Faculty of Information Technology
Monash University

Probability Estimation for Bayesian Network Classifiers and Decision Trees

A thesis submitted for the degree of Doctor of Philosophy
at Monash University in 2020

He ZHANG

Supervisor: Wray Buntine, François Petitjean

March 2020

Copyright notice

Abstract

For many practical applications, it is more important for classifiers to output accurate class probability estimates than class labels, because knowing the probabilities that a test sample belongs to each class not only gives us an idea of the reliability of the prediction but also helps domain experts make better decisions. Unfortunately, due to the data sparsity problem, Bayesian network classifiers and decision trees cannot provide accurate class probability estimates when the observed class frequencies are used as the estimates. This fact has led to the use of simple probability smoothing techniques for classifiers, such as Laplace smoothing and M-estimation, to make the class probability estimates less extreme by adding some prior probabilities. A more sophisticated technique is the hierarchical probability smoothing method, which assumes that the class probability of the current node in a hierarchical tree-structured model depends on the probabilities of its parent nodes in some hierarchy. The recent Hierarchical Dirichlet Process (HDP) smoothing method provides the most advanced parameter estimation for Bayesian network classifiers and language models. However, it is computation-intensive and has not yet been introduced to decision trees. Another commonly used technique for improving the class probability estimates of classifiers is to combine the results of multiple classifiers using ensemble learning techniques. This thesis aims to improve the class probability estimation of Bayesian network classifiers and decision trees by using advanced hierarchical probability smoothing and ensemble learning techniques. The main contributions are as follows. First, a new ensemble algorithm ESKDB (Ensemble SKDB) has been proposed to improve the performance of Bayesian network classifiers, which can handle both categorical and numerical attributes. ESKDB, together with the HDP smoothing, completely surpasses the Random Forest and achieves similar results to the XGBoost algorithm; Second, we

proposed the HGS (Hierarchical Gradient Smoothing) algorithm that can effectively improve the probability estimation of decision trees, making a single tree achieve similar performance to a Random Forest of ten trees; Last, we made the HDP smoothing much faster by adding a new appropriate gamma prior. In summary, this project not only makes the Bayesian network classifiers and decision tree obtain the best class probability estimates in their respective fields but also further improves the development of hierarchical probability smoothing techniques.

Declaration

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Print Name: He ZHANG

Date: 23/06/2020

Publications During Enrolment

Large portions of this thesis have been submitted in the form of journal and conference papers. This applies to:

- Chapter 3 was presented in:
  Bayesian Network Classifiers using Ensembles and Smoothing. He Zhang, Wray Buntine, François Petitjean. Accepted by the journal of KAIS (Knowledge and Information System) (Q1 journal) (Zhang et al., 2020a).

- Chapter 4 was presented in:
  Hierarchical Gradient Smoothing for Probability Estimation Trees. He Zhang, Wray Buntine, François Petitjean. Accepted by the PAKDD 2020 conference ('A'-ranked conference) (Zhang et al., 2020b).

Acknowledgements

It is not my effort alone to complete the doctoral study because there are many people behind me to support and encourage me. Please allow me to take this opportunity to express my sincere appreciation to them. First and foremost, I would like to thank my two supervisors: Prof. Wray Buntine and Dr François Petitjean.

I was deeply impressed by Wray's enthusiasm for scientific research and his strong support for his students. He always has a lot of new ideas that inspire me, and I can see the light in his eyes when we discuss techniques. He has a solid knowledge of statistics that can express complex ideas in simple mathematical language. When he answers my questions, he often deduces formulas for me on papers, which is amazing to me. Although he is busy, he always has time for his students, including attending weekly meetings, supporting award applications and helping students revise their papers. Wray is also very supportive of me attending academic conferences to learn about the latest hot research topics. As a great professor, he is always willing to help his students and devote himself to training good researchers for the future in the field of machine learning. In short, he is a great professor and supervisor.

I was touched by François's patience with his students, his attention to detail and his fancy professional skills. When I started my PhD, I did not have the skills to be an independent researcher, but he gave me enough time to learn and master these skills little by little. He always knows how to explain a complicated idea to me in the simplest way. He would patiently help me write down the details of the algorithms on paper. He showed me how to make plans and the importance of them. He taught me how to write academic papers, no matter which language I use. He helped me run experiments on clusters when I was struggling with large datasets. When my

paper was rejected, he encouraged me that my research was a high standard of work. Whenever I am at a loss about practical problems, he always helps me solve them easily. He just showed me an excellent example of a patient supervisor, and if I have the chance to supervise any students in the future, I must be more patient like him.

I was so lucky to have them as my supervisors. Their enthusiasm for new knowledge, patience and support for students, and pursuit of excellence all set a good example for me to continue to engage in scientific research in the future. Their love for their work makes me think they are working for the advancement of human technology, not just for personal benefit. They are real researchers.

Second, I want to thank my husband, Pengyu Li, for his sacrifice to our family and his support for my studies. During my PhD study, our daughter was born. In order to let me concentrate more on my study, he suspended his career to helping me take care of our daughter and take on more family responsibilities. I also want to thank my little girl Hannah Li. She always likes to show me her magic, which gives me infinite joy.

I would also like to thank Julie Holden, who gave me much help in preparing the milestone presentations. Thanks to Dr Shirui Pan and Dr Ming Liu for encouraging me to be a teaching tutor during the PhD study. Thank my "adopted" sister, Yien Duan, who always encourage me when I lose confidence.

The person I want to thank most is myself. The doctoral study is not only the stage of learning to do scientific research but also the period of continually knowing the true self. I met many difficulties in this process, but instead of giving up, I have been trying hard to overcome them and step out of my comfort zone. Thank me for trying to balance life and study, even though it is not easy. The study and life experience during the doctoral study period made me a stronger person and a dedicated researcher.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

This chapter aims to stress the importance and feasibility of this study. First, we show the motivation of this research by giving some practical examples seeking to answer the question of why class probability estimation is essential. Second, we define the research problem by narrowing down the motivation in a more formal perspective. Third, we show the feasibility of this research by reviewing and evaluating relevant literature, defining the research gap that needs to be filled and the research questions that need to be answered. The research method for each of the research questions is proposed and discussed. Fourth, we clearly define the research purpose and the contribution of this study. The last section lists the structure for the rest of the thesis.

1.1 Research Motivation by Examples

In recent years, machine learning classification techniques have achieved great success in many interdisciplinary areas, such as medical diagnosis (Erickson et al., 2017), spam detection (Crawford et al., 2015), traffic prediction (Rzeszótko and Nguyen, 2012), recommendation systems (Singhal et al., 2017), bioinformatics (Li et al., 2020) and political elections (Ramteke et al., 2016). To naturally propose the research motivation of this paper, we first give a few specific real-world examples as follows.

Example 1: Early Bushfire Detection. Australia is currently suffering from devastating bushfires across the country that have captured the world's attention. The massive fires, which lasted from 2019 to early 2020, killed at least 33 people, destroyed more than 2,700 homes and killed an

estimated 1.25 billion native animals. These fires are a threat to both humans and wildlife.

Early estimates of future bushfire hotspots can help authorities better manage risks, allocate resources and reduce losses. Nowadays, while patrols, monitoring towers and satellites have long been used, a surging number of researchers rely on machine learning classification algorithms to improve the detection rate by analysing weather and satellite data. More formally, this example can be defined as a typical binary classification problem in machine learning field to predict whether a location will be a hotspot or not.

However, knowing how likely a hotspot is to burn in the future is much more important than just predicting whether it is a hotspot or not, because based on these probabilities, we can allocate more resources and attention to places with higher probabilities when resources are limited and time is urgent. To conclude, for many binary classification tasks, we need to accurately predict how likely a test sample is to be positive.

Example 2: Political Election Prediction. During the 45th U.S. presidential election in 2016, many research institutes relied on machine learning techniques to predict whether Hillary Clinton or Donald Trump would win. Although Clinton won several major states in the election and was widely predicted to win, the world was shocked after the result was announced. An Indian firm called MogIA shot to fame after predicting Mr Trump's victory (Post, 2016).

MogIA owes much of its success to the 20 million data points it collected from various online social network platforms such as Google, YouTube and Twitter, including comments, news, ads and videos, to make predictions. These data points require sentiment analysis, which is the process of determining the attitude or opinion of the speaker or writer. In machine learning, it is usually defined as a classification problem where a classifier is fed with a text and returns a corresponding label, e.g. positive, neutral or negative.

Traditional sentiment analysis classifiers classify posts into three categories, but in reality, apart from some posts with obvious opinions, it is difficult to determine whether they are absolutely positive or negative, and the classification of neutrality varies according to the classifier. Only outputting the prediction category ignores the more detailed information hidden in the data. It is more appropriate to replace the predicted category with a probability value that belongs to the positive, as this probability can show

how reliable the predicted category is.

Research Motivation. The similarity between these two examples is that they can both be represented as classification tasks, and the predicted class probability estimates are more important than the most likely class labels because they can display more information about the prediction and allow domain experts to make better decisions.

This fact also applies to a variety of practical applications. For example, the weather forecast says that there is a "30% chance of rain" conveys more information than merely saying "It might rain tomorrow" (Gigerenzer et al., 2005). For HIV diagnosis, knowing a patient's chance of getting HIV can help doctors make better treatment because even when the probability is as low as 20%, we cannot assert that the patient is safe enough (Kéri et al., 2002; Jahanshahi et al., 2010). For an advertisement recommendation system, knowing the probability of customers' interest on an item can better decide whether to recommend the product or not (Chen and Canny, 2017; Li et al., 2005). For cost-sensitive classification tasks, accurate class probability estimates are particularly critical to minimise the total misclassification cost (Margineantu, 2002; Elkan, 2001).

The motivation of this thesis is to improve the class probability estimation of classifiers with proper machine learning techniques. We believe that based on the accurate class probability estimates, domain experts can make better decisions. The improved class probability estimation for classifiers can benefit many fields requiring predictive data analysis. In the following section, we will give a more formal definition of the research problem.

1.2 Research Problem

Although accurate class probability estimation is desired in many practical applications, not all classification models are naturally probabilistic to provide accurate class probability estimates. There are two main reasons. First, they are designed to generate accurate predictions of class categories rather than probability estimates. For example, the explicit goal of the decision tree classifier is to generate homogeneous (pure) leaves with probabilities close to zero or one cause the estimates are highly biased. Second, probability estimates are unreliable because of the data sparsity problem. For example, the zero probability problem frequently occurs in Bayesian network classifiers.

The number of parameters that need to be estimated increases significantly with the complexity of the network structure, while the amount of data used to estimate each parameter is small, resulting in high variance. In the extreme case, the class probability is likely to be zero because it is the product of many conditional probabilities. If any of them have no data, then the frequency result is also zero.

More sophisticated techniques are needed to improve the class probability estimation of classifiers. Ensemble learning is an effective way to get better probability estimates. The most notable benefit of ensemble learning is to reduce the variance of probability estimates without increasing bias by training a set of diverse learners to solve the same problem and averaging over them (Zhou, 2012, 2015; Arias et al., 2018). Just as Chawla (2006) pointed out, ensembles of decision trees, such as Bagging (Breiman, 1996) and Random Forest (Breiman, 2001; Genuer, 2012), significantly improve the quality of the probability estimates produced at the decision tree leaves. There are also some ensemble models build on Bayesian network classifiers proposed (Webb et al., 2005; Duan and Wang, 2017; Arias et al., 2018), but their performance is inferior to that of random forest.

As discussed by Provost and Domingos (2003) and others, another popular way of improving the probability estimates is to make these estimates smoother, i.e. to adjust them to be less extreme by probability smoothing techniques (Zadrozny and Elkan, 2001b). Smoothing is a technique used to better estimate probabilities when there is insufficient data to estimate probabilities (Wang et al., 2003). Not only do smoothing methods generally prevent zero probabilities, but they also attempt to improve the accuracy of the models as a whole (Zhai and Lafferty, 2004). Laplace smoothing (Provost and Domingos, 2003) and M-estimation (Zadrozny and Elkan, 2001b), as the two most popular and simple methods, have long been used to improve the estimates of Bayesian network classifiers Jiang et al. (2007); Burge and Lane (2007); Cherian and Bindu (2017) and decision trees (Zadrozny and Elkan, 2001b; Chawla and Cieslak, 2006; Chawla, 2005).

Until recently, a few recent studies show that hierarchical probability smoothing methods are more suitable for hierarchical tree structural classifiers that need to improve the probability estimation of leaf nodes, such as for n-grams (Shareghi et al., 2017b), decision trees (Ferri et al., 2003) and Bayesian network classifiers where the conditional probability of an attribute

can be represented by a tree (Petitjean et al., 2018). Unlike Laplace and M-estimation smoothing, the hierarchical approaches incorporate the parent nodes' estimates into the calculation of the leaf nodes' estimates, rather than based solely on the observations at leaf nodes locally. The intuition behind this is that the observations on leaves are continuous partitions of the entire data from the root, so probability estimates on leaves should not be independent, but should somehow correlate their parents and siblings.

Research Problem. To summarise, the class probability estimation of Bayesian network classifiers and decision trees are vital, but naturally unreliable and need improvement. Many studies adopted ensemble learning and probability smoothing techniques to improve the probability estimation of Bayesian network classifiers and decision trees. Recent studies found out that hierarchical probability smoothing techniques are more suitable for these two classifiers because of their hierarchical tree structure. How to improve the class probability estimation of Bayesian network classifiers and decision trees by more efficiently adopting ensemble learning and hierarchical probability smoothing techniques is the research problem of this thesis.

1.3 Research Gaps, Questions and Methods

As we have discussed above, class probability estimation of Bayesian network classifiers and decision trees needs to be improved, and hierarchical smoothing and ensemble learning techniques have been shown to help improve the class probability estimation of classifiers. Although some research has been done, there is still much room for improvement. In this section, we analyse the shortcomings of existing research and define research gaps we can fill to improve the probability estimation of these two classifiers further. The gaps, associated research questions and methods for both Bayesian network classifiers and decision trees are defined as follows.

1.3.1 Gap 1: Building A Diverse Ensemble Model of SKDB

Research Gap. It is necessary to propose a powerful ensemble model of Bayesian network classifiers for three reasons. First, Random forest is an ensemble model of decision trees and have got success in many fields, but we do not have an ensemble model on Bayesian network classifiers that can compete with Random Forest. Second, Bayesian network classifiers can

analyse large datasets with a small computer because of their out-or-core nature, i.e. no need to save data in main memory. They only need to go through the data some times to collect sufficient statistics, but Random Forest needs to hold all the data in the main memory for analysing. Last, SKDB is the latest single Bayesian network classifier with good classification accuracy and efficiency. Although there are some ensemble models of Bayesian network classifiers, such as AODE, AnDE, and KDF, none of them is built on the SKDB model. To conclude, it would be interesting to build a powerful ensemble model on SKDB (ESKDB) to compete with Random Forest and XGBoost with the ability to deal with both categorical and numerical data.

Research Question. Therefore, the research question is how would one build a robust ESKDB (ensemble model of multiple SKDB classifiers) model that can compete with Random Forest and the default XGBoost. Based on our knowledge, the secret of a good ensemble model is its diversity, which means the base classifiers should have low correlation and should be as different as possible. Thus, the research question can be narrowed down to how to increase the diversity of the ESKDB model to encourage each SKDB classifier to have its own network structure.

Research Method. In ensemble learning, one of the key techniques to increase diversity is to introduce randomness. The Random Forest algorithm is a powerful ensemble model because of its two sources of randomness: the use of Bagging technique to randomly generate training samples and the random selection of split attributes for each internal node. Inspired by random forest, we considered two sources of randomness when building ESKDB.

- **Method 1: Building ESKDB by sampling attribute orders.** The SKDB algorithm determines the attribute order of the network structure by first calculating the mutual information (MI) between each attribute and the target and then sorting them in descending order. The order is unique and the best. Instead of using a unique order, we can try to get different orders and then build an SKDB classifier for each of them, where different orders can be generated by first normalising the MI values into a multinomial distribution and then sampling at random multiple times.

- **Method 2: Building ESKDB by sampling parent orders**. The parent order for each attribute in the network structure of SKDB is

decided by calculating the conditional mutual information (CMI) for each pair of attributes with the class. Similarly to the above method, we can try randomly select the parent orders for each attribute by normalising the CMI into a distribution and sample from it.

1.3.2 Gap 2: Strengthen ESKDB for Handling Numerical Data

Research Gap. Although SKDB is a good Bayesian network classifier, it cannot handle data with numerical attributes that need to be discretised before learning classifiers. The ESKDB model we proposed is an ensemble model composed of many different SKDB classifiers with their own network structures, which cannot process numerical data either. Therefore, both SKDB and ESKDB need to extend their ability to handle continuous attributes. Thus, there is no need for discretisation prior to learning, and users do not have to bother about the data format.

Research Question. The research question is how to embed the existing attribute discretisation method into the learning process of ESKDB so that ESKDB can deal with both discrete and numerical data. An extended research question is whether this discretisation process can further increase the diversity of each SKDB classifier in ESKDB, so as to improve the performance of ESKDB further.

Research Method. The general idea of numeric attribute discretisation is to find the point with the maximum information gain among all intermediate points satisfying certain conditions as the optimal cutting point. These intermediate points are obtained by arranging attribute values in ascending order and calculating the average values between each pair of two adjacent points. Based on the comparison of the other points with this optimal point, divide the points into two subsets and for each subset repeat the above procedure until some stop condition is met. Merely executing this process before learning the network structure of the SKDB classifier makes it easier to enable SKDB with the ability to handle continuous attributes.

We can make full use of the data discretisation process of numerical attributes to further increase the diversity of different SKDB classifiers in the ESKDB model by making each SKDB classifier have its own cut point for numerical attributes. In particular, instead of choosing the optimal point with the maximum information gain, we can normalise the information gain values

for the points that satisfy particular condition into a probability distribution and randomly sample for cut points from the distribution. This is another source of randomness, which makes the ESKDB algorithms more diversified and can further improve the classification performance of ESKDB.

1.3.3 Gap 3: Better Probability Smoothing for Trees

The improvement of both the decision tree and the Bayesian network classifier can be regarded as the improvement of the tree structure classifier because the conditional probability of each attribute in the Bayesian network structure can be represented by a tree with the depth as the number of parent attributes. This part analyses the research gap of probability smoothing for tree classifiers and defines the research questions and methods accordingly.

Research Gaps. HDP has been shown to help Bayesian network classifiers achieve state-of-the-art estimates, but this approach has not yet been introduced into decision trees. The first gap is to analyse whether HDP can also help the decision tree improve probability estimation, but before that, it is necessary to fill the gap of improving the efficiency of HDP, since Petitjean et al. (2018) points out that HDP is computation-intensive. The third gap is to propose new and more efficient hierarchical smoothing methods inspired by existing studies.

Research Question. In this part, the main research question is how to get a better probability smoothing method for a tree classifier, which is divided into the following three sub-questions. First, how would one improve the efficiency of HDP smoothing? Second, compared with the standard smoothing and pruning methods, does HDP smoothing work well for decision trees and why? Third, can we propose another hierarchical smoothing method for decision trees that is faster than HDP but does not lose accuracy?

Research Methods. The research methods are listed as follows.

- **Method 3.1: Optimisation of HDP.** The primary purpose of HDP smoothing is to optimise the concentration parameters, which can be sped up by using a better prior to the concentration parameters.

