spsn* is also competitive on BACC on multiple instance data compared to *hmil*, which is a really good result given the completely different nature of both models.

Overall, these experiments offered extensive insights into the strengths and limitations of various models concerning classification, handling missing data, and clustering tasks across tabular, multiple-instance data and hierarchical multiple-instance data. Moreover, the fact that tractable probabilistic models with relatively sparse computational graphs rivalled intractable neural networks with dense computational graphs is a significant highlight of the thesis.

As for future work, we have to ask some more fundamental questions, such as: Are discriminative methods generally better calibrated than generative ones? When is it better to use probabilistic methods for discriminative tasks over non-probabilistic ones? Are imputation techniques enough for handling missing values?

# References


[1] Šimon Mandlík et al. "JsonGrinder.Jl: Automated Differentiable Neural Architecture for Embedding Arbitrary JSON Data". In: *J. Mach. Learn. Res.* 23.1 (2022). ISSN: 1532-4435.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[3] Kaiming He et al. *Mask R-CNN*. 2018. arXiv: 1703.06870 [cs.CV].

[4] Jerome Revaud et al. "R2D2: Reliable and Repeatable Detector and Descriptor". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/3198dfd0aef271d22f7bcddd6f12f5cb-Paper.pdf.

[5] Tomáš Mikolov et al. "Statistical language models based on neural networks". In: *Presentation at Google, Mountain View, 2nd April* 80.26 (2012).

[6] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[7] Břetislav Šopík and Tomáš Strenáčik. "Tree edit distance for hierarchical data compatible with HMIL paradigm". In: *arXiv preprint arXiv:2208.00782* (2022).

[8] Tomas Pevny and Petr Somol. "Discriminative Models for Multi-Instance Problems with Tree Structure". In: *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*. AISec '16. Vienna, Austria: Association for Computing Machinery, 2016, 83–91. ISBN: 9781450345736. DOI: 10.1145/2996758.2996761. URL: https://doi.org/10.1145/2996758.2996761.

[9] Tomas Pevny and Marek Dedic. *Nested Multiple Instance Learning in Modelling of HTTP network traffic*. 2020. arXiv: 2002.04059 [cs.CR].

[10] Lukas Machlica, Karel Bartos, and Michal Sofka. *Learning detectors of malicious web requests for intrusion detection in network traffic*. 2017. arXiv: 1702.02530 [stat.ML].

[11] Hyrum S. Anderson and Phil Roth. *EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models*. 2018. arXiv: 1804.04637 [cs.CR].

[12] Thomas G. Dietterich, Richard H. Lathrop, and Tomás Lozano-Pérez. "Solving the multiple instance problem with axis-parallel rectangles". In: *Artificial Intelligence* 89.1 (1997), pp. 31–71. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(96)00034-3. URL: https://www.sciencedirect.com/science/article/pii/S0004370296000343.

[13] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 7159. 2014. DOI: 10.17487/RFC7159. URL: https://www.rfc-editor.org/info/rfc7159.

[14] Simon Mandlik and Tomas Pevny. "Mapping the internet: Modelling entity interactions in complex heterogeneous networks". In: *arXiv preprint arXiv:2104.09650* (2021).

[15] R. A. FISHER. "THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS". In: *Annals of Eugenics* 7.2 (1936), pp. 179–188. DOI: https://doi.org/10.1111/j.1469-1809.1936.tb02137.x.

[16] Branislav Bosansky et al. "Avast-CTU public CAPE dataset". In: *arXiv preprint arXiv:2209.03188* (2022).

[17] Dmitrijs Trizna et al. *Nebula: Self-Attention for Dynamic Malware Analysis*. 2023. arXiv: 2310.10664 [cs.CR].

[18] Jian Liu et al. "An effective biomedical data migration tool from resource description framework to JSON". In: *Database* 2019 (July 2019), baz088. ISSN: 1758-0463. DOI: 10.1093/database/baz088.

[19] Asim Kumar Debnath et al. "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity". In: *Journal of medicinal chemistry* 34.2 (1991), pp. 786–797.

[20] Aditya Ramesh et al. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. arXiv: 2204.06125 [cs.CV].

[21] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[22] Patrick Billingsley. *Probability and measure*. John Wiley & Sons, 2017.