- **Method 3.2: HDP smoothing on decision trees.** In this method, we first apply the improved HDP smoothing technique to decision trees, then conduct experiments to compare the probability estimation of

HDP with Laplace correction and M-estimation. Probability pruning and random forest could be compared with the HDP estimates.

- **Method 3.3: Novel smoothing algorithm for trees.** HDP smooths the leaf estimates to the upper parent node recursively until reaching the root node. Compared with HDP, a new method can smooth the leaf estimates to all the ancestor nodes along the branch that containing the leaf and the root. Discovering proper loss function and conducting gradient descent on this loss for the new algorithm is essential.

1.3.4 Gap 4: Smoothing on Ensemble Models

Research Gap. As an ensemble model of decision trees, the Random Forest algorithm can improve the probability estimates of a single tree significantly. Although some previous studies discovered that ensemble models do not need any smoothing (Boström, 2012), it would be of interest to know whether the newly proposed ESKDB model needs smoothing and the reason behind it. An in-depth analysis of guidance on applying smoothing to ensemble models is critical for future studies.

Research Question. How would one make the newly proposed ESKDB model more accurate? Does Random Forest need smoothing and why? What is the guidance of applying smoothing to ensemble models?

Research Method. The proposed method to fill this gap is to apply the current and newly proposed probability smoothing techniques to the Random Forest and ESKDB classifiers, compare the experiment results of ensemble models with and without smoothing techniques, and analyse the impact and give guidance of introducing probability smoothing techniques to ensemble models.

1.4 Research Purpose and Contribution

The purpose of this study is to investigate machine learning techniques that can be used to improve the class probability estimation of Bayesian network classifiers and decision trees. We also want to examine the impact of applying probability smoothing methods to ensemble models. The hypothesis is that hierarchical probability smoothing techniques are more suitable for

hierarchically structured classifiers, such as Bayesian network classifiers and decision trees, and together with ensemble learning techniques the variance of the probability estimates of classifiers could be significantly reduced.

Therefore, we expect to propose new ensemble models and hierarchical smoothing algorithms that can achieve the research purpose, provide insights for establishing powerful ensemble models, and give guidance on applying smoothing to ensemble models. We believe this study will be a significant contribution to applications that require highly accurate estimates of class probabilities, such as cost-sensitive classification, outlier detection, ranking and recommendation systems.

1.5 Thesis Structure

In this section, we provide an outline of the rest of the thesis. The outline and summary of each chapter are as follows:

- **Chapter 2: Literature Review on Improving Probability Estimation.** This chapter provides a thorough overview of the foundations for the research described in this thesis, including the bias-variance analysis of classifiers and the ways to control the variance of classifiers, such as probability smoothing and ensembling techniques. Proper scores for evaluating the accuracy of probability estimates are also introduced, such as RMSE, Brier score, cross-entropy and KL divergence. We also shed light on the significance test on Win-Draw-Loss results, the Leave-One-Out Cross-Validation technique and the numerical attribute discretisation methods used in this research.

- **Chapter 3: Literature Review on Bayesian Network Classifiers and Decision Trees.** This chapter introduces the background knowledge of Bayesian network classifiers and decision trees, as well as the existing Bayesian network classifiers and the works on improving class probability estimation of decision trees. We also illustrate common metrics used to measure the performance of classification algorithms

- **Chapter 4: ESKDB Algorithm.** In this chapter, we present our ESKDB algorithm, which is a novel ensemble model of Bayesian network classifiers with better parameter estimation incorporated with both ensembling and hierarchical probability smoothing techniques.

ESKDB combines three main components: (1) an effective strategy to vary the networks that are built by single classifiers (to make it an ensemble), (2) a stochastic discretization method which allows to both tackle numerical data as well as further increases the variance between different components of our ensemble and (3) a superior probability smoothing technique to ensure proper calibration of ESKDB's probabilities. We report experimental results on 72 datasets showing ESKDB's competitiveness with state of the art.

- **Chapter 5: HGS Algorithm.** In this chapter, we introduce the HGS algorithm proposed by us aiming to improve the class probability estimation of decision trees. We first apply a recent advanced smoothing method called Hierarchical Dirichlet Process (HDP) to trees, and then propose a novel hierarchical smoothing approach called Hierarchical Gradient Smoothing (HGS) as an alternative. HGS smooths leaf nodes up to all the ancestors, instead of recursively smoothing to the parent used by HDP. HGS is made faster by efficiently optimising the Leave-One-Out Cross-Validation (LOOCV) loss measure using gradient descent, instead of sampling used in HDP. An extensive set of experiments conducted on 143 datasets are reported in this chapter showing that our HGS estimates are not only more accurate but also do so within a fraction of HDP time. Besides, HGS makes a single tree almost as good as a Random Forest with ten trees. For applications that require more interpretability and efficiency, a single decision tree plus HGS is more preferred than Random Forest.

- **Chapter 6: Conclusions and Future Work.** This chapter summarises the main conclusions in this thesis and highlights the potential directions for future work.

Chapter 2

Literature Review on Improving Probability Estimation

In this chapter, we review the existing techniques used to improve the probability estimation of classifiers and the reasons. First, we introduce the fundamental bias-variance tradeoff of classifiers and explain why decision trees and Bayesian network classifiers need to control the variance. Second, simple and advanced hierarchical probability smoothing techniques are introduced as the primary method to control variance. Third, the ensembling technique, another way to reduce the classifier variance, is introduced. Fourth, we introduce the proper scores for evaluating the accuracy of the class probability estimates of classifiers. Finally, we introduce significance testing, cross-validation and discretisation techniques because they play a vital role in the new variance control algorithms we proposed in this thesis.

2.1 The Bias-Variance Analysis of Classifiers

In this section, we first introduce the bias-variance tradeoff of classifiers. After that, we analyse the bias and variance of decision trees and Bayesian network classifiers and propose the necessity of controlling the variance for them. Finally, the mainstream techniques that can control the variance of these classifiers are summarised.

2.1.1 The Bias-Variance Tradeoff

The bias-variance tradeoff (Geman et al., 1992), also known as the bias-variance dilemma or the bias-variance problem, is a well-known problem in statistics and machine learning. It means that the bias and variance of the model's parameter estimation cannot be minimised at the same time. A low-biased model tends to have high variance, and vice versa.

The bias-variance tradeoff is an important concept for analysing the performance of learning algorithms. **Figure 1** shows the tradeoff between bias, variance and model complexity. It can be seen from this figure that when the model is simple (learned on limited data), the model tends to have low variance and high bias. In contrast, as the model becomes more complicated (fitted with more training data), the model tends to be high-variance and low-bias. The sweet spot for any model is the optimum model complexity at which the increase in bias is equivalent to the reduction in variance, and the total error is minimised.

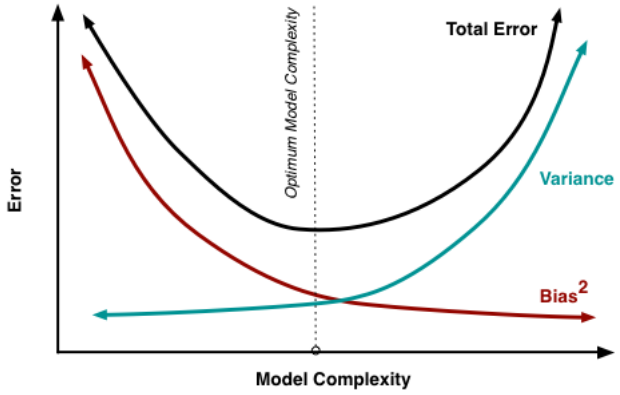

Figure 1: The figure of bias-variance tradeoff with the model complexity (Fortmann-Roe, 2012).

2.1.2 The Bias-Variance Decomposition of the Squared Loss

Assuming that we have a dataset $\mathcal{X}$ with $\vec{X}$ being the predictive attributes and Y being the target. For regression tasks, the target Y is a numerical value. We want to find the true mapping function f from $\vec{X}$ to Y such that

$$Y = f(\vec{X}) + \epsilon, \tag{1}$$

where ϵ is the noise term. Now we want to learn a model $h(\vec{X})$ such that it approximates $f(\vec{X})$ as well as possible. If the "as well as possible" is measured by squared loss, it means we want $(Y - h(\vec{X}))^2$ to be minimal both for the training data and any unseen test examples. The expected squared error between the true target value and the prediction is

$$\begin{aligned} Err(\vec{X}) &= E[\left(Y - h\left(\vec{X}\right)\right)^2] \\ &= E[(f(\vec{X}) + \epsilon - h(\vec{X}))^2] \\ &= (E[h(\vec{X})] - f(\vec{X}))^2 + E[(h(\vec{X}) - E[h(\vec{X})])^2] + \sigma^2 \\ &= bias^2 + variance + irreducible\ error. \end{aligned} \tag{2}$$

We can see from this formula (James et al., 2013) that the expected error is decomposed into three parts: $bias^2$, variance and irreducible error. The bias error measures the difference between the correct value $f(\vec{X})$ which we are trying to predict and the average prediction of the learned model $E[h(\vec{X})]$. It depends on the choice of the learning algorithm among enormous hypotheses. Variance is the variability of the model prediction for a given test example. It is only related to the learning algorithm $h(\vec{X})$ itself and not to the true function $f(\vec{X})$. Irreducible error measures the noise in our data. Regardless of which algorithm we choose or how good the model is, the term cannot be reduced or eliminated. To minimise the expected error, both bias and variance should be minimised. Unfortunately, it is impossible to do both simultaneously.

A similar decomposition has been also derived for classification (Geurts, 2009; Pedro, 2000). Classifiers, such as Decision trees and Bayesian network classifiers, have been mentioned in many studies to suffer from high variance problem (Han, 2011; Boström, 2012; Petitjean et al., 2018). The main reason is the data sparsity problem, a term used to describe the phenomenon of not observing enough data to learn a parameter. If the amount of data used to estimate the parameter is small, then the calculated probability estimate is unreliable and cannot represent the real data distribution. This problem often occurs even in big data classification tasks, because big data tends to produce more complex models, resulting in a large number of parameters that need to be estimated, but for each parameter the amount of data is small.

To conclude, the probability estimation of classifiers, such as decision tree and Bayesian network classifier, is not reliable and need improvement.

Usually, if we can get enough data, we can improve the probability estimates. But in reality, we run into the problem of data sparsity problem. Other machine methods are needed to solve the problem, which will be introduced in the next section.

2.1.3 Ways to Control Variance

There are two effective methods to help reduce the variance of probability estimation for classifiers. Probability smoothing techniques reduce variance by adding appropriate priors to the observed relative frequencies, thus making the estimates "smoother". The ensemble learning techniques reduce the variance by building many high-variance base classifiers and averaging them. In the next sections, we will review each of these learning techniques in detail.

2.2 Probability Smoothing

In this section, we describe the probability smoothing techniques commonly used to improve the parameter estimation of classifiers. First, we introduce the Maximum Likelihood Estimation (MLE) method, which produces unreliable estimates by taking the observed relative frequencies as probability estimates. The idea of smoothing is to make the MLE estimates "smoother" by adding an appropriate prior probability. After summarising the existing smoothing methods, we find that they all use the Dirichlet distribution as the prior probability, but with different parameters for the distribution. Therefore, before introducing the existing methods, we add an introduction to the Dirichlet distribution.

2.2.1 Maximum Likelihood Estimation

Suppose we have a set of N data points $\mathcal{X} = \{(\overrightarrow{x_1}, y_1), (\overrightarrow{x_2}, y_2), ..., (\overrightarrow{x_n}, y_n)\}$ belonging to K different classes, and $n_1, n_2, ...n_K$ are the counts for each of class in $\mathcal{X}$, then the joint distribution of $\mathcal{X}$ is a multinomial distribution. The joint probability mass function is

$$\begin{aligned} P(\mathcal{X}|\overrightarrow{\theta}) &= P(n_1, ..., n_k|\theta_1, ..., \theta_k) \\ &= \frac{N!}{\prod_{k=1}^{K} n_k!} \prod_{k=1}^{K} \theta_k^{n_k}, \end{aligned} \tag{3}$$

where $\vec{\theta}$ is the probability vector with $\sum_{k=1}^{K} \theta_k = 1$, among which θ_k is the probability of class k. N is the total count and equals to $\sum_{k=1}^{K} n_k$.

In the case of discrete distributions, the likelihood is a synonym for the joint probability of the data so that $L(\vec{\theta}|\mathcal{X}) = P(\mathcal{X}|\vec{\theta})$. For different values of parameters, the likelihood of the data will be different. If the correct parameter estimates are obtained, the likelihood should be maximised in the limit of infinite data. The procedure of getting these optimum parameter estimates is called Maximum Likelihood Estimation (MLE). The MLE estimate is defined as

$$\begin{aligned} \vec{\theta}_{MLE} &= \underset{\vec{\theta}}{\operatorname{argmax}} P(\mathcal{X}|\vec{\theta}) \\ &= \underset{\vec{\theta}}{\operatorname{argmax}} \frac{N!}{\prod_{k=1}^{K} n_k!} \prod_{k=1}^{K} \theta_k^{n_k}. \end{aligned} \tag{4}$$

This equation is a product of many probability numbers, each of which is a value between 0 and 1. In practice, a log-likelihood function is often used by taking the natural logarithm of the likelihood function. Therefore, **Equation 4** is equal to maximizing the logarithm of the joint probability mass function, as shown in **Equation 5**.

$$\begin{aligned} \vec{\theta}_{MLE} &= \underset{\vec{\theta}}{\operatorname{argmax}} \log P(\mathcal{X}|\vec{\theta}) \\ &= \underset{\vec{\theta}}{\operatorname{argmax}} \log \left(\frac{N!}{\prod_{k=1}^{K} n_k!} \prod_{k=1}^{K} \theta_k^{n_k} \right) \\ &= \underset{\vec{\theta}}{\operatorname{argmax}} \log \prod_{k=1}^{K} \theta_k^{n_k} \\ &= \underset{\vec{\theta}}{\operatorname{argmax}} \sum_{k=1}^{K} n_k \log \theta_k. \end{aligned} \tag{5}$$

Equation 5 is maximized when for each class k, it has

$$\theta_k = \frac{n_k}{N}. \tag{6}$$

The MLE may result in a zero probability estimate when the count n_k equals 0. For a Bayesian network classifier, the joint probability is zero when any term in the product is zero. For the decision tree, any test example that falls into a leaf node with an estimated class probability of zero will

receive a zero score. Even if the n_k count is small, this problem can decrease the classifier's accuracy and ranking performance. We should not use MLE directly to obtain parameter estimates. Probability smoothing technique can be used to solve the zero probability problem. Maximum posterior (MAP) estimation, as the basis for smoothing, is covered in the next section.

2.2.2 Maximum A Posteriori Estimation

In Bayesian statistics, a maximum a posteriori (MAP) estimate is an estimate of a parameter of a model using the MAP estimation. MAP is a regularisation method of MLE with a prior over the parameters. It maximises the posterior distribution based on Bayes rule.

$$P(\vec{\theta}|\mathcal{X}) = \frac{P(\mathcal{X}|\vec{\theta})P(\vec{\theta})}{P(\mathcal{X})}, \tag{7}$$

where $P(\mathcal{X}|\vec{\theta})$ is the likelihood and $P(\vec{\theta})$ is the prior probability over $\vec{\theta}$. $P(\mathcal{X})$ is a normalizing constant value, and can be eliminated. The MAP estimate in log domain becomes

$$\begin{aligned} \vec{\theta}_{MAP} &= \underset{\vec{\theta}}{\operatorname{argmax}} \Big\{ \log \frac{P(\mathcal{X}|\vec{\theta})P(\vec{\theta})}{P(\mathcal{X})} \Big\} \\ &= \underset{\vec{\theta}}{\operatorname{argmax}} \Big\{ \log P(\mathcal{X}|\vec{\theta})P(\vec{\theta}) \Big\} \\ &= \underset{\vec{\theta}}{\operatorname{argmax}} \Big\{ \log P(\mathcal{X}|\vec{\theta}) + \log P(\vec{\theta}) \Big\}. \end{aligned} \tag{8}$$

The only difference between $\vec{\theta}_{MAP}$ and $\vec{\theta}_{MLE}$ is that there is a prior term $P(\vec{\theta})$ in $\vec{\theta}_{MAP}$, otherwise they are identical. It means the likelihood is now weighted with some weight coming from the prior.

When the prior probability is uniformly distributed, MAP and MLE are identical because the term $\log P(\vec{\theta})$ in the above equation becomes a constant. The probability estimates of MAP varies with the selection of different prior probabilities. How to select the proper prior probability is critical. In the following section, we will introduce the Dirichlet prior and the probability smoothing techniques derived from it.

2.2.3 Smoothing with Dirichlet Priors

In probability and statistics, the Dirichlet distribution, often denoted $Dir(\vec{c})$, is a family of continuous multivariate probability distributions parameterised by a vector $\vec{c}$ of positive reals. It is a multivariate generalisation of the beta distribution. Dirichlet distributions are commonly used as prior distributions in Bayesian statistics (Blei et al., 2003; Saputro et al., 2017). The probability density function is

$$P(\vec{\theta}\,|\,\vec{c}) = \frac{1}{B(\vec{c})} \prod_{k=1}^{K} \theta_k^{c_k - 1}, \tag{9}$$

where $\sum_{k=1}^{K} \theta_k = 1$ and $0 \leq \theta_k \leq 1$ for $k \in K$. The normalizing constant $B(\vec{c})$ is the multivariate beta function, which can be expressed in terms of the gamma function:

$$B(\vec{c}) = \frac{\prod_{k=1}^{K} \Gamma(c_k)}{\Gamma(\sum_{k=1}^{K} c_k)}. \tag{10}$$

The mean of a $Dir(\vec{c})$ is

$$E(\theta_k|c_k) = \frac{c_k}{\sum_{k=1}^{K} c_k}. \tag{11}$$

A common special case is the symmetric Dirichlet distribution, where all of the elements in the vector $\vec{c}$ have the same value. The symmetric case might be useful, for example, when a Dirichlet prior over components is called for, but there is no prior knowledge favouring one component over another. Since all elements of the parameter vector have the same value, the symmetric Dirichlet distribution can be parametrised by a single scalar value c, called the concentration parameter. In terms of c, the density function has the form

$$P(\vec{\theta}|c) = \frac{\Gamma(cK)}{\Gamma(c)^K} \prod_{k=1}^{K} \theta_k^{c-1}, \tag{12}$$

In fact the Dirichlet distribution is the conjugate prior of the categorical distribution and multinomial distribution. Assuming the likelihood $P(\mathcal{X}|\vec{\theta})$ is a multinomial distribution and the parameters $\vec{\theta}$ have a $Dir(c)$ prior, the

posterior probability of $\vec{\theta}$ given data $\mathcal{X}$ and c is

$$\begin{aligned} P(\vec{\theta}|\mathcal{X}, c) &= \frac{P(\mathcal{X}|\vec{\theta}) \cdot P(\vec{\theta}|c)}{P(\mathcal{X})} \\ &\propto P(\mathcal{X}|\vec{\theta}) \cdot P(\vec{\theta}|c) \\ &= \frac{N!}{\prod_{k=1}^{K} n_k!} \prod_{k=1}^{K} \theta_k^{n_k} \cdot \frac{\Gamma(cK)}{\Gamma(c)^K} \prod_{k=1}^{K} \theta_k^{c-1} \\ &\propto \prod_{k=1}^{K} \theta_k^{n_k+c-1}. \end{aligned} \tag{13}$$

In other words, the posterior is also a Dirichlet distribution, which means that the Dirichlet distribution is a conjugate distribution to the multinomial distribution. The mean for the new Dirichlet distribution $Dir(n_k + c)$ is

$$E(\theta_k) = \frac{n_k + c}{N + Kc}. \tag{14}$$

2.2.3.1 Laplace Smoothing: $Dir(1)$ Prior

Laplace smoothing (Provost and Domingos, 2003) is a basic and simple smoothing method that could improve probability estimates. It smoothes the likelihood by adding a Dirichlet distribution prior with $c = 1$. According to **Equation 14**, the expected estimate for θ_k becomes

$$\theta_k^{Lap} = \frac{n_k + 1}{N + K}. \tag{15}$$

It can also be explained by adding one data point to each class to make the estimates less extreme. In addition, the zero probabilities will be turned into non-zero ones. However, in practice adding one to each class count may cause over-smoothing of the probability estimates, especially when the two classes are far from equiprobable (Zadrozny and Elkan, 2001a).

2.2.3.2 M-estimation: $Dir(\frac{m}{K})$ Prior

M-estimation (Zadrozny and Elkan, 2001b) is another smoothing method of the Dirichlet prior with $c = \frac{m}{K}$. The estimation formula of **Equation 14** then becomes to

$$\theta_k^{M-esti} = \frac{n_k + m \cdot \frac{1}{K}}{N + m}, \tag{16}$$

where $\frac{1}{K}$ is a uniform base rate and m is a parameter that controls how much scores are shifted towards uniform distribution. Usually m is optimized by conducting a cross validation experiment on the validation set. Laplace smoothing is a special case of M-estimation with the $m = K$.

2.3 Hierarchical Smoothing

Hierarchical smoothing methods also use a Dirichlet prior to smooth the probability estimates of a leaf node, except that the prior comes from the probability distribution of the parent node, which is a Dirichlet distribution. It assumes that the probability estimates on a leaf node depend on the parent node's estimates. In this section, we first emphasise the advantages of hierarchical smoothing methods over simple methods such as Laplace and M-estimation. Then we introduce the existing hierarchical smoothing methods for decision trees and Bayesian network classifiers, including M-branch and HDP.

2.3.1 Hierarchical Smoothing: Low Variance and Low Bias

Hierarchical smoothing methods allows better control of variance while retaining the low bias compared with single-layer methods, such as Laplace and M-estimation, who calculate the estimates on the leaf nodes locally. **Figure 2** is a decision tree model for disease diagnosis, knowing that this disease is related to some rare gene G and sex, and it is more prevalent for females. Now we want to predict the probability of $p(disease|hasgene\&male)$. There is only one patient without disease at this leaf node. The probability estimate based on the observed relative frequency is 0, which means that male has the gene is unlikely to be sick, but this fails to consider the fact that 90.9% (calculated by $\frac{10}{(10+1)}$) of people who carry the gene are diagnosed with the disease (the parent estimate). This estimate also did not consider that there are only about 10% (calculated by $\frac{100}{(100+901)}$) people have the disease in our limited training data (the root estimate).

Single-layer smoothing methods, including Laplace and M-estimation, can improve the probability from 0% to 33% and 25% respectively, but they are still quite far away from 90.9%. They did not consider the parent nodes' class distribution. These estimates have high variance and low bias.

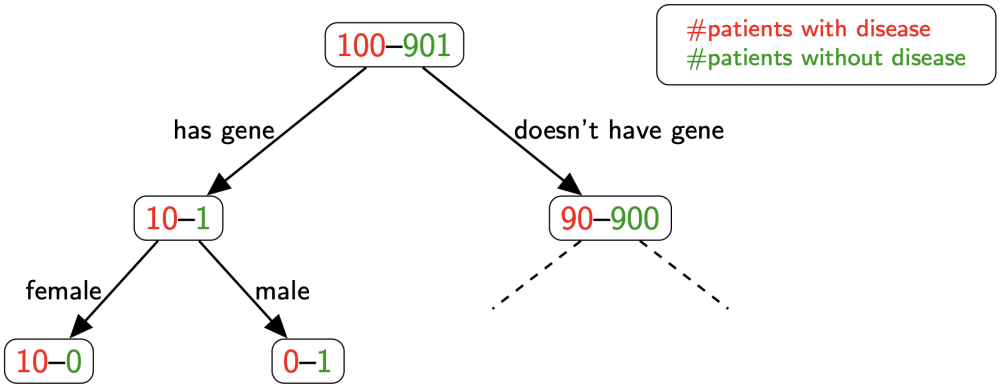

Figure 2: The tree structure for the medical dataset.

Compared with the single-layer smoothing methods, hierarchical smoothing methods can better control the variance of the estimates at leaves. Since the class probability estimate of the parent and ancestor nodes have a lower variance than those of the leaf nodes, and hierarchical methods take the parents and ancestors' estimates into consideration when calculating the class probability estimates at leaves, hierarchical smoothing is a better smoothing idea for tree-structured models than single-layer methods.

2.3.2 M-branch Smoothing

The M-branch smoothing method (Ferri et al., 2003) was first introduced to decision trees. It considers each leaf is a subsample of the upper parent, and parent also makes a subsample of the upper node, until the root is reached. This means that the sample used to obtain the probability estimates in a leaf is the result of many sampling steps, as many as the depth of the leaf. Then it is natural to consider all the history of samples when trying to obtain the probability estimates of a leaf.

Let $\vec{v} =< v_l, v_{l-1}, ..., v_2, v_1 >$ represents all the nodes on the branch that contains the leaf node v_l, where v_{l-1} is the parent node of l and v_1 is the root. Let $n_{l,k}$ denote the observed count of class k at node v_l, and N_l is the total count. Let $\hat{\theta}_{l,k}$ represent the class probability estimate for class k of node v_l. The M-branch method smoothes the leaf node estimate $\hat{\theta}_{l,k}$ to its

parent node $\hat{\theta}_{l-1,k}$ using the M-estimation method in the following way,

$$\hat{\theta}_{l,k}^{Mbranch} = \frac{n_{l,k} + m_l \cdot \hat{\theta}_{l-1,k}^{Mbranch}}{N_l + m_l}. \tag{17}$$

Here the base rate of M-estimation is the parent estimate $\hat{\theta}_{l-1,k}^{Mbranch}$, which also needs to be smoothed to the parent node at a higher level $\hat{\theta}_{l-2,k}^{Mbranch}$. Repeating these steps recursively to a higher parent node until the root node v_1 reached. The root node v_1 is smoothed to a uniform probability $\hat{\theta}_{0,k} = \frac{1}{K}$.

M-branch smoothing is a generalisation of M-estimation, with the base rate for each node being the immediate parent estimate. The m_i parameter for each node controls the degree of smoothness to its parent. m_i is assumed to be bigger when the node is closer to the leaf. m_i is defined as a function of the node height in the tree, and the overall training data size N. Let $h = d + 1 - j$ represents the node height, where d is the depth of the branch and j the depth of the node. The normalised height of a node is defined as $\Delta = 1 - 1/h$ in order to make the correction bigger the closer to the root. So for leaves, the height is 1, and the normalised height is 0. From here, the m_i value for parent node i is defined as

$$m_i = M \cdot \left(1 + \Delta_i \cdot \sqrt{N}\right). \tag{18}$$

Here M is a user-supplied constant value as in the M-estimation, which is set to be 4 in (Ferri et al., 2003). N is the size of the training data.

2.3.3 HDP Smoothing

In the M-branch smoothing method, the parameters are defined as a function based on node height h, the size of the training data N and a user-supplied count M. The purpose of this function is to ensure that the parameter is larger when the node is closer to the leaf node. We think this method of setting parameters is not well justified. Its starting point is to achieve a goal by assuming a specific relationship between parameters. This hypothesis is just a guess without solid theoretical support. There should be a more principled way for setting parameters. The HDP smoothing method uses the Dirichlet Process (DP) theory to set the parameters, which is described in detail below.

A Dirichlet Process $DP(c, G)$ is specified by a base distribution G and a

concentration parameter c (Teh, 2010). G is the expectation of the process, i.e., the DP draws distributions around the base distribution the way a normal distribution draws real numbers around its mean. c is a positive real number that controls how similar the sampled distribution is to the base distribution. A larger c means they should be more similar. The DP can also be seen as the infinite-dimensional generalisation of the Dirichlet distribution.

The Hierarchical Dirichlet Process (HDP) is a nonparametric Bayesian approach to clustering grouped data (Teh and Jordan, 2010). It uses a DP for each group of data, with the DPs for all groups sharing a base distribution which is itself drawn from a DP. The clusters at the base distribution are shared across all the groups. HDP models can be applied to decision trees, as shown in **Figure 3**.

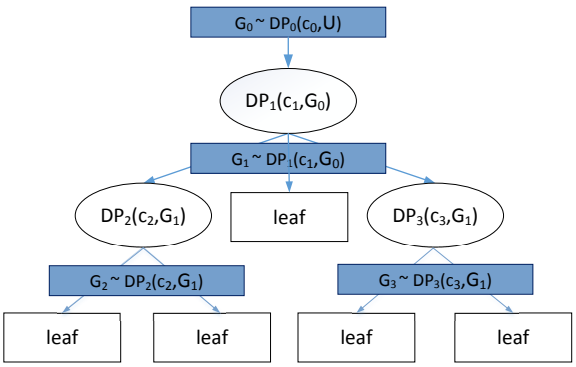

Figure 3: An HDP model tree. If each node in the tree is associated with a DP, the whole tree can be turned into an HDP model. The drawn distribution from the parent DP serves as the base distribution of its children. Specifically, the base distribution of the root node DP is uniform U. Each node in the tree contains a group of data that shares the same K classes.

Unlike single-layer smoothing methods that only smooth the probability at the leaves, HDP smoothing assumes that the conditional distribution of a leaf node in the hierarchical model is similar to its parent node and the sibling nodes who share the common parent to it. This is achieved using a hierarchical Dirichlet prior to all the parameters in the tree.

Unlike M-branch, HDP assumes that only leaf nodes have data and all the internal nodes are empty, but each leaf passes some "imagined" data to its parent conceptually. The passage of data goes higher recursively until the root node is reached. This is not actual data but prior counts for the

purpose of inference. The imagined data is generated during the sampling algorithm (Petitjean et al., 2018).

Suppose $t_{u,k}$ is the number of "imagined data with class k" that node u passes up to its parent node ϕ. These "imagined data" is a subset of the data at node u so it must meet the constrain of $t_{u,k} \leq n_{u,k}$. The data for the parent node ϕ is collected from all its children so that $n_{\phi,k} = \sum_{u \in child(\phi)} t_{u,k}$ where $u \in child(\phi)$ means u is the child of ϕ.

The HDP smoothing formula for node u and class k is defined recursively as follows

$$\hat{\theta}^{HDP}_{u,k} = \frac{n_{u,k} + c_\phi \cdot \hat{\theta}^{HDP}_{\phi,k}}{N_u + c_\phi}. \tag{19}$$

Here $\hat{\theta}^{HDP}_{\phi,k}$ is the parent estimate which also needs to be smoothed from its parent. The concentration c_ϕ affects how much data passes up to the ϕ, i.e. $t_{u,k}$. If we expect $\hat{\theta}_{u,k}$ to be very similar to $\hat{\theta}_{\phi,k}$, then choose a bigger c_ϕ that makes most of the data pass up and the parent probability contribute more to the estimate. If c_ϕ is small, the parent probability contributes less to the estimate.

The HDP smoothing for HDP model tree is in a top-down manner from the root to the leaves. It starts by smoothing the root node to a uniform distribution, where each class has the same probability estimate. It then smoothes the children nodes of the root using the root estimates as their priors. Repeat this process until all nodes are smoothed. Thus, when reaches the leaves, the probability estimates are already properly smoothed.

Equation 19 can also be explained by the Hierarchical Chinese Restaurant Process (CRP) (Teh and Jordan, 2010). The whole tree is like a multilevel restaurant offering the same K dishes. Data at leaf l with class k is like a customer in restaurant l choosing dish k. $n_{l,k}$ is the number of customers eating dish k in this restaurant and N_l is the total customers in the restaurant. Then a new customer comes in and can sit in an existing table serving dish k with probability $\frac{n_{l,k}}{N_l + c_\phi}$, or choose to eat upstairs with probability $\frac{c_\phi}{N_l + c_\phi}\hat{\theta}^{HDP}_{\phi,k}$. c_ϕ corresponds to the restaurant's attraction to the new customer. The customer can continually choose a restaurant upstairs until reaching the highest level, i.e. the root node.

Note that all the concentration parameters need to be sampled rather than using **Equation 18** as in M-branch. The sampling procedure aims to reconstruct the posterior distribution if enough sampling iteration is

permitted, but sampling takes time. HDP works better if the sibling nodes are more similar because it allows sharing across the siblings.

The two most critical parameters for the HDP smoothing in Bayesian network classifiers (Petitjean et al., 2018) are *iteration* and *tying*. *iteration* is the cycles that Gibbs sampling needs to sample the concentration parameters. *tying* strategy is to tie some nodes together to share the same concentration parameter to reduce the number of parameters that need to be sampled. Big trees using HDP tend to be very slow because there are many parameters to be sampled. In Petitjean et al. (2018), it shows that tying some nodes to make them share the same parameter value is more efficient. There are four types of tying. $SINGLE$ means tying all the nodes. $LEVEL$ means the nodes on the same depth are tied together. $PARENT$ means tying the sibling nodes under one parent. $NONE$ means no tying. The number of parameters is increasing for these four strategies.

HDP has proven to be very useful for language model smoothing (Shareghi et al., 2017a) and BNCs (Petitjean et al., 2018) recently. Please refer to (Petitjean et al., 2018) for more detail about HDP smoothing on BNCs. Moreover, the HDP smoothing package in Java is available on Github [1].

2.4 Ensemble Learning

Ensemble learning, as another popular way to reduce variance, is introduced in this section, including the background knowledge, ensemble techniques, existing ensemble models and the bias-variance decomposition of ensembles.

2.4.1 Background Knowledge

Ensemble learning, also known as multiple classifier systems, is a way to improve the performance of single learners by training a set of learners to solve the same problem and combining the performance of them to make predictions (Zhou, 2012). **Figure 4** shows a common ensemble architecture. Each $learner_i, i \in \{1, 2, ..., E\}$ in the ensemble model is called a base learner.

An ensemble is much stronger than a single learner for several reasons. First, ensemble methods can apply different information from the training data by combining some equally performing single learners. Second, ensembles

[1] https://github.com/fpetitjean/HDP

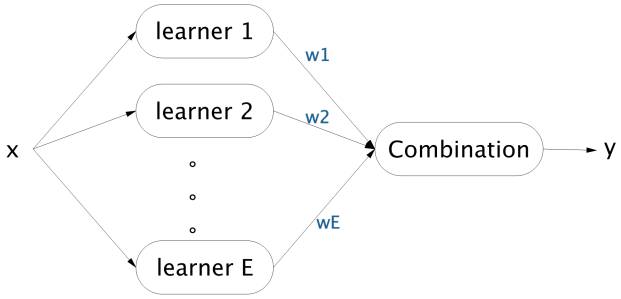

Figure 4: A common ensemble learning architecture.

are more likely to include a better hypothesis than a single learner. Third, ensembles can give better approximations to the real target function than single ones (Zhou, 2015).

An ensemble model can be learned in two steps in general. The first step is to select the base learning algorithms to learn the base learners, and the second step is to combine the performance of the base learners as an ensemble model.

2.4.2 Base Learning Algorithm Selection

For the first step, the base learning algorithms can be decision trees, Bayesian network classifiers, neural networks and other classification algorithms. The choice of base learning algorithms can be the one unique algorithm or a combination of different algorithms. The former is called homogeneous ensembles, and the latter is called heterogeneous ensembles (Zhou, 2015).

To build a good ensemble model, the base learners should be as accurate as possible and as diverse as possible. "As diverse as possible" means that the base classifiers should have low correlation and as different as possible so that they can give different output predictions. The reason is related to the bias-variance-decomposition of ensembles, which will be covered in **Section 2.4.4**.

2.4.3 Combination Methods

For the second step, instead of choosing the best single learner, ensemble models combine the performance of all the base learners to achieve a strong

generalisation ability. It has been confirmed by many empirical studies that combination can reduce the variance as well as the bias of the base learning algorithms (Xu et al., 1992; Bauer and Kohavi, 1999). This point will be further discussed in **Section 2.4.4**.

The combination methods can be divided into two categories, i.e. averaging and voting. Averaging is the most popular combination method for numeric outputs, while voting is the most popular combination method for nominal outputs.

2.4.3.1 Averaging

Suppose we have an ensemble model composed of T individual learners $H =< h_1, h_2, ..., h_T >$. The output of example $\vec{x}$ given by learner h_i is $h_i(\vec{x})$. The result of simply averaging is

$$H(\vec{x}) = \frac{1}{T}\sum_{i=1}^{T} h_i(\vec{x}). \tag{20}$$

The weighted averaging method is an extension of the simple averaging method. It assumes that the importance of different base learners in the ensemble model is different. A weight parameter w_i is used for each base learner to indicate the importance of the learner. The output of an ensemble calculated by the weighted averaging method is

$$H(\vec{x}) = \frac{1}{T}\sum_{i=1}^{T} w_i h_i(\vec{x}), \tag{21}$$

where $w_i \geq 0$ and $\sum_{i=1}^{T} w_i = 1$. It is easy to see that the averaging method is a special case of the weighted averaging method with all the individual learners taking equal weights.

2.4.3.2 Voting

The voting combination methods are used for nominal classification, in which case the ensemble output is a predicted class label c among all the K possible classes $\{c_1, c_2, ...c_K\}$, or a predicted class probability estimate among all the class estimates. Given an example, the prediction of the base learner h_i is assumed to be a K dimensional vector $< h_i^1(\vec{x}), h_i^2(\vec{x}), ..., h_i^K(\vec{x}) >$, where

$h_i^k(\vec{x})$ is the output of h_i for class k. For class label prediction tasks, $h_i^k(\vec{x})$ takes value one if h_i predicts class k as the class label and zero otherwise. For class probability estimates prediction tasks, $h_i^k(\vec{x}) \in [0, 1]$ can be regarded as an estimate of the posterior probability $P(c_k|\vec{x})$.

Majority voting is the most popular used voting method. Every base learner votes for one class label, and the class label that receives more than half of the votes is selected as the final output of the ensemble. If none of the class labels has more than half of the votes, the ensemble takes no predictions. The prediction is

$$H(\vec{x}) = \begin{cases} c_j, & \text{if } \sum_{i=1}^{T} h_i^j(\vec{x}) > \frac{1}{2} \sum_{k=1}^{K} \sum_{i=1}^{T} h_i^k(\vec{x}). \\ rejection, & \text{otherwise.} \end{cases} \tag{22}$$

Plurality voting is an extension of the majoring voting method without the rejection mechanism. It takes the class received the largest number of votes as the output, which is defined as

$$H(\vec{x}) = c_{\text{argmax}_j \sum_{i=1}^{T} h_i^j(\vec{x})}. \tag{23}$$

Weighted voting deals with the case that the individual learners are not equally important. A bigger weight can be given to the learners that are more important. The output of the ensemble is defined as

$$H(\vec{x}) = c_{\text{argmax}_j \sum_{i=1}^{T} w_i h_i^j(\vec{x})}. \tag{24}$$

Soft voting is applicable to ensembles that produce class probability estimates. The predicted class probability estimate for class c_j by an ensemble model is

$$H^j(\vec{x}) = \sum_{i=1}^{T} w_i h_i^j(\vec{x}). \tag{25}$$

For base learners that are equally important, take $w_i = \frac{1}{T}$.