[23] René L Schilling. *Measures, integrals and martingales*. Cambridge University Press, 2017.

[24] Richard J Rossi. *Mathematical statistics: an introduction to likelihood based inference*. John Wiley & Sons, 2018.

[25] Hirotogu Akaike. "Information Theory and an Extension of the Maximum Likelihood Principle". In: *Selected Papers of Hirotugu Akaike*. Ed. by Emanuel Parzen, Kunio Tanabe, and Genshiro Kitagawa. New York, NY: Springer New York, 1998, pp. 199–213. ISBN: 978-1-4612-1694-0. DOI: 10.1007/978-1-4612-1694-0_15. URL: https://doi.org/10.1007/978-1-4612-1694-0_15.

[26] D.M. Hawkins. *Identification of Outliers*. Monographs on applied probability and statistics. Chapman and Hall, 1980. ISBN: 9780412219009.

[27] Tony Jebara. *Machine learning: discriminative and generative*. Vol. 755. Springer Science & Business Media, 2012.

[28] Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*. Vol. 31. Springer Science & Business Media, 2013.

[29] Pasha Khosravi et al. "What to Expect of Classifiers? Reasoning about Logistic Regression with Missing Features". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. IJCAI'19. Macao: AAAI Press, 2019, 2716–2724. ISBN: 9780999241141.

[30] Eric Wang, Pasha Khosravi, and Guy Van den Broeck. "Probabilistic sufficient explanations". In: *arXiv preprint arXiv:2105.10118* (2021).

[31] Robert Gens and Domingos Pedro. "Learning the Structure of Sum-Product Networks". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, pp. 873–880. URL: https://proceedings.mlr.press/v28/gens13.html.

[32] Andy Shih, Dorsa Sadigh, and Stefano Ermon. "Training and Inference on Any-Order Autoregressive Models the Right Way". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 2762–2775.

[33] Yang Li, Shoaib Akbar, and Junier Oliva. "ACFlow: Flow models for arbitrary conditional likelihoods". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5831–5841.

[34] Ryan Strauss and Junier B Oliva. "Posterior Matching for Arbitrary Conditioning". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 18088–18099.

[35] Ian Goodfellow et al. "Generative adversarial networks". In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[36] Rob J. Hyndman. "Computing and Graphing Highest Density Regions". In: *The American Statistician* 50.2 (1996), pp. 120–126. DOI: 10.1080/00031305.1996.10474359.

[37] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[38] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using real nvp". In: *arXiv preprint arXiv:1605.08803* (2016).

[39] Hoifung Poon and Pedro Domingos. "Sum-product networks: A new deep architecture". In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. IEEE. 2011, pp. 689–690.

[40] Konstantinos Plataniotis and D. Hatzinakos. "Gaussian mixtures and their applications to signal processing". In: Jan. 2000.

[41] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. "Importance weighted autoencoders". In: *arXiv preprint arXiv:1509.00519* (2015).

[42] Laurent Dinh, David Krueger, and Yoshua Bengio. "Nice: Non-linear independent components estimation". In: *arXiv preprint arXiv:1410.8516* (2014).

[43] Robert Peharz et al. "On the latent variable interpretation in sum-product networks". In: *IEEE transactions on pattern analysis and machine intelligence* 39.10 (2016), pp. 2030–2044.

[44] Robert Peharz et al. "On Theoretical Properties of Sum-Product Networks". In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Guy Lebanon and S. V. N. Vishwanathan. Vol. 38. Proceedings of Machine Learning Research. San Diego, California, USA: PMLR, 2015, pp. 744–752. URL: https://proceedings.mlr.press/v38/peharz15.html.

[45] YooJung Choi, Antonio Vergari, and Guy Van den Broeck. "Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models". In: (2020). URL: http://starai.cs.ucla.edu/papers/ProbCirc20.pdf.

[46] Mattia Desana and Christoph Schnörr. "Expectation maximization for sum-product networks as exponential family mixture models". In: *arXiv preprint arXiv:1604.07243* (2016).

[47] Robert Peharz et al. "Random Sum-Product Networks: A Simple and Effective Approach to Probabilistic Deep Learning". In: *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*. Ed. by Ryan P. Adams and Vibhav Gogate. Vol. 115. Proceedings of Machine Learning Research. PMLR, 2020, pp. 334–344. URL: https://proceedings.mlr.press/v115/peharz20a.html.