2.4.4 The Secret of Good Ensemble Models: Diversity

The bias-variance decomposition of a single model is discussed in **Section 2.3.1**. In this part, we discuss the bias-variance-covariance decomposition of ensemble models. This decomposition can give us insight into how

to build a good ensemble model.

For an ensemble of T learners $H =< h_1, ...h_T >$, the well known bias-variance decomposition can be further expanded, yielding the bias-variance-covariance decomposition of ensembles (Zhou, 2012). Suppose that the individual learners are combined with equal weights. The averaged bias and averaged variance are defined respectively as

$$\overline{bias}(H) = \frac{1}{T}\sum_{i=1}^{T}(E[h_i] - f), \qquad \overline{variance}(H) = \frac{1}{T}\sum_{i=1}^{T}E[(h_i - E[h_i])^2].$$

The averaged covariance for a pair of learners in the ensemble is defined as

$$\overline{covariance}(H) = \frac{1}{T(T-1)}\sum_{i=1}^{T}\sum_{j=1,j\neq i}^{T}E(h_i - E[h_i])E(h_j - E[h_j]).$$

The bias-variance-covariance decomposition of the squared error of the ensemble is

$$MSE(H) = \overline{bias}(H)^2 + \frac{1}{T}\overline{variance}(H) + (1 - \frac{1}{T})\overline{covariance}(H). \quad (26)$$

The first term in this equation requires the individual learners have a low bias on average and the final term requires they have low covariance.

The decomposition of this bias-variance-covariance formula shows that the secret of a well-constructed ensemble mode is that the base learners should have low correlation (or, high diversity), i.e. they should be as different as possible. A common technique for increasing diversity is to introduce randomness into the learning process, which can be reflected in data sample manipulation and input feature manipulation (Zhou and Liu, 2010).

Data sample manipulation is the most popular way to inject randomness to the learning process by generating multiple different training data samples based on sampling methods, and then the base learners can be learned from different data samples. Instead of constructing base learners based on different training subsets, input feature manipulation is another popular way to increase diversity by constructing the based learners trained from different feature subsets. The base learners are usually diverse because different feature subsets provide different views on the data.

2.4.5 Existing Ensemble Models and Their Diversity

The two most popular ensemble methods are Bagging and Boosting, both of which use the data sample manipulation method to train diverse base learners. Bagging (Breiman, 1996) constructs an ensemble by learning multiple base learners on different sub-training sets in a parallel way. These sub-training sets are sampled with replacement from the original training dataset, where the sizes of these subsets are the same as the whole training set. Boosting (Freund and Schapire, 1995) is another kind of ensemble technique that builds individual models in a sequential way. Each individual model learns from mistakes made by the previous one by altering the distribution of the training dataset and forcing learners to focus on misclassification errors. It finishes learning when no further improvements can be made.

Random Forest (Breiman, 2001) is an example of adopting both the data sample manipulation and input feature manipulation. It further increases the diversity of Bagging by adding the split feature random selection mechanism for learning each individual decision trees in the forest. Random Forest generates a random subset of the splitting features for every internal node and then selects the best attribute from it, whereas a standard decision tree selects the most important of all possible attributes. The size of the random subset is usually set to be $\sqrt{M}$, where M is the total number of attributes in the training data. Compared with Bagging, Random Forest has more diversity and is more competitive.

XGBoost (Chen and Guestrin, 2016), short for eXtreme Gradient Boosting, is an implementation of a generalised gradient boosting algorithm that has recently been dominating in machine learning competitions for structured and tabular data. This is due to its excellent predictive performance, highly optimised multi-core and distributed machine implementation and the ability to handle sparse data. Despite good performance relative to existing gradient boosting implementations, XGBoost can be very time consuming to run. Common tasks can take hours or even days to complete. Building highly accurate models using gradient boosting also requires extensive parameter tuning (Mitchell and Frank, 2017).

2.4.6 Summary

The two sources of diversity in Random Forest makes it a highly diverse ensemble model of decision trees. This inspires us to build a powerful ensemble model of Bayesian network classifiers in this research project. Details of the ensemble model are discussed in **Section 4.2**.

2.5 Scoring Rules for Evaluating Probability Estimation

For classification tasks in which predictions must assign probability estimates to a set of mutually exclusive classes, the probability estimates must sum to one, where each probability is in the range of 0 and 1. Scoring rules for measuring the prediction accuracy of the class probabilities are preferred compared with classification accuracy or error rate.

In decision theory, a scoring rule (Gneiting and Raftery, 2007), measures the accuracy of probabilistic predictions. RMSE and KL divergence are proper scoring rules, which means minimising them leads to well calibrated learners in the limit of large data. In this section, we will discuss the commonly used scores for estimating the accuracy of the class probability estimation of classifiers. These scores can be divided into two categories: least-squared error-based scores and entropy-based scores. Before introducing these methods, let's make some general definitions.

Given a dataset $\mathcal{D}$ containing N examples and K classes $\{c_1, c_2, ..., c_K\}$, the true class probability estimates for one of the example $\langle \vec{x}, y \rangle$ is $\vec{p} =< p_1, p_2, ..., p_K >$, where only p_k is equal to 1 if $y = c_k$ and everything else is 0. Let $\vec{q} =< \hat{q}_1, \hat{q}_2, ..., \hat{q}_K >$ denote the predicted class probability estimates produced by a probabilistic classifier for this test example.

2.5.1 Least-Squared Error Scores

One commonly employed measure of the accuracy of predicted class probabilities is called squared error. Squared error measures the distance between the true probability distribution $\vec{p}$ and the predicted probability distribution

$\vec{q}$. The squared error for this specific example is then defined as

$$SE = \sum_{k=1}^{K} (p_k - q_k)^2, \tag{27}$$

where p_k the actual class probability for class c_k, while q_k is the predicted probability for class c_k. Squared error is the summation over all the classes.

2.5.1.1 Brier Score (MSE)

Brier score (Brier, 1950), named for Glenn Brier, is a proper score function that measures the accuracy of the probability predictions. Brier score is the same with mean squared error (MSE) in discrete context. The performance of a model measured by the Brier score can be summarized as the average squared error across all classes predicted for a test dataset of N examples, which is defined as

$$BS = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} (p_{ik} - q_{ik})^2, \tag{28}$$

where p_{ik} is the true probability estimate for example i with class k, and q_{ik} is the estimate of predicting the example i as class k. The value of the Brier score is in the range of 0 and 1. The lower the Brier score is, the better the predictions are estimated.

2.5.1.2 RMSE

Root Mean Squared Error (RMSE) (Barnston, 1992) is simply the square root of the MSE (Brier score), which is defined as

$$\begin{aligned} RMSE &= \sqrt{BS} \\ &= \sqrt{\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} (p_{i,k} - \hat{p}_{i,k})^2}, \end{aligned} \tag{29}$$

Since RMSE is an extension of Brier score, it can also be used to measure the accuracy of the probabilistic predictions.

2.5.2 Entropy-Based Scores

In information theory, entropy (Shannon, 1948) measures the uncertainty of the distribution of a random variable. For a random variable $\mathcal{X}$ with distribution $\vec{p}$, the entropy is defined as

$$\mathbb{H}(\vec{p}) \stackrel{\Delta}{=} -\sum_{k=1}^{K} p_k \log_2 p_k, \tag{30}$$

where K is the number of states for the random variable $\mathcal{X}$ and p_k is the probability for state k. In a binary case where $K = 2$ and $\mathcal{X} \in \{0, 1\}$, the equation becomes

$$\begin{aligned} \mathbb{H}(\vec{p}) &\stackrel{\Delta}{=} -\left[p_0 \log p_0 + p_1 \log p_1\right] \\ &\stackrel{\Delta}{=} -\left[\theta \log_2 \theta + (1-\theta) \log_2 (1-\theta)\right], \end{aligned} \tag{31}$$

where $p_0 = \theta$ and $p_1 = 1 - \theta$. This is called binary entropy function and can be denoted by $\mathbb{H}(\theta)$.

When $\vec{p}$ is a uniform distribution, i.e., for any $k \in K, p_k = \frac{1}{K}$, the entropy is maximized to $log_2 K$. Take the binary entropy as an example, the maximum entropy 1 occurs when $\theta = 0.5$. In this case, the random variable $\mathcal{X}$ with distribution p has the greatest uncertainty. In contrast, when $\vec{p}$ is any delta-function that puts all of its mass to just one state, i.e., $p_k = 1$ for states k while for other states the probabilities are zeros, the entropy is minimized to 0. Such distribution has no uncertainty.

2.5.2.1 Cross-Entropy

Cross-entropy (De Boer et al., 2005) has been used as an alternative to squared error to measure the distance between two probability distributions given a random variable. The intuition of the cross-entropy score comes if we consider a true probability distribution $\vec{p}$ and a prediction distribution $\vec{q}$, then the cross-entropy of $\vec{q}$ from $\vec{p}$, denoted by $\mathbb{H}(\vec{p}, \vec{q})$, is the number of bits to represent an event using $\vec{q}$ instead of $\vec{p}$. The cross-entropy of $\vec{q}$ from $\vec{p}$ is defined as

$$\mathbb{H}(\vec{p}, \vec{q}) \stackrel{\Delta}{=} -\sum_{k=1}^{K} p_k \log_2 q_k. \tag{32}$$

Cross-entropy is a positive value measured in bits, and smaller values indicate better predictions. The cross-entropy heavily penalizes predicted probabilities that are far away from their expected value. When the two distributions are identical, the cross-entropy reduces to the entropy of $\vec{p}$, i.e., $\mathbb{H}(\vec{p}, \vec{q}) = \mathbb{H}(\vec{p}, \vec{p}) = \mathbb{H}(\vec{p})$. On the contrary, when the difference between the two distributions is large, the cross-entropy is large.

Cross-entropy is not log loss, but they calculate the same quantity when used as loss functions for classification problems and can be used interchangeably.

2.5.2.2 KL divergence

The other way to measure the dissimilarity between two probability distribution is the Kullback-Leibler divergence (KL divergence) or the relative entropy (Kullback and Leibler, 1951). The KL divergence of distribution $\vec{q}$ from $\vec{p}$ is defined as

$$\mathbb{KL}(\vec{p}||\vec{q}) \triangleq \sum_{k=1}^{K} p_k \log \frac{p_k}{q_k}, \tag{33}$$

where K is number of classes. It can be decomposed into

$$\begin{aligned} \mathbb{KL}(\vec{p}||\vec{q}) &\triangleq \sum_{k=1}^{K} p_k \log p_k - \sum_{k=1}^{K} p_k \log q_k \\ &\triangleq -\mathbb{H}(\vec{p}) + \mathbb{H}(\vec{p}, \vec{q}) \\ &\triangleq \mathbb{H}(\vec{p}, \vec{q}) - \mathbb{H}(\vec{p}, \vec{p}), \end{aligned} \tag{34}$$

where $\mathbb{H}(\vec{p}, \vec{q})$ is the cross-entropy of $\vec{q}$ from $\vec{p}$, and $\mathbb{H}(\vec{p}) = \mathbb{H}(\vec{p}, \vec{p})$ is the entropy of the distribution $\vec{p}$.

The above equation indicates that the KL divergence $\mathbb{KL}(\vec{p}||\vec{q})$ can be decomposed into the difference between the cross-entropy of $\vec{q}$ from $\vec{p}$ and the entropy of $\vec{p}$. When $\vec{p}$ is a delta-function that puts all of its mass to just one class, the entropy of $\vec{p}$ is zero, then the KL divergence is equal to the cross-entropy, i.e., $\mathbb{KL}(\vec{p}||\vec{q}) = \mathbb{H}(\vec{p}, \vec{q})$ when $\mathbb{H}(\vec{p}) = 0$. The KL divergence is a measure of the additional bits of information needed when $\vec{q}$ is used to approximate $\vec{p}$ instead of $\vec{p}$.

Although KL divergence measures the distance between two probability

distribution, but it is not a distance measure. KL divergence is non-symmetric, which means that the KL divergence of $\vec{q}$ from $\vec{p}$ is different from the KL divergence of $\vec{p}$ from $\vec{q}$ when $\vec{p} \neq \vec{q}$, i.e., $\mathbb{KL}(\vec{p}||\vec{q}) \neq \mathbb{KL}(\vec{q}||\vec{p})$.

2.5.3 Summary

In this section, we covered scores that are commonly used to evaluate probability estimation, including least-squared scores and entropy-based scores.

Least-squared scores, such as Brier score and RMSE, have a long history of being used in measuring the quality of class probability estimates. They work well when we do not have extreme probabilities. However, for cases with extreme probabilities, such as in Natural Language Processing (NLP) applications, relative probabilities are needed and entropy-based scores, such as cross-entropy and KL divergence, work better.

In this research, we focus on improving the class probability estimation of Bayesian network classifiers and decision trees. We use RMSE as the score to evaluate the class probability estimation because we do not have extreme probabilities.

2.6 Sign Test on Win-Draw-Loss Statistics

When comparing the performance of two different models, the Win-Draw-Loss is used. "Win" refers to the number of datasets with better performance of the newly proposed model compared with the existing model, while "Loss" refers to the number of data sets with the poor performance of the new model. "Draw" is the number of data sets they perform the same. If the "Win" is greater than the "Loss", then the new model is better than the existing one.

To test whether the new model is significantly better than the existing one, sign test technique is used. Sign test is a statistical method to test for consistent differences between pairs of observations (Wikipedia contributors, 2020). The paired observations may be designated x and y. For comparisons of paired observations (x, y), the sign test is most useful if comparisons can only be expressed as $x > y$ (win), $x = y$ (draw), or $x < y$ (loss). In Win-Draw-Loss statistics, the pair of observations is the RMSE or 0-1 loss of two different models on the same dataset. Given pairs of observations for each subject, the sign test determines if one member of the pair tends to be greater than (or less than) the other member of the pair.

There are two different ways to do the sign test: two-sided and one-sided. For two-sided sign test, the null hypothesis is that there is no difference between "Win" and "Loss", and the alternative hypothesis is "Win" may be either greater than or less than "Loss". A one-sided sign test could be "Win" is greater than "Loss" so that the difference can only be in one direction (greater than).

In this thesis, we adopt the one-sided sign test to test whether a newly proposed model is significantly better than an existing model. The null hypothesis is that the new model is not better than the other model. The alternative hypothesis is that the new model is better than the other one. A difference is considered to be significant if the p-value is less than or equal to an α value.

The "Win" and "Loss" can be regarded as the two outcomes of a binomial trial, so we can calculate the p-value using the probability mass function of the binomial distribution, which is defined as

$$p - value = \binom{N}{k} p^{Win}(1-p)^{Loss}, \tag{35}$$

where $N = Win + Loss$ is the total number of trials. $p = 0.5$ because if the none hypothesis holds, there should be no difference between "Win" and "Loss", the probability of winning and losing are equally to be 0.5. After calculating the p-value, we compare it with the α value, which is set to be 0.05 in this thesis. If the p-value is smaller than or equal to α, then reject the null hypothesis and say the new model is better. Otherwise, accept the null hypothesis and say the new model is not better than the other model.

2.7 Discretization of Continuous Features

Many Machine Learning algorithms can only be applied to data described by discrete features, such as most of the Bayesian network classifiers. For these algorithms, discretization algorithm should be used to transform continuous features into discrete ones before learning the model.

Discretization is the process of partitioning continuous features into many intervals by finding a set of cut points (Kotsiantis and Kanellopoulos, 2006). "Cut point" refers to a real value within the range of the continuous feature. It divides the range into two intervals, one less than or equal to the cut point

and the other greater than the cut point.

A typical discretization process generally consists of four steps: (a) sorting the values of the continuous feature in ascending or descending order, (b) evaluating a cut-point for splitting or adjacent intervals for merging, (c) according to some criterion, splitting or merging intervals of continuous value, and (d) finally stopping at some point (Kotsiantis and Kanellopoulos, 2006; Dash et al., 2011).

There are many discretization methods available. These discretization methods can be divided into two categories: unsupervised and supervised. The difference between the two is whether the class membership information is used during the process of discretization. Unsupervised methods do not use class information for discretization. In contrast, supervised methods bring class information into the discretization process. Previous research indicated that supervised are better than unsupervised methods (Dougherty et al., 1995; Kotsiantis and Kanellopoulos, 2006). In the following sections, we will describe their common methods.

2.7.1 Unsupervised Discretization

Unsupervised discretization methods do not make use of class membership information during the discretization process. Some representative algorithms of unsupervised methods are the equal-width method, equal-frequency method and K-means clustering method.

2.7.1.1 Equal-Width Discretization

The equal-width discretization algorithm determines the minimum and maximum values of the discretized attribute and then divides the range into a user-defined number N of equal-width intervals. The width is calculated by

$$width = \frac{maximum\ value - minimum\ value}{N}. \tag{36}$$

The obvious weakness of the equal-width method is that in cases where the outcome observations are not distributed evenly, a large amount of important information can be lost after the discretization process.

2.7.1.2 Equal-Frequency Discretization

The equal-frequency algorithm determines the minimum and maximum values of the discretized attribute, sorts all values in ascending order, and divides the range into a user-defined number of intervals so that every interval contains the same number of sorted values.

The weakness of equal-frequency is that many occurrences of a continuous value could cause the occurrences to be assigned into different bins. One improvement can be after continuous values are assigned into bins, boundaries of every pair of neighbouring bins are adjusted so that duplicate values should belong to one bin only Kotsiantis and Kanellopoulos (2006).

2.7.1.3 K-Means Discretization

K-means (MacQueen et al., 1967), as one of the most popular clustering method, is also suitable for discretization. K-means aims to partition n observations into K clusters in which each observation belongs to the cluster with the nearest mean (cluster centers). When used for discretization, the intervals are the clusters identified by the k-means algorithm. The number of clusters (K) is defined by the user. In fact, discretization by k-means is equivalent to a 1-dimensional k-means, where only the continuous feature we want to discretize is used.

Initially, the algorithm assigns K data points to be the so-called centers (or centroids) of the clusters randomly. Then each data point of the given set is associated with the closest center resulting in the initial distribution of the clusters. After this initial step, the next two steps are performed until the convergence is obtained:

- Recompute the centers of the clusters as the average of all values in each cluster.
- Each data point is assigned to the closest center. The clusters are formed again.

The algorithm stops when there is no data point that needs to be reassigned, or the number of data point's reassignments is less than a given small number (Dash et al., 2011).

2.7.2 Supervised Discretization

Supervised discretization methods make use of the class information when doing discretization for continuous features. In the following sections, two examples of supervised discretization methods, the Minimum Description Length Principle and the decision tree discretization methods, are introduced.

2.7.2.1 Minimum Description Length Principle

The Minimum Description Length Principle (MDLP) (Fayyad and Irani, 1992) is a top-down discretization method. It considers one big interval containing all known values of a continuous feature and then partitions this interval into smaller and smaller sub-intervals until the Minimum Description Length stopping criterion is achieved.

The MDLP method first sorts all the values of a continuous feature in ascending order and calculates the midpoint for each pair of values. These midpoints are the candidate cut points that need to be evaluated. For each candidate, it splits the data into two intervals, and the resulting information gain and the minimum description length of the cut are calculated. The cut point with the maximum information gain of all candidate cutting points is selected as the best cut point. If the information gain of the best cut point is greater than the minimal description length of the cut, the partition induced by the cut point is accepted. Otherwise, it is rejected. This binary discretization is applied recursively until the gain is lower than the minimal description length of the best cut.

A partition induced by a cut point T of feature A for a data set S of N examples is accepted if and only if

$$Gain(A,T,S) > \frac{\log_2(N-1)}{N} + \frac{\Delta(A,T,S)}{N}. \tag{37}$$

$\Delta(A,T,S)$ is calculated by

$$\Delta(A,T,S) = \log_2(3^k-2) - [kEntropy(S) - k_1 Entropy(S_1) - k_2 Entropy(S_2], \tag{38}$$

where k is the number of classes in S, k_1 and k_2 are the number of classes the two partitions S_1 and S_2, respectively. k_1 and k_2 are subsets of k.

To summarize, the key idea of MDLP is to divide the range of the continuous features into intervals that maximize the information, measures

by entropy. However, the number of intervals should not be too many to avoid overfitting. That is why the MDL stopping criterion is adopted.

2.7.2.2 Decision Tree Discretization

Decision tree can also be used as a discretization method (Kohavi and Sahami, 1996). It consists of two steps:

- Step 1: Train a complete binary tree with limited depth (2, 3 or 4) using the continuous feature A we want to discretize to predict the class. The method selects the value of A that has the minimum entropy as a split-point, and recursively partitions the resulting intervals to arrive at a hierarchical discretization. The values fall into the same terminal node are grouped as an interval. For a tree with depth n, the tree gets 2^n terminal nodes.

- Step 2: Apply train pruning to find an appropriate number of terminal nodes (i.e. the number of discretization intervals) in a bottom-up way.

Decision trees find the number of intervals and the cut points automatically. However, it may cause overfitting and the parameters (i.e., maximum depth and the minimum number of samples in one terminal node) tuning is time-consuming.

2.7.3 Summary

In this section, we introduced commonly used discretization methods for continuous features, including unsupervised methods and supervised ones. Previous studies show that supervised discretization methods are better than unsupervised ones.

2.8 Leave-One-Out Cross-Validation

In the SKDB algorithm mentioned in this chapter, the Leave-One-Out Cross-Validation (LOOCV) method is used to tune the hyperparameter to get the optimum network structure. In the HGS algorithm we proposed in **Chapter 5**, we proposed a cost function for decision trees based on the idea of LOOCV. Therefore, it is necessary to give a brief introduction to the LOOCV approach in this section.

K-Fold Cross-Validation is mainly used for hyperparameter tuning of predictive models conducted on a validation set. It first divides the whole data set into K equal partitions randomly. Then for each partition, use it as the testing data to test a model, and combined other partitions together as the training data to learn a model. This procedure is repeated K times, and the final result is the average over the K performances. The variance of the resulting estimate is reduced as K is increased.

LOOCV is a special case of the K-fold cross-validation where K is maximised by the number of instances N in the data. The variance of the estimates is the lowest compared to other K values. In each fold, nearly all the data except for a single observation are used for training, and the model is tested on that single observation.

The evaluation given by LOOCV is accurate, but it seems expensive to compute because it rebuilds the model from scratch for each fold. However, LOOCV could be sped up by using incremental LOOCV (Kohavi, 1995; Joulani et al., 2015) method for any algorithms that support for incremental learning, such as decision trees. The idea is instead of training a model during each fold of the cross-validation, first, train a model on the full dataset, then delete the one example that is left out, test on that example, then insert it into the model again. This delete-test-insert phase is repeated for each of the N folds. Incremental LOOCV can be conducted on any algorithm that supports incremental learning, allowing for dynamically adding or removing examples from the model. Note incremental LOOCV means the model structure remains unchanged.

2.9 Summary

In this chapter, we reviewed a number of techniques related to the research topic, including the bias-variance analysis of classifiers, variance reduction methods, scoring rules for evaluating probability estimates, a significant testing technique of Win-Draw-Loss statistics, discretization of continuous features, and the leave-one-out cross-validation method.

Chapter 3

Literature Review on Bayesian Network Classifiers and Decision Trees

3.1 Bayesian Network Classifiers

Bayesian network classifier, as one of the primary classifiers in this study, will be introduced in detail in this section, including the basic framework, existing classification models and the current work to improve parameter estimation. A brief summary will be given at the end of this section.

3.1.1 Background Knowledge

Let capital letters $\mathbf{X} = (X_1, X_2, ...X_n)$ represent n attributes in a dataset. Lower case letters $\mathbf{x} = (x_1, x_2, ..., x_n)$ represents specific data values taken by these attributes. Specifically, Y is the class variable, and y is the possible class labels that data $\mathbf{x}$ belongs. The basic task of a classification problem is to compute the conditional class probability distribution $P(y|\mathbf{x})$ over all the possible classes and assign the class to $\mathbf{x}$ most suited given the context, for instance depending on the tradeoff between recall and precision desired.

A Bayesian Network (BN) $\mathcal{B} = \langle \mathcal{G}, \Theta \rangle$ is characterized by two parameters $\mathcal{G}$ and Θ. $\mathcal{G}$ is a directed acyclic graph whose nodes and edges represent random variables $X_1, X_2, ...X_n$ and direct dependencies between those variables, respectively. We say X_i is the parent of X_j if X_i is pointing directly to X_j via a single edge. Another parameter Θ quantifies the network structure

with a set of parameters $\theta_{x_i|\Pi_i(x)} = P_{\mathcal{B}}(x_i|\Pi_{x_i})$ for each possible value x_i of X_i, and Π_{x_i} of Π_{X_i}, where Π_{X_i} denotes the set of parents of X_i in $\mathcal{G}$. A BN defines the joint probability distribution over $\mathbf{x}$ given by

$$P_{\mathcal{B}}(\mathbf{x}) = \prod_{i=1}^{n} P_{\mathcal{B}}(x_i|\Pi_{x_i}) = \prod_{i=0}^{n} \theta_{x_i|\Pi_{x_i}}. \tag{39}$$

When using a Bayesian Network as a classifier, the precision of posterior estimates $P_{\mathcal{B}}(y|\mathbf{x})$ matters rather than the precision of $P_{\mathcal{B}}(y, \mathbf{x})$. As a result, it is usually important to ensure that all variables in the class' Markov blanket are connected directly to the class, which means that Y should be the common parent for all the random variables in the network structure of a BNC. The conditional class probability of a BNC can then be written as

$$P_{\mathcal{B}}(y|\mathbf{x}) = \frac{P_{\mathcal{B}}(y, \mathbf{x})}{P_{\mathcal{B}}(\mathbf{x})} = \frac{\theta_y \prod_{i=1}^{n} \theta_{x_i|y,\Pi_{x_i}}}{\sum_{y' \in Y} \theta_{y'} \prod_{i=1}^{n} \theta_{x_i|y',\Pi_{x_i}}} \propto \theta_y \prod_{i=0}^{n} \theta_{x_i|y,\Pi_{x_i}}. \tag{40}$$

3.1.2 The Bias-Variance of Bayesian Network Classifiers

Bayesian network classifiers are good models for analysing large datasets because they are out-of-core learners, i.e., without the need to save the data in the main memory. The bias-variance tradeoff of Bayesian network classifiers can easily be tuned by putting a limit on the maximum number of parents that a node can have. With k parents, the model then looks at all possible combinations of the $(k + 1)$ nodes connected with each other. The higher the value of k, the lower the bias of the algorithm (and usually the higher the variance as well). For large datasets, a higher complexity or low-biased model is preferable because it allows the model to capture fine detail in data more precisely. But this low bias model has more parameters and potentially higher variance, leading to poorer predictions. As a result, more data is usually needed (or a superior parameter estimation technique).

3.1.3 Existing Bayesian Network Classifiers

Some existing BNCs have been developed and gained popularity, including single BNCs and ensemble BNCs. Single BNCs only consists of one classifier, including NB, TAN, KDB and the most recently SKDB. Ensemble BNCs

improve the probability estimates by building multiple single BNCs and averaging the results of them. AODE and KDF are introduced as examples of ensemble BNCs.

In this section, we take the Iris dataset from the UCI repository (Lichman, 2013) as an example to train a Naive Bayes, TAN, KDB and SKDB classifier respectively. This dataset predicts the iris class by the length and width of sepal and petal. The class Y has three values, which are Iris Setosa, Iris Versicolour and Iris Virginica.

3.1.3.1 Naive Bayes

NB (Lewis, 1998) is the simplest BNC with a strong unrealistic independence assumption that each attribute is conditionally independent of every other attribute given the class label. This makes the class the parent of all other attributes and includes no other edges. Although this assumption is unrealistic in many practical applications, NB is computationally efficient, especially when the number of attributes is small. **Figure 5** shows the NB structure for the Iris dataset, where the Iris class is the common and unique parent of the predictive features, and they are independent of each other.

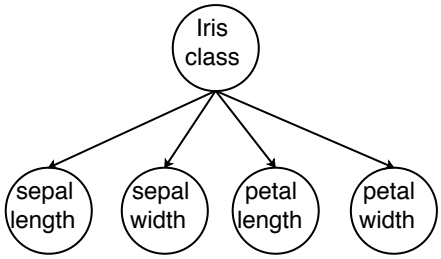

Figure 5: The NB structure for the Iris dataset.

3.1.3.2 Tree-Augmented Bayesian Network Classifier

TAN (Friedman et al., 1997) relaxes the strong independence assumption of NB by adding an extra single parent to each non-class feature. For the Iris dataset, the sepal length attribute is chosen to be the parent for other attributes, as shown in **Figure 6**.

The key problem of learning a TAN classifier is how to learn the single parent for each non-class feature. Chow and Liu (1968) equates this problem

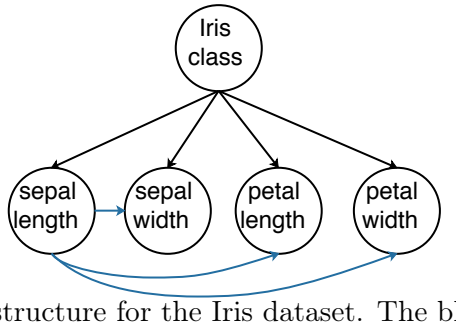

Figure 6: The TAN structure for the Iris dataset. The blue edges are the extra parents added by TAN for the NB structure.

to the problem of how to build the maximum weighted spanning tree from a complete undirected graph. The problem of finding such a tree is to select a subset of edges from the complete undirected graph such that the selected edges constitute a tree and the sum of weights attached to the selected edges is maximised. Chou and Liu use the Mutual Information (MI) $MI(X_i, X_j)$ as the weight of the edge between them, which is defined as follows

$$MI(X_i; X_j) = \sum_{x_i, x_j} P(x_i, x_j) \log \frac{p(x_i, x_j)}{p(x_i) \cdot p(x_j)}. \tag{41}$$

Roughly speaking, this function measures how much information X_j provides about X_i.

Unlike Chou and Liu's method, the TAN algorithm uses the Conditional Mutual Information (CMI) $CMI(X_i; X_j|Y)$ as the weight value between X_i and X_j to generate the maximum weight spanning tree, which is defined as follows

$$CMI(X_i; X_j|Y) = \sum_{x_i, x_j, y} p\left(x_i, x_j|y\right) \log \frac{p(x_i, x_j|y)}{p(x_i|y) \cdot p(x_j|y)}. \tag{42}$$

Roughly speaking, $CMI(X_i; X_j|Y)$ measures the information that X_j provides about X_i when the value of class Y is known. **Algorithm 1** shows how to learn a TAN classifier given training data $\mathcal{T}$.

3.1.3.3 K-dependence Bayesian Network Classifier

KDB (Sahami, 1996) further relaxes the independence assumption of NB and TAN by allowing each non-class attribute to have up to K parents,

Algorithm 1: learnTAN($\mathcal{T}$)

Input : A training set $\mathcal{T}$ with $|\mathbf{X}|$ features and $|Y|$ classes
Output : A TAN model $\mathcal{B}$

`// structure learning`
Compute class conditional mutual information $CMI(X_i; X_j|Y)$ between each pair of features X_i and X_j given Y, where $i \neq j$.
Let $\mathcal{G}$ be a complete undirected graph in which the vertices are the features and the weight of an edge between X_i and X_j is $CMI(X_i; X_j|Y)$.
Build a maximum weighted spanning tree.
Transform the resulting undirected tree to a directed one by choosing a root variable and setting the direction of all edges to be outward from the it.
Construct a TAN model by adding a vertex labeled by Y and adding an arc from Y to each X_i.
`// parameter learning`
Compute the conditional probability tables Θ inferred by the structure of $\mathcal{G}$ by using counts from $\mathcal{T}$.
Let $\mathcal{B} = \langle \mathcal{G}, \Theta \rangle$.
return $\mathcal{B}$.

where K is a user-defined value. The network structure of a KDB classifier is determined by the **attribute order** and **parent order**. Attribute order is the importance of the attributes ranked from largest to smallest by their MI value. An attribute that ranks higher in the order provides more information for classification of the class and has priority to be chosen as the parent of the other attributes. The parent order for each attribute is the parent nodes in order of CMI values from largest to smallest.

Figure 7 is the structure of KDB model with $K = 2$ for the Iris dataset with each attribute has up to two parents. As it can be seen from **Figure 7**, the attribute order is petal width, petal length, sepal length and sepal width. This ordering demonstrates that petal information is more important for predicting iris categories than sepal. The parent order for each attribute is listed in **Table 1**.

Algorithm 2 is the training algorithm of KDB. It takes a training dataset T and a user-supplied K as inputs, and returns a KDB model $\mathcal{B}$ as output. The learning method consists of two parts: structural learning and parametric learning. **Algorithm 3** learns the KDB structure by first sorting the attributes on MI with the class and the CMI for each pair of attributes

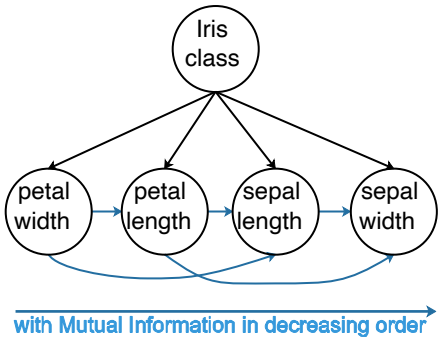

Figure 7: The KDB-2 structure for the Iris dataset.

Table 1: Attribute order and parent orders for the Iris datasets.

Attribute order (↓)	Parent orders (→)
petal width	Iris class
petal length	Iris class, petal width
sepal length	Iris class, petal width, petal length
sepal width	Iris class, petal length, sepal length

given the class. Each attribute X_i is assigned the K parent attributes that maximise CMI out of those attributes with higher MI. **Algorithm 4** learns the parameters by compute the Conditional Probability Table (CPT) from T. The last step returns a KDB model $\mathcal{B}$.

Algorithm 2: learnKDB($\mathcal{T}, K$)

Input : A training set $\mathcal{T}$ with features $\mathbf{X}$ and classes Y
Input : Number of parents allowed for each feature K
Output : A KDB model $\mathcal{B}$

Let $\mathcal{G}$ be a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which $\mathcal{V}$ is a set of vertices and $\mathcal{E}$ is a set of edges.
$\mathcal{G} = learnStructure(\mathcal{T})$. `// Algorithm 3`
$\Theta = learnParameters(\mathcal{T}, \mathcal{G})$. `// Algorithm 4`
Let $\mathcal{B} = \langle \mathcal{G}, \Theta \rangle$.
return $\mathcal{B}$.

3.1.3.4 Selective KDB

SKDB (Martınez et al., 2016) is an extension of KDB with a smaller network structure but better performance. SKDB considers a complete KDB model

Algorithm 3: learnStructure($\mathcal{T}$)

Input : A training set $\mathcal{T}$
Output : A KDB structure $\mathcal{G}$

Calculate $MI(X_i; Y)$ from $\mathcal{T}$ for all attributes $X_i \in \mathbf{X}$.
Calculate $MI(X_i; X_j|Y)$ from $\mathcal{T}$ for each pair of attributes $(i \neq j)$.
Let $\mathcal{L}$ be a list of all X_i in decreasing order of $MI(X_i; Y)$.
$\mathcal{V} = \{Y\}$; $\mathcal{E} = \emptyset$.
for $i = 1 \rightarrow \mathcal{L}.size$ **do**
    $\mathcal{V} = \mathcal{V} \cup \mathcal{L}_i$.
    $\mathcal{E} = \mathcal{E} \cup (Y, \mathcal{L}_i)$.
    $vk = min(i - 1, k)$.
    while $vk > 0$ **do**
        $m = argmax_j\{MI(\mathcal{L}_i; \mathcal{L}_j|Y)|1 \leq j < i \wedge (\mathcal{L}_i, \mathcal{L}_j) \notin \mathcal{E}\}$.
        $\mathcal{E} = \mathcal{E} \cup (\mathcal{L}_j, \mathcal{L}_i)$.
        $vk = vk - 1$.
    end
end
return $\mathcal{G}$.

Algorithm 4: learnParameters($\mathcal{T}, \mathcal{G}$)

Input : Training set $\mathcal{T}$ and $\mathcal{G}$
Output : Θ

Initialize Θ to structure $\mathcal{G}$.
Compute the CPTs for $Theta$ from $\mathcal{T}$.
return Θ.

is composed of $K \times n$ sub-models learned from a dataset with n predictive attributes. It compares the performance of all the sub-models and selects the best one as the final selected KDB (SKDB) model. **Table 2** shows all the sub-models for a full KDB-4 model build from 4 attributes, where $M_{i,j}$ means KDB with $K = i$ using attributes from 1 to j.

Table 2: A full KDB-4 model is composed of many sub-models.

K	**Attributes (ordered by MI)**			
	X_1	X_2	X_3	X_4
1	$M_{1,1}$	$M_{1,2}$	$M_{1,3}$	$M_{1,4}$
2	$M_{2,1}$	$M_{2,2}$	$M_{2,3}$	$M_{2,4}$
3	$M_{3,1}$	$M_{3,2}$	$M_{3,3}$	$M_{3,4}$
4	$M_{4,1}$	$M_{4,2}$	$M_{4,3}$	$M_{4,4}$

Figure 8 is the SKDB network structure of the Iris datasets. Compared with the KDB structure, SKDB only selects the petal width and the petal length as the two predictive attributes and the non-class parents for these two attributes are all discarded.

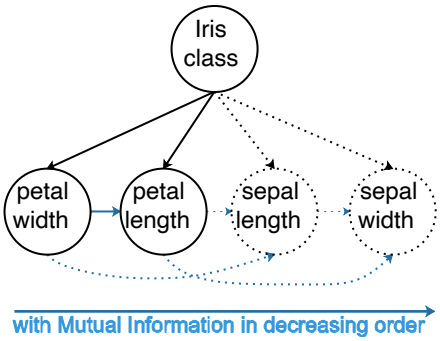

Figure 8: The SKDB-2 structure for the Iris dataset.