[48] Martin Trapp et al. "Bayesian Learning of Sum-Product Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/5421e013565f7f1afa0cfe8ad87a99ab-Paper.pdf.

[49] Adnan Darwiche. "A differential approach to inference in Bayesian networks". In: *Journal of the ACM (JACM)* 50.3 (2003), pp. 280–305.

[50] Robert Peharz et al. "Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 7563–7574. URL: https://proceedings.mlr.press/v119/peharz20a.html.

[51] Aaron Dennis and Dan Ventura. "Learning the architecture of sum-product networks using clustering on variables". In: *Advances in Neural Information Processing Systems* 25 (2012).

[52] Amirmohammad Rooshenas and Daniel Lowd. "Learning Sum-Product Networks with Direct and Indirect Variable Interactions". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, 2014, pp. 710–718. URL: https://proceedings.mlr.press/v32/rooshenas14.html.

[53] Alejandro Molina et al. "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (2018). DOI: 10.1609/aaai.v32i1.11731. URL: https://ojs.aaai.org/index.php/AAAI/article/view/11731.

[54] Antonio Vergari et al. "Automatic Bayesian Density Analysis". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (2019), pp. 5207–5215. DOI: 10.1609/aaai.v33i01.33015207. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4977.

[55] Fabrizio Ventola et al. “Probabilistic circuits that know what they don’t know”. In: *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*. Ed. by Robin J. Evans and Ilya Shpitser. Vol. 216. Proceedings of Machine Learning Research. PMLR, 2023, pp. 2157–2167. URL: https://proceedings.mlr.press/v216/ventola23a.html.

[56] Tomáš Pevný et al. “Sum-Product-Transform Networks: Exploiting Symmetries using Invertible Transformations”. In: *Proceedings of the 10th International Conference on Probabilistic Graphical Models*. Ed. by Manfred Jaeger and Thomas Dyhre Nielsen. Vol. 138. Proceedings of Machine Learning Research. PMLR, 2020, pp. 341–352. URL: https://proceedings.mlr.press/v138/pevny20a.html.

[57] Sahil Sidheekh, Kristian Kersting, and Sriraam Natarajan. “Probabilistic flow circuits: towards unified deep models for tractable probabilistic inference”. In: *Uncertainty in Artificial Intelligence*. PMLR. 2023, pp. 1964–1973.

[58] Ping Liang Tan and Robert Peharz. “Hierarchical Decompositional Mixtures of Variational Autoencoders”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6115–6124. URL: https://proceedings.mlr.press/v97/tan19b.html.

[59] Harrison Edwards and Amos Storkey. “Towards a neural statistician”. In: *arXiv preprint arXiv:1606.02185* (2016).

[60] Krikamol Muandet and Bernhard Schölkopf. “One-class support measure machines for group anomaly detection”. In: *arXiv preprint arXiv:1303.0309* (2013).

[61] Maximilian Ilse, Jakub Tomczak, and Max Welling. “Attention-based Deep Multiple Instance Learning”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 2127–2136. URL: https://proceedings.mlr.press/v80/ilse18a.html.

[62] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[63] Ba-Ngu Vo et al. “Model-based learning for point pattern data”. In: *Pattern Recognition* 84 (2018), pp. 136–151. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2018.07.008. URL: https://www.sciencedirect.com/science/article/pii/S0031320318302395.

[64] Ronald Mahler. *Statistical Multisource-Multitarget Information Fusion*. Artech, 2007. ISBN: 9781596930933.

[65] Krikamol Muandet et al. “Learning from distributions via support measure machines”. In: *Advances in neural information processing systems* 25 (2012).

[66] Min-Ling Zhang and Zhi-Hua Zhou. “Multi-instance clustering with applications to multi-instance prediction”. In: *Applied intelligence* 31 (2009), pp. 47–68.

[67] Haoqiang Fan, Hao Su, and Leonidas Guibas. *A Point Set Generation Network for 3D Object Reconstruction from a Single Image*. 2016. arXiv: 1612.00603 [cs.CV].

[68] Robin Fuchs, Denys Pommeret, and Cinzia Viroli. “Mixed Deep Gaussian Mixture Model: a clustering model for mixed datasets”. In: *Advances in Data Analysis and Classification* 16.1 (2022), pp. 31–53.

[69] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. *Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks*. 2015. arXiv: 1503.00075 [cs.CL].

[70] Milan Papez et al. “Sum-Product-Set Networks”. In: *The 6th Workshop on Tractable Probabilistic Modeling*. 2023. URL: https://openreview.net/forum?id=8hqxY5fUwg.

[71] Philip Bille. “A survey on tree edit distance and related problems”. In: *Theoretical computer science* 337.1-3 (2005), pp. 217–239.

[72] B.-N. Vo, S. Singh, and A. Doucet. “Sequential Monte Carlo methods for multitarget filtering with random finite sets”. In: *IEEE Transactions on Aerospace and Electronic Systems* 41.4 (2005), pp. 1224–1245. DOI: 10.1109/TAES.2005.1561884.

[73] Ronald Mahler. ""Statistics 102" for Multisource-Multitarget Detection and Tracking". In: *IEEE Journal of Selected Topics in Signal Processing* 7.3 (2013), pp. 376–389. DOI: 10.1109/JSTSP.2013.2253084.

[74] Ronald Mahler. ""Statistics 103" for Multitarget Tracking". In: *Sensors* 19.1 (2019). ISSN: 1424-8220. DOI: 10.3390/s19010202. URL: https://www.mdpi.com/1424-8220/19/1/202.

[75] Jesper Moller and Rasmus Plenge Waagepetersen. *Statistical inference and simulation for spatial point processes*. CRC press, 2003.

[76] Kay Henning Brodersen et al. "The balanced accuracy and its posterior distribution". In: *2010 20th international conference on pattern recognition*. IEEE. 2010, pp. 3121–3124.

[77] B.W. Matthews. "Comparison of the predicted and observed secondary structure of T4 phage lysozyme". In: *Biochimica et Biophysica Acta (BBA) - Protein Structure* 405.2 (1975), pp. 442–451. ISSN: 0005-2795. DOI: https://doi.org/10.1016/0005-2795(75)90109-9.

[78] William M Rand. "Objective criteria for the evaluation of clustering methods". In: *Journal of the American Statistical association* 66.336 (1971), pp. 846–850.

[79] Lawrence Hubert and Phipps Arabie. "Comparing partitions". In: *Journal of classification* 2 (1985), pp. 193–218.

[80] Peter J Rousseeuw. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis". In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65.

[81] Xianghong Lin, Xiaofei Yang, and Ying Li. "A deep clustering algorithm based on Gaussian mixture model". In: *Journal of Physics: Conference Series*. Vol. 1302. 3. IOP Publishing. 2019, p. 032012.

[82] Harold W Kuhn. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[83] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. "Obtaining well calibrated probabilities using bayesian binning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 29. 1. 2015.

[84] Kolby Nottingham Markelle Kelly Rachel Longjohn. *The UCI Machine Learning Repository*. URL: https://archive.ics.uci.edu.

[85] Jan Motl and Oliver Schulte. *The CTU Prague Relational Learning Repository*. 2015. arXiv: 1511.03086 [cs.LG].

[86] Tomas Pevny and Matej Racinsky. *JsonGrinder.jl: a flexible library for automated feature engineering and conversion of JSONs to Mill.jl structures*. Version 2.4. URL: https://github.com/CTUAvastLab/JsonGrinder.jl.

[87] Yarin Gal and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059.

[88] Jeff Bezanson et al. "Julia: A fresh approach to numerical computing". In: *SIAM review* 59.1 (2017), pp. 65–98. URL: https://doi.org/10.1137/141000671.

[89] Janez Demšar. "Statistical comparisons of classifiers over multiple data sets". In: *The Journal of Machine learning research* 7 (2006), pp. 1–30.