Algorithm 5 shows the detail of learning an SKDB model. It firstly learns the structure and the parameters of a complete KDB model. Secondly, it requires one additional pass through the training data to select the best sub-model using the Leave-One-Out Cross-Validation (LOOCV) (Step 7 - 14).

3.1.3.5 AODE

AODE (Webb et al., 2005) is an augmentation of NB that relaxes the strong independence assumption of NB by averaging over several One-Dependence Estimators (ODEs). In each of these estimators, a different attribute is set to be the parent of all other attributes. Then, at prediction time, class probability estimates from the different ODEs are averaged. AODE is an efficient ensemble Bayesian Network classifier with the same simplicity of NB.

3.1.3.6 KDB Forest

KDF (Duan and Wang, 2017) is an ensemble model combining multiple KDBs by changing the predictive attribute orders. The ensemble size of KDF is equal to the number of attributes in the dataset. Each of these base KDB estimators uses a different first attribute. The other attributes are ordered according to the conditional mutual information with the previous attributes

Algorithm 5: learnSKDB($\mathcal{T}, K$)

Input : A training set $\mathcal{T}$ with $|\mathbf{X}|$ features and $|Y|$ classes
Input : A maximum number of parents value K
Output : A SKDB model $\mathcal{B}$

$\mathcal{G} = learnStructure(\mathcal{T})$
$\Theta = learnParameters(\mathcal{T}, \mathcal{G})$
Let $\mathcal{L}$ be a list of all X_i in decreasing order of $MI(X_i; Y)$.
Let $\mathcal{P}$ be a $k \times a$ matrix of posterior probabilities.
Let $\mathcal{LF}$ be a matrix of LOOCV results (of length $k \times a$) initialized with zeros.
Let $\Theta^{\downarrow \mathbf{x}}$ be the $\mathcal{B}$ Θ with example $\mathbf{x}$ discounted from its CPTs.
for $k^{'} = 1 \rightarrow K$ **do**
    $\mathcal{P}[k^{'}][y^*] = \hat{p}_{\Theta^{\downarrow \mathbf{x}}}(y^* | \mathbf{x}), \forall y^* \in Y$.
    for $l = 1 \rightarrow \mathcal{L}.size$ **do**
        $X_{max} = \mathcal{L}$.nextElement.
        $\mathcal{P}[k^{'}][y^*] = \mathcal{P}[k^{'}][y^*] \cdot \hat{p}_{\Theta^{\downarrow \mathbf{x}}}\left(x_{max} | pa^{k^{'}}_{x_{max}}, y^*\right)$, $\forall y^* \in \Omega_Y$, where $pa^{k^{'}}_{x_{max}}$ are the $k^{'}$ first parent-values of X_{max} in $\mathbf{x}$.
        $\mathcal{LF}[k^{'}][l] = \mathcal{LF}[k^{'}][l] + LossFunction(\mathcal{P}[k^{'}], y^{\mathbf{x}})$, where $\mathcal{P}[k^{'}]$ is the vector of posterior probabilities considering the top l attributes by MI.
    end
end
Select b and k indexes with best $\mathcal{LF}$.
Truncate Θ to attribute subset $\{1...b\}$ and maximum number of parents k.
Let $\mathcal{B} = \langle \mathcal{G}, \Theta \rangle$.
return $\mathcal{B}$.

already been chosen. KDF considers not only the mutual information between attribute and the class, but also the conditional mutual information between prior selected attributes, which they demonstrate makes an improvement to performance. The classification accuracy of KDF outperforms single BNCs and AODE.

Algorithm 6 shows the detail of how KDF learns the multiple attribute order. **Algorithm 7** shows the detail of how KDF learns the parent orders for a given attribute order.

Algorithm 6: KDF: attribute order learning

Input : A training set $\mathcal{T}$ with attributes $\{X_1, ..., X_n\}$
Output : Sequences $\{S_1, ..., S_n\}$

for *sequence* $S_i, i \in \{1, ..., n\}$ **do**
  Let S_i be empty.
  Let predictive attribute X_i be the root node.
  Add the root node to S_i.
  while $S_i.size < n$ **do**
    Compute Sum_CMI_j for the predictive attribute $X_j (j \neq i)$, which is not in S_i.
    Select X_{max}, which has the maximum value of Sum_CMI_j.
    Add X_{max} to S_i.
  end
end

Algorithm 7: KDF: parent order learning

Input : A training set $\mathcal{T}$ with attributes $X_1, ..., X_n$
Input : Sequences $\{S_1, ..., S_n\}$
Output : A KDF classifier KDF

Compute $CMI(X_i; X_j|Y)$, for each pair of attributes X_i and X_j, where $i \neq j$.
for *sequence* $S_i, i \in \{1, ..., n\}$ **do**
  Let KDF_i being constructed begin with a single class node, Y.
  while $KDF_i.size < n$ **do**
    Select the attribute X_{first}, which is the first attribute in S_i and not in KDF_i.
    Add a node to KDF_i representing X_{first}.
    Add an arc from Y to X_{first} in KDF_i.
    Select X_j, which is in KDF_i and has the largest value of $CMI(X_{first}; X_j|Y)$, as the first parent of X_{first}.
    Select other $b-1$ parents from ancestor attributes of X_j by comparing the value of $CMI(X_{first}; X_p|Y)$, where X_p is one of the ancestor attributes of X_j, $b = min(d, K)$ and d is the number of ancestor attributes of X_j.
  end
  Compute the conditional probability tables inferred by the structure of KDF_i by using counts from $\mathcal{T}$.
  $KDF.add(KDF_i)$.
end
return KDF.

3.1.4 Summary

In this section, we reviewed the existing Bayesian network classifiers, including single classifiers NB, TAN, KDB and SKDB and ensemble classifiers AODE and KDF. AODE and KDF are ensemble models build on TAN and KDB, respectively. In **Section 4.2**, we show the details of how to build an ensemble model of SKDB.

3.2 Decision Trees

In this section, the C4.5 algorithm is introduced first as a typical example of decision tree learning algorithms. Then the bias and variance of the decision tree are analyzed. The poor class probability estimates of decision trees and the corresponding methods in recent years are then analysed. A summary is provided at the end.

3.2.1 Background Knowledge: C4.5 Algorithm

C4.5 is a decision tree learning algorithm proposed by Quinlan (1993). This algorithm is viral and ranks first among the top ten algorithms of data mining (Wu et al., 2008). In 2011, authors of the Weka machine learning software described the C4.5 algorithm as a "landmark decision tree program that is probably the machine learning workhorse most widely used in practice to date" (Garner et al., 1995).

Algorithm 8 shows the pseudo-code of building a decision tree classifier. Given a training dataset $\mathcal{T}$ with attributes $\mathbf{X}$ and class Y, it first checks if all the examples in $\mathcal{T}$ belong to the same class or $\mathcal{T}$ is small. If that is the case, return the tree with a single leaf node labelled with the most frequent class in $\mathcal{T}$. Otherwise, select the attribute with the maximum information gain X_{best} as the splitting attribute of the root and create one child node for each outcome value of the attribute. Partition $\mathcal{T}$ into corresponding subsets $\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_v$ according to the attribute outcome for each example. Then for each child node, apply the same procedure recursively until all the nodes stop growing.

Decision tree uses entropy (Shannon, 1948) to measure the homogeneity of a sample. If the sample is completely homogeneous, the entropy is zero. If the sample is equally divided into different classes, then it has entropy of $\log_b$

(entropy equals to one when $b = 2$). The entropy of a sample $\mathcal{D}$ is defined as

$$Entropy(\mathcal{D}) = -\sum_{k=1}^{K} p_k \log p_k, \tag{43}$$

where $p_k = \frac{|\mathcal{D}_k|}{|\mathcal{D}|}$ is the probability of class k in the sample.

The information gain (Quinlan, 1986) is the decrease in entropy after a sample is split on an attribute. Splitting on the attribute with the highest information gain produce the most homogeneous branches. The information gain for the sample $\mathcal{D}$ and attribute A is defined as

$$IG(\mathcal{D}, A) = Entropy(\mathcal{D}) - \sum_{i=1}^{|A|} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} Entropy(\mathcal{D}_i), \tag{44}$$

where $|A|$ is the number of values of attribute A. $Entropy(\mathcal{D})$ is the entropy before splitting. $Entropy(\mathcal{D}_i)$ is the entropy for the ith value of attribute A after splitting. $\frac{|\mathcal{D}_i|}{|\mathcal{D}|}$ is the weight for the ith partition by splitting on A.

Algorithm 8: C4.5($root$,$\mathcal{T}$)

Input : A root node associated with dataset $\mathcal{T}$ with attributes $\{X_1, ..., X_n\}$
Output : A C4.5 $Tree$

if $\mathcal{T}$ *is pure or other stopping criteria met* **then**
  return a tree with *root* labelled with the most frequent class.
end
for $X_i \in \{X_1, ..., X_n\}$ **do**
  Compute information gain of splitting on X_i.
end
Let X_{best} be the attribute with the largest information gain.
Create a root node split on X_{best}.
Create children nodes $Children$ and split data $\mathcal{T}$ to the children nodes.
for $child_v \in Children$ **do**
  $Tree_v = C4.5(child_v, \mathcal{T}_v,)$
  Attach $Tree_v$ to the corresponding branch of Tree.
end
return $Tree$

3.2.2 The Bias-Variance of Decision Trees

A decision tree cannot produce accurate class probability estimates because it aims to build small trees with accurate class label predictions rather than accurate class probability estimates. Tree pruning techniques are used to make decision trees more accurate by removing the nodes and branches at the bottom of the tree that fitted to noise data. However, they are focused on accuracy maximisation that is therefore not suited to estimate class probabilities, especially for datasets with an unbalanced class distribution because they tend to prune the decision tree down to a single node if all classes except one are rare (Zadrozny and Elkan, 2001b).

The class probability estimates on the leaf nodes are usually calculated by the observed relative frequencies. However, this method may lead to unreliable estimates with high bias and high variance (Zadrozny and Elkan, 2001a). First, decision tree growing methods try to make leaves homogeneous, so the observed relative frequencies are systematically shifted towards zero and one. This leads to high bias estimates. Second, when the number of training examples associated with a leaf is small, observed relative frequencies are not statistically reliable, which leads to high variance estimates. To conclude, the probability estimation of a decision tree is not reliable and needs to be improved by proper machine learning techniques.

3.2.3 Existing Works on Improving Tree Estimates

The probability estimation of decision trees can be improved in many ways, including smoothing, curtailment, ensemble and calibration. In this section, we introduce these techniques one by one.

3.2.3.1 Tree Smoothing

Research suggests that when using decision trees to estimate class probabilities, it is preferable to do no pruning, but use smoothed probabilities instead (Zadrozny and Elkan, 2001b). Provost and Domingos (2003) believe that a thorough study of what are the best smoothing methods (and why) for PETs would be a useful contribution to machine learning research. Probability smoothing techniques, including Laplace correction and M-estimation, has been used for PETs in many research works and get improved probabilities than MLE.

3.2.3.2 Tree Curtailment

Tree curtailment method is based on the idea that if the parent of a small leaf, i.e. a leaf with few training examples, contains enough examples to induce a statistically reliable probability estimate, then assigning this estimate to a test example associated with the small leaf may be more accurate than smoothed estimates. When classifying an example, curtailment stops searching the decision tree as soon as it reaches a node that has less than v examples, where v is a parameter of the method. Their experiments show that v between 100 and 400 give similar results. Zadrozny and Elkan (2001b) recommend to combine curtailment with smoothing to produce relatively small and hence understandable decision trees, while still giving high-resolution, well-calibrated probability estimates.

3.2.3.3 Tree Ensemble

Zadrozny and Elkan (2001a) suggest that Bagging does not give probability estimates that are unbiased and well-calibrated, whether or not the base learning method is stable. However, Chawla and Cieslak (2006) and Provost and Domingos (2000) show that Bagging substantially improves probability estimates even for large and unbalanced datasets. They recommend Bagging or other ensemble generation methods with decision trees for improving the calibration of the probability estimates of decision trees. Bagging can reduce the variance and bias in estimation effectively.

Smoothing and ensembles both can improve the probability estimates of decision trees, but smoothing does not help much for ensemble models. Chawla (2006) show that the prior smoothing at the leaves using Laplace smoothing does not offer much gain with ensembles. Besides, Bostrom (2007) has investigated Laplace correction, M-estimation into the random forest, but each of them has its limitations. Then in 2008, he applied calibration methods to RF, Platt scaling and Isotonic regression, which has been proved successful in other applications but less beneficial for RF (Boström, 2008). Further, Boström (2012) focuses on improving the probability estimates performance of forests by introducing four types of forest: a forest of classification trees, a forest of PETs using relative frequency, a forest of PETs using Laplace estimate and a forest of PETs using the M-estimation. He concluded that Laplace and m-estimate have a similar performance compared to the forest

of classification trees and forest of PETs. He demonstrated that probability correction should only be employed in small forests of PETs and that for larger forests, classification trees and PETs are equally good alternatives.

3.2.3.4 Tree Calibration

Probability calibration methods are a similar way to smoothing to help improve the probabilities. Platt scaling (Platt et al., 1999) is a probability calibration method proposed by Platt for support vector machines (SVMs). In this method, predictions in the range $[-\infty, +\infty]$ are passed through a sigmoid function to produce probability estimates in the range $[0, 1]$. The sigmoid function is fitted with logistic regression. It produces probability estimates

$$P(y = 1|x) = \frac{1}{1 + exp(Af(x) + B)}.$$

i.e. a logistic transformation of the classifier score $f(x)$, where A and B are two scalar parameters that are learned by the algorithm. Note the prediction can now be made according to $y = 1$ iff $P(y = 1|x) > 1/2$; if $B \neq 0$, the probability estimates contain a correction compared to the old decision function. The parameters A and B are estimated using a maximum likelihood method that optimises on the same training set as that for the original classifier f. To avoid overfitting to this set, a held-out calibration set or cross-validation can be used, but Platt additionally suggests transforming the labels y to target probabilities

$$y_+ = \frac{N_+ + 1}{N_+ + 2}, \quad y_- = \frac{1}{N_- + 2}.$$

Here N_+ and N_- are the number of positive and negative samples, respectively. This transformation follows by applying Bayes's rule to a model of out-of-sample data that has a uniform prior over the labels. Platt scaling has been shown to be effective for SVMs as well as other types of classification models, including boosted models and even naive Bayes classifiers.

An alternative approach to probability calibration is to fit an Isotonic regression model to an ill-calibrated probability model. Isotonic regression is more general than Platt scaling because no assumptions are made about the form of the mapping function, other than it needs to be monotonically increasing (isotonic). This has been shown to work better than Platt scaling,

in particular, when enough training data is available.

Niculescu-Mizil and Caruana (2005) experimented with calibration methods, including Platt Scaling and Isotonic Regression, to correct the poor probability estimates predicted by decision tree classifiers. Both of Platt scaling and Isotonic regression can only be used for binary classification problems when a one-vs-rest method (Rifkin and Klautau, 2004) used for multiclass classification. Furthermore, Rüping (2006) shows that both of them are greatly affected by outliers in the probability space. In their research, Platt scaling is modified using methods from robust statistics to make the calibration less sensitive to outliers. Jiang et al. (2011) proposes to construct a smooth, monotonically increasing spline that interpolates between a series of representative points chosen from an isotonic regression function. Zhong and Kwok (2013) incorporates manifold regularisation into isotonic regression to make the function smooth, and adapt the technique to be better suited to calibrating the probabilities produced by an ensemble of classifiers, rather than a single classifier.

Leathart et al. (2017) proposed a probability calibration tree model, a modification of logistic model trees that identifies regions of the input space in which different probability calibration models are learned to improve performance. They compare their model to Platt scaling and Isotonic regression and show that their model results in lower root mean squared error on average than both methods, for estimates produced by a variety of base learners.

3.2.4 Summary

In this section, we introduced the C4.5 decision tree learning algorithm, analysed the bias and variance of decision trees and reviewed the existing works on improving the class probability estimation of decision trees. In **Chapter 5**, we propose a new hierarchical smoothing method HGS and compare it with the existing methods listed in this section.

Chapter 4

ESKDB Algorithm for Bayesian Network Classifiers

As we have discussed in **Section 1.3**, building the ESKDB (Ensemble model of SKDB) model is important. First, we know that Random Forest is a very successful ensemble model of decision trees, but we do not have an ensemble model of Bayesian network classifiers (BNCs) that can compete with Random Forest. Second, Bayesian network classifiers can analyse larger datasets because they can be learnt out-of-core, while Random Forest is an in-core learner. Third, SKDB can only deal with categorical data, but we would like to build an ensemble model that can deal with both categorical and numerical data. In this chapter, we show the details of building the ESKDB model and conduct an extensive set of experiments to compare it with existing models.

4.1 Introduction

With the rapid development of Web technologies in the last decades, large datasets are created everywhere, such as in social media, E-commerce and health care. In-core algorithms, e.g. Random Forest (RF) (Breiman, 2001) and Support Vector Machines (SVM) (Hearst, 1998), are less suited to large amounts of data because they require the data to be stored in main memory. Out-of-core learners – i.e. algorithms that can learn from a dataset without holding it fully in the main memory – appear to be more suited to large quantities of data because of their ability to scale. Bayesian network classifiers

are out-of-core learners and thus show great potential; instances of this class of classifiers include the famous Naïve Bayes (NB) (Lewis, 1998) algorithm, Tree Augmented Naïve Bayes (TAN) (Friedman et al., 1997), K-Dependence Bayes (KDB) (Sahami, 1996), as well as Selective KDB (SKDB) (Martınez et al., 2016). Note, SKDB was shown to be competitive to RF on categorical data.

A BNC is a directed acyclic graph whose nodes represent the variables of the dataset and edges indicate the direct dependencies between those variables. Generally the target or class variable is a parent of all other variables, with several additional connections existing between the other variables. The bias/variance trade-off of BNCs can be easily tuned by putting a limit on the maximum number of parents that a node can have. With k parents, the model then looks at all possible combinations of the $(k+1)$ nodes connected with each other. The higher the value of k, the lower the bias of the algorithm (and usually the higher the variance as well).

SKDB is a highly scalable BNC that achieves a good trade-off between structural complexity and classification performance by efficiently choosing the value of the maximum number of parents ($maxK$). For large datasets, a higher complexity or low-biased model is preferable because it allows the model to capture fine detail in data more precisely. But this low bias model has more parameters and potentially higher variance, leading to poorer predictions. As a result, more data is usually needed (or a superior parameter estimation technique, as we will see later).

In this chapter, we propose to ensemble the SKDB algorithm to both increase its accuracy as well as to make it applicable to numerical data. There are two broad frameworks for ensembling, Bayesian model averaging (Hoeting et al., 1999), first implemented for Bayesian networks in (Madigan et al., 1995), and the more frequentist style commonly associated with ensembles (Zhou, 2012) best illustrated by Random Forest (Breiman, 2001). We use the second broad framework because it is more suited to larger amounts of data.

The difficulty in creating an ensemble of a base classifier lies in varying the results of the original classifier without raising the bias of the base classifier, and while keeping the covariance between the (varied) classifiers low. To do this, we combine two sources of stochasticity: (1) we vary the order in which the variables are considered, which controls what combination of attributes will be considered and (2) we vary the discretisation for numerical

attributes, which allows different 'elemental' classifiers of the ensemble to consider different cut-points. We will detail in **Section 4.2** how these stochasticities are defined to obtain both high accuracy and diversity of the elemental classifiers composing the ensemble. Finally, we add a third component to ESKDB in using an advanced smoothing technique based on Hierarchical Dirichlet Processes (HDP): it allows to control the variance of ESKDB further and, as we will show, substantially improves accuracy and probability calibration.