[90] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

# Chapter A

# Appendix A: Hyperparameters

| description | values |
|---|---|
| number of product node splits (D) | $\{1, 2, 3, 4, 5\}$ |
| number of product node children (S) | $\{2, 4, 8\}$ |
| number of input distributions per leaf (I) | $\{2, 4, 8\}$ |
| number of product node scope repetitions (R) | $\{1, 5, 10\}$ |
| leaf distribution | normal |
| cardinality distribution | Poisson |
| number of training epochs | 500 |
| optimizer | ADAM [90] |
| learning rate | $\{0.1, 0.01, 0.01\}$ |
| batch_size | 50 |

Table A.1: Hyperparameter setup for RATSPN model for classification on tabular datasets. See [47] for more details about RATSPN's hyperparameters.

| description | values |
|---|---|
| hidden layer dimension | $\{5, 10, 20, 30, 40\}$ |
| number of hidden layers | $\{1, 2, 3, 4\}$ |
| aggregation function | concatenation of mean and maximum |
| activation function | ReLU, Sigmoid |
| number of training epochs | 500 |
| optimizer | ADAM |
| learning rate | $\{0.1, 0.01, 0.01\}$ |
| batch_size | 50 |

Table A.2: Hyperparameter setup for MLP model for classification on tabular datasets.

| description | values |
|---|---|
| number of product node splits | 1 |
| number of sum node children | $\{1, 2, 4, 8, 16, 32, 64\}$ |
| leaf distribution type | Gaussian |
| covariance matrix type | {diagnoal, scalar, unit, diagonal+regularization ($\sigma_{\min} = 0.01$), scalar+regularization ($\sigma_{\min} = 0.01$)} |
| cardinality distribution | Poisson |
| number of training epochs | 4000 |
| optimizer | ADAM [90] |
| learning rate | $\{0.1, 0.01, 0.01\}$ |

Table A.3: Hyperparameter setup for SPSN+GMM model for multiple instance datasets.

| description | values |
|---|---|
| number of product node splits | 1 |
| number of sum node children | 1 |
| leaf distribution type | RealNVP flow |
| cardinality distribution | Poisson |
| number of training epochs | 4000 |
| optimizer | ADAM [90] |
| learning rate | $\{0.1, 0.01, 0.01\}$ |

Table A.4: Hyperparameter setup for SPSN+RealNVP model for multiple instance datasets.

| description | values |
|---|---|
| number of product node splits | $\{2, 3, 4\}$ |
| number of sum node children | $\{2, 4, 6, 8, 10\}$ |
| leaf distribution type | Gaussian |
| covariance matrix type | {diagnoal, scalar, unit, diagonal+regularization ($\sigma_{\min} = 0.01$), scalar+regularization ($\sigma_{\min} = 0.01$)} |
| cardinality distribution | Poisson |
| number of training epochs | 4000 |
| optimizer | ADAM [90] |
| learning rate | $\{0.1, 0.01, 0.01\}$ |

Table A.5: Hyperparameter setup for SPSN+SPN model for multiple instance datasets.

| description | values |
|---|---|
| hidden layer dimension | $\{8, 16, 32, 64, 128, 256\}$ |
| number of hidden layers per edge of schema | $\{1, 2, 3, 4\}$ |
| aggregation function | concatenation of mean and maximum |
| activation function | ReLU, Sigmoid |
| number of training epochs | 1000 |
| optimizer | ADAM |
| learning rate | $\{0.1, 0.01, 0.01\}$ |

Table A.6: Hyperparameter setup for HMIL model for multiple instance datasets.

| description | values |
|---|---|
| number of product node splits | $\{1, 2, 3\}$ |
| number of product node children | 2 |
| number of sum node children | $\{1, 2, \ldots, 8\}$ |
| cardinality distribution | Poisson |
| number of training epochs | 200 |
| optimizer | ADAM [90] |
| learning rate | $\{0.1, 0.01, 0.01\}$ |
| batch_size | 10 |

Table A.7: Hyperparameter setup for SPSN model for clustering and classification on hierarchical multiple instance datasets.

| description | values |
|---|---|
| hidden layer dimension | $\{10, 20, 30, 40, 50\}$ |
| number of hidden layers per edge of schema | $\{1, 2\}$ |
| aggregation function | concatenation of mean and maximum |
| activation function | ReLU, Sigmoid |
| number of training epochs | 200 |
| optimizer | ADAM |
| learning rate | $\{0.1, 0.01, 0.01\}$ |
| batch_size | 10 |

Table A.8: Hyperparameter setup for HMIL model for classification on hierarchical multiple instance datasets.