We carry out an extensive set of experiments on 72 datasets with data quantity up to 5M examples. We start by performing a large sensitivity analysis to show the influence of each of the three contributions on our final ESKDB algorithm (varying the attribute order, varying the discretisation and advanced smoothing). Having shown that all three components indeed bring significant improvement to the classifier, we then proceed by showing how it compares with the state of the art methods. We start by comparing ESKDB to existing Bayesian network classifiers – NB, TAN, KDB, KDF, AODE and SKDB. We show that ESKDB significantly outperforms state-of-the-art BNCs. We then proceed to compare ESKDB to non-BNCs. Our results show that ESKDB significantly outperforms both XGBoost (Chen and Guestrin, 2016) with default parameterisation and Random Forest (Breiman, 2001) and that its performance is not significantly different from XGBoost with highly tuned parameters. We believe these are strong results because ESKDB can handle both numerical and categorical data while being able to perform with a limited number of passes over the data (and hence not needing to load the data in main memory such as RF and XGBoost). We finally complete our experiments section by studying the running time of ESKDB and give guidance on using different smoothing methods on ESKDB.
The main contributions of this chapter are as follows:

- A novel ensembling method for Bayesian network classifiers – ESKDB – with two novel elements:
  - we vary the attribute order over the variables by sampling orders following the mutual information with the class;
  - we vary the number of cut-points for the discretisation of each numerical attribute by sampling the information gain (and combined with the MDL stopping criterion).

- An improved probability estimation technique: we add a prior to existing Hierarchical Dirichlet Process smoothing techniques. The result is a sampler that requires only 10% percent of the samples compared to the state-of-the-art one proposed in (Petitjean et al., 2018).

Put together, these three components make ESKDB become the most accurate Bayesian Network Classifier to date. It achieves better accuracy, better probability calibration, can handle numerical attributes and does all these much faster than the state-of-the-art BNC – SKDB-HDP (Petitjean et al., 2018). We show that our classifier runs virtually parameter-free and significantly outperforms Random Forest and scores just behind a highly-tuned XGBoost algorithm.

4.2 ESKDB Algorithm

Like most other Bayesian network classifiers, the learning process for ESKDB consists of two steps: structure learning and parameter learning. The structure learning process is achieved by constructing E different SKDB classifiers, while the parameter learning process is to apply probability smoothing techniques to each SKDB classifier.

Algorithm 9 is the learning algorithm of ESKDB: it takes as inputs a training set $\mathcal{T}$ and an user-supplied ensemble size E. It returns an ensemble model $\mathcal{B}$ that consists of E different SKDB classifiers. The algorithm first initialises $\mathcal{B}$ to be an empty set (line 1). Then in the second step, it learns E different SKDB classifiers one by one and adds them into $\mathcal{B}$ (line 2-10). Last, it returns the ensemble model $\mathcal{B}$ (line 11).

The second step of learning the ESKDB model is to learn different SKDB classifiers one by one in the ensemble. To enable ESKDB to work with numeric attributes and further increase the diversity of ESKDB, ESKDB first learns the different cut points for numerical attributes before learning the structure and parameters of each SKDB classifier. The cut points $cutPoints_i$ for classifier $\mathcal{G}_i$ are learned by **Algorithm 10**, which will be explained in detail in the next section. The training data $\mathcal{T}$ is then discretized into $\mathcal{T}_i$ according to the $cutPoints_i$. The attribute order $\mathcal{O}_i$ for classifier $\mathcal{G}_i$ is learned by **Algorithm 11**, which will be described in detail in **Section 4.2.2**. After the cut points and attribute order are learnt, it then learns a standard SKDB classifier $learnSKDB(\mathcal{O}_i, \mathcal{T}, K)$ except that the attribute order $\mathcal{O}_i$ is given.

Last, it learns the parameters by applying probability smoothing methods, such as M-estimation and HDP, to improve the parameter estimates. Then complete the i_{th} SKDB classifier $\mathcal{B}_i$ by combining the $\mathcal{G}_i$ and Θ_i together.

Algorithm 9: learnESKDB($\mathcal{T}, E, K$)

Input : A training set $\mathcal{T}$
Input : an ensemble size E
Input : The maximum number of parents allowed in SKDB K
Output : An ESKDB model $\mathcal{B}$

Let $\mathcal{B} \leftarrow \emptyset$.
for $i \leftarrow 1$ *to* E **do**
    $cutPoints_i \leftarrow learnDiscretizer(\mathcal{T})$. `// Section 4.2.1`
    $\mathcal{T}_i \leftarrow discretize(\mathcal{T}, cutPoints_i)$
    $\mathcal{O}_i \leftarrow randomSampleForOrders(\mathcal{T}_i)$. `// Algorithm 12`
    $\mathcal{G}_i \leftarrow learnSKDB(\mathcal{O}_i, \mathcal{T}_i, K)$. `// (Martınez et al., 2016)`
    $\Theta_i \leftarrow learnParameters(\mathcal{G}_i, \mathcal{T}_i)$. `// build CPTs and smooth`
    $\mathcal{B}_i \leftarrow (\mathcal{G}_i, \Theta_i, cutPoints_i)$.
    $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{B}_i$.
end
return $\mathcal{B}$.

4.2.1 Randomized Discretization

The first source of stochasticity to make ESKDB more diverse is to make each classifier in the ensemble have its own cut points for continuous attributes. We achieve this by proposing a new randomised discretisation method for continuous attributes based on the method of MDLP (Fayyad and Irani, 1993).

Algorithm 10 is called in the ESKDB algorithm to learn the cut points for each classifier. For each of the numerical attribute in the training data, it first calculates all the candidate cut points. As done in the MDLP method, the candidate cut points are all the midpoint values P_A for each pair of adjacent attribute values after the example are sorted with the attribute value in ascending order. Second, it randomly selects cut points from P_A using **Algorithm 11**. Third, it gathers the selected cut points P_A^S to the set $cutPoints$ and returns.

The MDLP method calculates the entropy and the MDLP threshold for each cut point. The difference between the entropy and the threshold can be easily calculated. The points with a positive difference satisfy the MDLP

Algorithm 10: learnDiscretizer($\mathcal{T}$)

Input : A dataset T

Output : A set *cutPoints* containing the selected cut points for each continuous attribute

```
1 Let cutPoints ← ∅
2 for each attribute A ∈ 𝒯 do
3     if A is numerical then
4         P_A ← allPossibleCutPoints(A)// First, arrange all the
              values of A from smallest to largest, then
              calculate the middle value of every two values as
              one of the possible cut points of A. To make
              ESKDB model out-of-core, we only select a subset
              of T to calculate the possible cut points.
5         P_A^S ← randomSampleCutPoints(𝒫_A, true) // Algorithm 11
6         cutPoints ← cutPoints ∪ P_A^S
7     end
8 end
9 return cutPoints.
```

criteria and can be accepted. The cut point with the largest difference is selected as the best cut point. Then the same process is conducted recursively for the left subset and the right subset. The selection will be stopped when none of the points in the current set can meet the MDL criteria.

In our method, instead of selecting the best cut point, we pick one at random from all the points that satisfy the MDLP criteria. The random sampling can be done by first building a probability vector over all the cut points and then normalizing it into a probability distribution. The value in the vector is the difference between the entropy of each cut point and the MDLP threshold. Points with negative difference value will have probability set to 0 to guarantee that they are not selected. After randomly selecting a cut point from this distribution, the selection process can be applied recursively to the left and right subset (as in the deterministic case) until none of the points meets the MDLP criteria.

In the extreme case that all points do not meet the MDLP metric the first time a random selection is performed, the selection will be stopped with no point being selected.. This is not appropriate for an ensemble model because it greatly reduces ensemble diversity. To allow for more diversity, we return at least one cut point in this case, selected by normalising the entropy into a

probability distribution and sampling from it.

Algorithm 11 shows how to randomly sample cut points. The algorithm is easier to understand with the above ideas explained. The input is a set of possible cut points $\mathcal{P}$ and a flag $firstFlag$. The output is a set of the selected cut points $\mathcal{P}^s$. The algorithm starts by calculating the entropy vector IG and the difference vector IG^t (line 3-14). If the extreme case occurs, i.e. the $\sum_{i=1}^{|IG^t|} IG_i^t = 0$ and $firstFlag = true$, normalize the entropy vector IG, samples a cut point and return it (line 16-26), otherwise, sample a cut point from the normalized IG^t and recursively selects point for the left subset $\mathcal{P}_l = \{p \in \mathcal{P} | p < p^s\}$ and the right subset $\mathcal{P}_l = \{p \in \mathcal{P} | p > p^s\}$ (line 28-35).

4.2.2 Randomized Attribute Ordering

The second source of stochasticity to ensure ESKDB has more diversity is sampling the attribute order for each SKDB classifier in our ESKDB model. As discussed in **Section 3.1**, the attribute order determines the dependencies between the attributes. The top selected attribute is more likely to be selected as the parent of another attribute. SKDB and KDB both calculate the Mutual Information $MI(A_i; Y)$ to measure the correlation between attributes A_i and the class Y. The attribute order is decided by sorting the attributes with the MI in descending order.

We know from experience that the larger the MI value, the more likely the attribute is to be chosen first in the attribute order. This motivated us to normalise the MI into a probability distribution to do the attribute sampling. The method returns $\mathcal{O}$ a sampled attribute order to build an SKDB classifier.

4.2.3 Randomized Parent Ordering

The parent order of each attribute is the parent nodes in order of CMI values from largest to smallest. Instead of choosing the best parent order for each attribute, we can also increase ESKDB's diversity by ensembling on the parent order of the SKDB classifier. However, the results of this approach have not improved the performance of ESKDB much. It can be better explained by an example. Assume that the parent order for attribute X_3 is (X_1, X_2) and we would like to estimate the probability $P(X_3|X_1, X_2)$. However, the difference between $P(X_3|X_1, X_2)$ and $P(X_3|X_2, X_1)$ is not big. If we learn the probability using single-layer methods, such as MLE,

Algorithm 11: $randomSampleCutPoints(\mathcal{P}, firstFlag)$

Input : possible cut points $\mathcal{P}$
Input : $firstFlag \leftarrow true$ if this is the first time this method been called to ensure we have at least 1 cut point
Output : the selected cut points $\mathcal{P}^s$

$\mathcal{P}^s \leftarrow \emptyset$
/* we sample cut points by building a probability distribution over all the possible cut points. */
Let IG be a vector of the information gain for all the cut points in $\mathcal{P}$
Let IG^t be a vector of the information gain minus the MDL threshold.
for *each cut point* $p \in \mathcal{P}$ **do**
    $IG_p \leftarrow InformationGain(p)$
    $threshold_p \leftarrow MDL(p)$
    // refer to
    $IG^t_p \leftarrow IG_p - threshold_p$
    if $IG^t_p < 0$ **then**
        // if p doesn't meet the MDL criterion, then p will not be selected
        $IG^t_p \leftarrow 0$
    end
end
/* if all the cut points do not meet the MDL criterion and $firstFlag = true$, we assume that we get at least 1 cut point */
if $\sum IG^t = 0$ **then**
    if $firstFlag$ **then**
        Normalize IG into a probability distribution.
        if $\sum IG \neq 0$ **then**
            $p^s \sim IG$
            $\mathcal{P}^s \leftarrow \mathcal{P}^s \cup \{p^s\}$
            return $\mathcal{P}^s$
        end
    end
    return $\emptyset$
end
/* Otherwise, at least one cut point meet the MDL criterion */
Normalize IG^t into a probability distribution.
$p^s \sim IG^t$
$\mathcal{P}^s \leftarrow \mathcal{P}^s \cup \{p^s\}$
/* recursively calling the left and right of p^s */
$\mathcal{P}_l \leftarrow \{p \in \mathcal{P} | p < p^s\}$
$\mathcal{P}_r \leftarrow \{p \in \mathcal{P} | p > p^s\}$
$\mathcal{P}^s \leftarrow \mathcal{P}^s \cup randomSampleCutPoints(\mathcal{P}_l, false)$
$\mathcal{P}^s \leftarrow \mathcal{P}^s \cup randomSampleCutPoints(\mathcal{P}_r, false)$
return $\mathcal{P}^s$

Algorithm 12: randomSampleForOrders($\mathcal{T}$)

Input : A training set $\mathcal{T}$ with attributes $\mathcal{A}$
Output : Orders $\mathcal{O}$

$n \leftarrow |\mathcal{A}|$
MI $\leftarrow \{MI_i \leftarrow 0, i = 1, ..., n\}$
for *each* $A_i \in \mathcal{A}$ **do**
    $MI_i \leftarrow mutualInformation(A_i; Y)$
end
Sort **MI** in decreasing order.
Reorder $\mathcal{A}$ corresponding to the sorted **MI**.
Let $\mathcal{O} \leftarrow \emptyset$.
while $|\mathcal{O}| < n$ **do**
    $MI \leftarrow normalize(MI)$ `// normalize` MI `into a probability distribution`
    $z \sim MI$ `// random sample from` MI
    $\mathcal{O} \leftarrow \mathcal{O} \cup \mathcal{A}_z$.
    $MI_z \leftarrow 0$. `// make sure` A_z `will not be selected again`
end
return $\mathcal{O}$

Laplace smoothing or M-estimation, the results are the same. Only with the hierarchical probabilistic smoothing method does the result change a bit. In consideration of increasing the diversity of SKDB classifiers in ESKDB, we do not consider working on parent order in building the final ESKDB model. In fact, the parent order is heavily influenced by the attribute order, because only attributes that are sorted earlier than the current attribute can be selected as its parent.

4.2.4 Parameter Learning

The two sources of stochasticity mentioned above are to make each SKDB classifier in the ESKDB model have a different network structure. After the structures are learnt, parameter learning is required for each classifier. **Algorithm 13** shows the details of learning the parameters for each SKDB classifier in ESKDB. Instead of using frequencies calculated by MLE, probability smoothing techniques, such as HDP smoothing and M-estimation, can be applied to improve the probability estimates of the parameters in this step.

Algorithm 13: learnParameters($\mathcal{G}, \mathcal{T}$)

Input : Training set $\mathcal{T}$ and classifier $\mathcal{G}$
Output : parameter Θ

Configure Θ for structure $\mathcal{G}$.
Collect the Conditional Probability Tables for Θ by going through $\mathcal{T}$.
Apply HDP smoothing or M-estimation for each of the conditional tree.
return Θ

4.2.5 Improved HDP Smoothing

A critical issue for HDP estimation shown in (Petitjean et al., 2018) is its computational complexity. They show that the Gibbs sampler with 50,000 iterations gives the most accurate probability estimates, and they believe that even more iterations could further improve accuracy. The training time complexity increases linearly with the number of iterations, and 50,000 iterations make HDP extremely slow.

In this section, we show how to improve this dramatically. In our new method, only 1,000 iterations could give very accurate estimates. We achieve this by adding more carefully thought out priors on the concentration parameter. **Algorithm 14** describes the sampling for concentration parameters in the tree, taken from (Buntine and Mishra, 2014). In the experiments, we use a $Gamma(2, 1)$ prior instead of the uniform prior of (Petitjean et al., 2018), which corresponds to having $priorShape = 2$ and $priorRate = 1$ in **Algorithm 14**. This makes α have a prior mean of 2 and a prior standard deviation of 1.4, and this is a default prior reported to work well with advanced topic models (Buntine and Mishra, 2014).

4.2.6 Testing Algorithm

Given a test sample, ESKDB USES each SKDB classifier to test the sample and simply averages all the predictions as the final prediction. A very important point in the test algorithm is that since each SKDB $\mathcal{B}_i$ has its own cut points $\mathcal{B}_i.cutPoints$ for numerical attributes, the test sample needs to be discretized with the cut points of each SKDB classifier before making predictions. **Algorithm 15** shows the details of the $testESKDB$ algorithm.

Algorithm 14: sampleConcentration(α, *nodes*, *priorShape*, *priorRate*)

Input : α: concentration to sample
Input : *nodes*: nodes sharing this concentration parameter (tying)
Input : *priorRate*: prior on rate
Input : *priorShape*: prior on shape

$rate \leftarrow priorRate$
$sumTk \leftarrow 0$
for *each node* $\in$ *nodes* **do**
  $q \sim Beta(\alpha, node.n)$
  $rate \leftarrow rate - log(q)$
  $sumTk \leftarrow sumTk + node.t$
end
$\alpha \sim Gamma\,(sumTk + priorShape\,,\, rate)$
for *each node* $\in$ *nodes* **do**
  $node.\alpha \leftarrow \alpha$
end

Algorithm 15: testESKDB(*test*, $\mathcal{B}$)

Input : A test example *test*
Input : an ESKDB classifier $\mathcal{B}$ with E classifiers and K classes
Output : A predicted probability distribution *res*

for $i \leftarrow 1$ *to* E **do**
  $test_i \leftarrow discretizeExample(test, \mathcal{B}_i.cutPoints)$
  $res_i \leftarrow testSKDB(test_i, \mathcal{B}_i)$
  $res \leftarrow res \cup res_i$
end
return $normalize(res)$.

4.3 Experiment Results

The aim of this section is to compare the performance of our ESKDB model with out-of-core BNCs (NB, TAN, KDB, SKDB, AODE and KDF) in **Section 4.3.2**. The advantages of the two sources of stochasticity in ESKDB is demonstrated in **Section 4.3.3**. The ensemble size is learned in **Section 4.3.4**. The parameters of HDP smoothing is learned in **Section 4.3.5**. Different probability smoothing techniques, such as HDP smoothing and M-estimation, are compared in **Section 4.3.6**. XGBoost and RF, as the two most powerful tree ensemble models, are also compared in **Section 4.3.7**. We compare ESKDB with the fully discretized data in **Section 4.3.8**. Last,

we compare the running time of ESKDB and other models in **Section 4.3.9**. In **Section 4.3.1**, we give the general settings for our experiments.

4.3.1 Experiment Design and Setting

Design: An extensive set of experiments are conducted on 72 standard datasets, where most of them are from the UCI archive (Lichman, 2013), but some larger datasets are also included from (Martınez et al., 2016). **Table 3** summarizes the characteristics of 72 datasets, including the dataset name, size and number of attributes. A missing value is treated as a separate attribute value.

Evaluation Measure: We use 5 times 2-fold cross-validation for all the methods because it has lower variance than 10-fold cross-validation. Although, for the six largest datasets in **Table 3**, only a single run of 2-fold cross-validation is required because the test set is sufficiently large. The results are assessed by RMSE and 0-1 Loss. Win-Draw-Loss (WDL) is used when comparing two different models. A one-tail binomial sign test is used to determine the significance of the results, using $p \leq 0.05$. RMSE is the square root of the Brier score in discrete contexts, which is used to measure how well-calibrated the probability estimates are. We use RMSE as the most important measure because the main research goal of this paper is to improve the probability estimates, a reasonable proxy for a variety of other decision contexts.

Software: To ensure reproducibility of our work and allow other researchers to build on our research easily, we have made our source code for ESKDB available on Github [1].

Compared models and parameters: In our experiment, we use BNCs including NB, TAN, KDB, SKDB, AODE, KDF and ESKDB, and tree-based ensemble models including RF and XGBoost. We compare HDP and M-estimation as the smoothing techniques for them. Here M-estimation is used with a backoff strategy, which means when the leaf node has no data, back off the probability estimates to its nearest non-empty ancestor. The list of parameter settings is shown in **Table 4**.

[1] https://github.com/icesky0125/ESKDB-on-numerical-data

Table 3: Datasets for ESKDB.

Domain	Case	Att	Domain	Case	Att
Donation	5,749,132	12	Vowel	990	14
Poker-hand	1,025,010	11	Tic-Tac-Toe	958	10
Census	299,285	42	Anneal	898	39
Skin-Segment	245,057	4	Vehicle	846	19
Localization	164,860	6	PIndiansDiabetes	768	9
Diabetes	101,766	47	BreastCancer-w	699	10
Connect-4	67,557	43	Credit Screening	690	16
Shuttle	58,000	10	Balance Scale	625	5
Adult	48,842	15	Syncon	600	61
Letter Recognition	20,000	17	Chess	551	40
Magic	19,020	11	Cylinder	540	40
Nursery	12,960	9	Musk1	476	167
Sign	12,546	9	House Votes84	435	17
Pen Digits	10,992	17	Horse Colic	368	22
Thyroid	9,169	30	Dermatology	366	35
Mushrooms	8,124	23	Ionosphere	351	35
Musk2	6,598	167	Primary Tumor	339	18
Satellite	6,435	37	Heart Disease-c	303	14
Optical Digits	5,620	49	Hungarian	294	14
Texture	5500	41	Audiology	226	70
Page Blocks	5,473	11	New-Thyroid	215	6
Wall-following	5,456	25	Glass-id	214	10
Nettalk(Phoneme)	5,438	8	Sonar	208	61
Waveform-5000	5,000	41	Autos	205	26
Spambase	4,601	58	Wine	178	14
Abalone	4,177	9	Hepatitis	155	20
Hypothyroid	3,772	30	Teaching Assistant	151	6
Sick	3,772	30	Iris	150	5
Kr vs. kp	3,196	37	Lymphography	148	19
Splice-C4.5	3,190	62	Echocardiogram	131	7
Segment	2,310	20	Promoters	106	58
Car	1,728	8	Zoo	101	17
Yeast	1,484	9	Post-operative	90	9
Contraceptive-mc	1,473	10	Labor	57	17
German	1,000	21	Lung Cancer	32	57
LED	1,000	8	Contact-lenses	24	5

Table 4: List of parameters.

Methods	Parameters
BNCs	$K = 5$ (Number of maximum parents allowed for each feature) $E = 10$ (ensemble size for ESKDB)
HDP	$Iteration = 1000$ (Gibbs sampling iteration) $BurnIn = 100$ (Collect counts after sampling for 100 times) $Tying$ (tying some nodes to share the same concentration)
M-estimation	$M = 1/C$ (the value for calculating M-estimation, C is the number of class labels)
RF	$F = 100$ (number of trees in random forest) $Atts = log_2(n) + 1$ (number of splitting attributes allowed)
XGBoost	$objective = softprob$ (softmax objective) $rounds = 100$ (the number of rounds for boosting) others = default

4.3.2 ESKDB is better than existing BNCs

In this section, we compare several single BNCs, including NB, TAN, KDB and SKDB with our proposed ensemble model ESKDB using M-estimation. This is to show the benefit of ensembling of our model. Besides, we also compare ESKDB with two other existing ensemble models of BNCs, including AODE and KDF. This is to show our ensembling technique is better than the others. The ensemble size of ESKDB is set to be 10 in this part.

Figure 9 shows the averaged RMSE and 0/1 Loss over all the datasets for each model mentioned above, which are represented by blue curve and red curve respectively. It clearly shows that our ESKDB with only 10 classifiers gets much better performance than other models both on RMSE and 0/1 loss.

Table 5 shows the Win-Draw-Loss result of comparing ESKDB with the existing BNC models. Values in boldface are statistically significant tested by a one-tailed binomial sign test. A difference is considered to be significant if $p \leq 0.05$. It can be seen from this table that ESKDB is significant better than all of the existing Bayesian network classifiers both on RMSE and error rate.

4.3.3 The benefits of the two stochasticities in ESKDB

In this section, we show the benefit of the two stochasticities in our ESKDB model. The first one is that the cut points are randomly selected for each

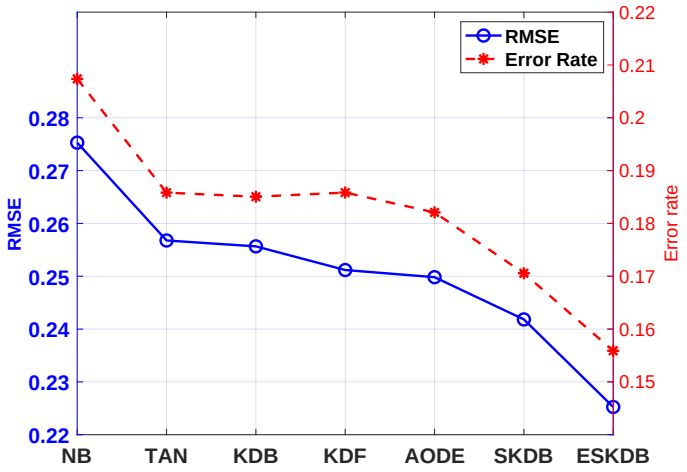

Figure 9: The blue curve is the RMSE for NB, TAN, KDB, SKDB, KDF, AODE and our proposed ESKDB with 10 SKDB classifiers. The red curve is the corresponding 0/1 loss. All the values are averaged over all the datasets listed in **Table 3**.

.

Table 5: Win-Draw-Loss for the ESKDB compared with existing BNCs. The value in boldface is statistically significant better tested by a one-tailed binomial sign test. A difference is considered to be significant if $p \leq 0.05$.

ESKDB vs.	RMSE	0/1 Loss
NB	**63-0-9**	**60-1-11**
TAN	**64-0-8**	**57-2-13**
KDB	**64-0-8**	**61-2-9**
KDF	**71-0-1**	**67-2-3**
AODE	**62-1-9**	**60-0-12**
SKDB	**60-0-12**	**59-3-10**

SKDB classifier in the ensemble. The second one is that the attribute order of each SKDB is randomly selected.

First, to show the benefits of the first stochasticity, we compare ESKDB with ESKEB_randomO, which represents ESKDB with stochasticity on attribute orders only. **Figure 10** is the scatter plot ESKDB with ESKDB_randomO on both RMSE and 0/1 Loss. The dots above the diagonal line represent ESKDB performs better on these datasets, and the dots below the diagonal line represent the other model is better. It can be seen

from this figure that the random selection of the cut points makes ESKDB perform better both on RMSE and 0/1 Loss compared with same cut points in ESKDB_randomO. The averaged RMSE value of ESKDB and ESKDB_randomO are 0.2248 ± 0.016 and 0.2326 ± 0.017, respectively. This means that the random selection of cut points makes an ensemble model have more diversity.

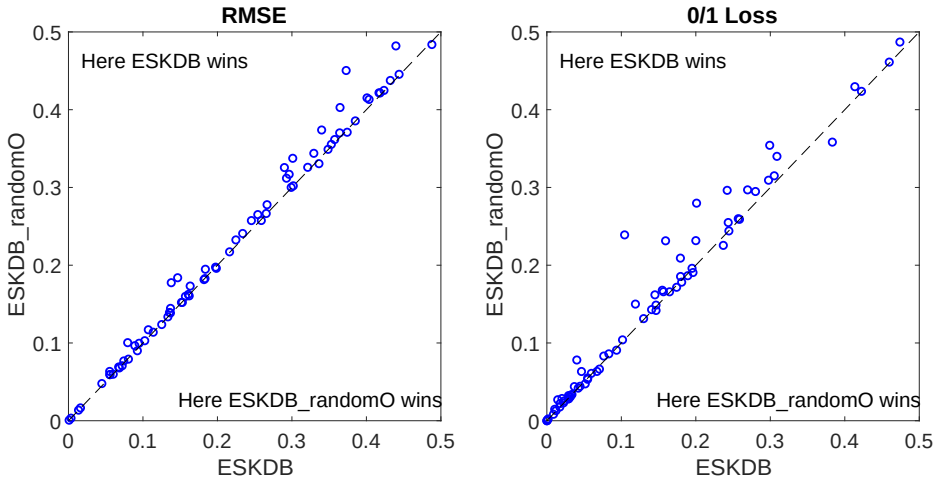

Figure 10: Scatter plot of ESKDB with ESKDB_randomO.

Second, to show the benefits of the random selection of the attribute order for each SKDB classifier, we compare ESKDB with ESKDB_randomCP, which represents ESKDB but where the attribute orders are not sampled. **Figure 11** is the scatter plot of ESKDB and ESKDB_randomCP. It can be seen from this figure that ESKDB performs slightly better than ESKDB_randomCP, which indicates that the second stochasticity of the attribute orders can also make our ESKDB model have more diversity and get better results.

Two compare the importance of these two randomnesses, we show the scatter plot of ESKDB_randomCP with ESKDB_randomO, as shown in **Figure 12**. It can be seen from this figure that the two versions of ESKDB have a quite similar effect on ESKDB. The averaged RMSE value of ESKDB_randomO and ESKDB_randomCP are 0.2326 ± 0.017 and 0.2312 ± 0.017, respectively. This indicates that the randomness of the cut points makes ESKDB slightly better than the randomness of the attribute

orders.

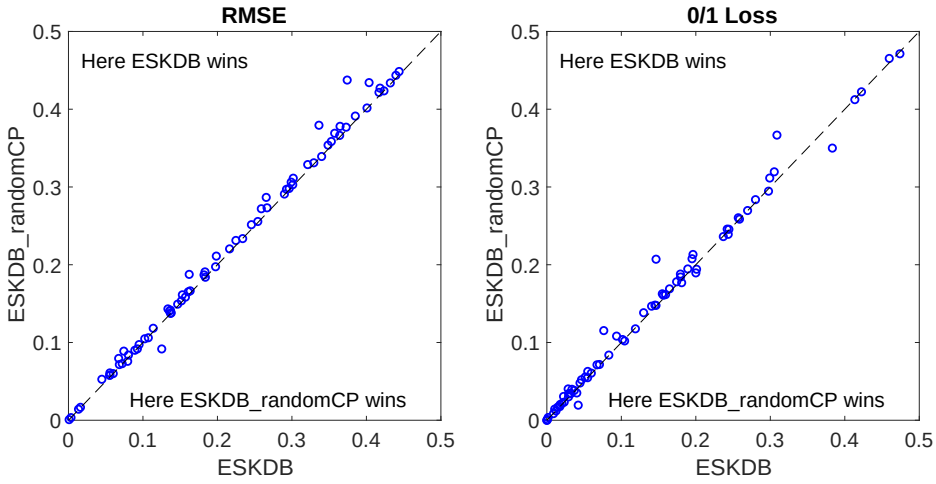

Figure 11: Scatter plot of ESKDB with ESKDB_randomCP.

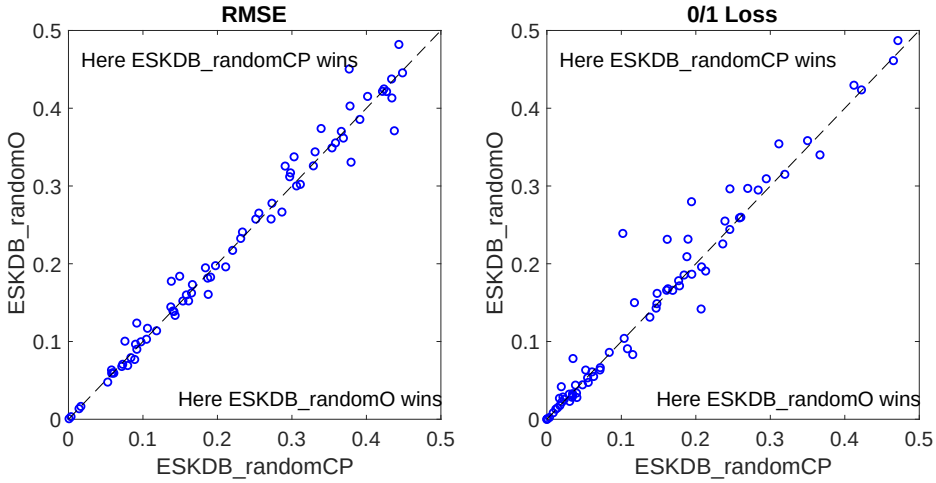

Figure 12: Scatter plot of ESKDB_sameO compared with ESKDB_sameP.

4.3.4 The ensemble size of ESKDB

In this part of the experiment results, we show how ESKDB performs with different ensemble sizes, including 1, 10, 20, and 30. **Figure 13** shows that

both RMSE and 0/1 Loss has a big improvement when ensembling on 10 SKDB classifiers compared with one single classifier. However, we increase the ensemble to 20 and 30, but further improvement is not as big. This indicates that our ESKDB with only 10 classifiers already gives good results. In the following experiments, 10 is used for ESKDB.

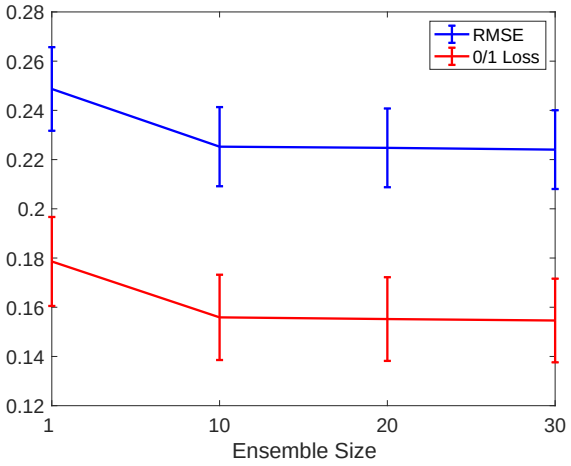

Figure 13: The RMSE and 0/1 Loss changes as the increasing of the ensemble size.

4.3.5 Improved HDP

In this part of the experiment results, we test the four different tying strategies of HDP smoothing on ESKDB and the benefit of the new Gamma prior, as was discussed in Chapter 2. **Figure 14** shows the RMSE of HDP smoothing using four different tying strategies and two different priors. It can be seen from this figure that no matter which tying strategy HDP used, the result of the new Gamma(2,1) is always better than the uniform prior. Besides, the *Level* strategy gets the best performance among all the four tying strategies. The *Level* tying strategy and the $Gamma(2,1)$ are used in the following experiments.

4.3.6 HDP compared with M-estimation for ESKDB

HDP has been shown for single Bayesian network classifiers to achieve more accurate parameter estimates compared with M-estimation. Here we aim

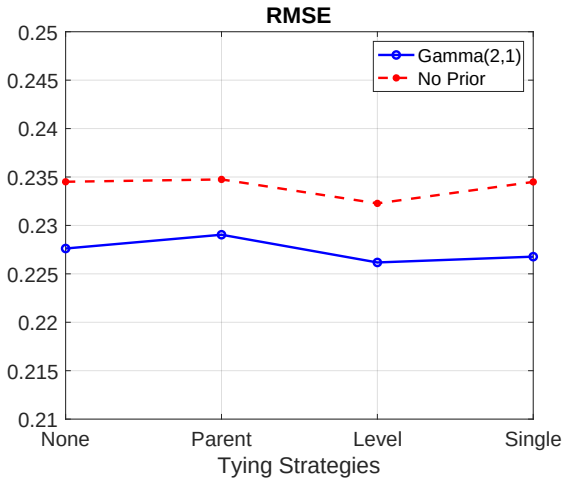

Figure 14: The HDP smoothing using different tying strategies and the new prior.

to verify that HDP could also improve the performance of ensemble models. Note that for decision trees, conventional wisdom is that no smoothing leads to superior results with ensembles, but we need to check the case for Bayesian network classifiers.

Table 6 is the WDL of HDP compared with M-estimation applied to ESKDB. We can see from this table that HDP outperforms M-estimation on ESKDB both on RMSE and on 0/1 Loss. This indicates that HDP smoothing is helpful both to single and ensemble models. The averaged performance for HDP and M-estimation are shown in **Table 7**. HDP makes the RMSE improve from 0.2252 to 0.2227 with a cost of 7 times longer training time.

Both M-estimation and HDP can get good estimates, but they have advantages and shortcomings. M-estimation on ESKDB gets good results compared with the state-of-the-art classifiers with only a limited learning time. HDP achieves better performance but with a cost of computation. We recommend the use of HDP for ESKDB and use that configuration in the remainder of this paper.

4.3.7 ESKDB compared with XGBoost and RF

In this section, we compare our ESKDB model with the two state-of-the-art ensemble classifiers: Random Forest (RF) and XGBoost. RF is built with 100

Table 6: Win-Draw-Loss for the ESKDB using HDP and M-estimation. The value in boldface is statistically significant better tested by a one-tailed binomial sign test. A difference is considered to be significant if $p \leq 0.05$

Classifiers	RMSE	0/1-loss
ESKDB_HDP vs ESKDB_M	**48-1-23**	**38-7-27**

Table 7: Averaged performance of ESKDB using HDP and M-estimation.

Smoothing	RMSE	0/1-loss	Time (s)
M-estimation	0.2252 ± 0.016	0.1559 ± 0.017	25
HDP	0.2227 ± 0.016	0.1536 ± 0.017	190

trees, to pure leaves, and with the number of randomly selected features to be $atts = log_2(|A|) + 1$ where $|A|$ is the total number of attributes. XGboost is an advanced implementation of gradient boosting algorithm. We present two versions of XGBoost: XGBoost$_{default}$, which uses default parameters ($max\_depth = 6$ and $num\_rounds = 100$); and XGBoost$_{tuned}$, for which we tune max_depth and num_rounds using 10-fold cross-validation (we use values $\{10, 50, 100\}$ for num_rounds and $\{2, 4, 6, 8\}$ for max_depth).

Our ESKDB method runs basically parameter-free, with the only parameter controlling the maximum depth of our trees – $maxK$ – which we set to 5. Note that the higher $maxK$, the higher the accuracy as the actual value of K is cross-validated internally with a fast leave-one-out cross-validation in SKDB.

Tables 8 gives the average classification performance over all the datasets listed in **Table 3**. We can observe from **Table 8** that, compared to RF, ESKDB has similar error rate but much better probabilities. This result also holds when compared to XGBoost ran with default parameters. Performing a grid-search for the best XGBoost parameters allows it to move ahead of the competition. It is important to remember here that the aim of this paper is not to show that ESKDB should now replace any other algorithm 'on the market', but rather that it is possible to obtain very accurate classifiers based on Bayesian network classifiers. Also, note that our algorithm runs completely out-of-core while XGBoost does require large amounts of data to

Table 8: ESKDB compared with RF and XGBoost

Classifier	RMSE	0/1 Loss	Time (s)
RF	0.2314 ± 0.016	0.1563 ± 0.018	6.8
XGBoost$_{default}$	0.2288 ± 0.018	0.1565 ± 0.018	2.7
ESKDB$_{M}$	0.2252 ± 0.016	0.1559 ± 0.017	25
ESKDB$_{HDP}$	0.2227 ± 0.016	0.1536 ± 0.017	190
XGBoost$_{tuned}$	0.2179 ± 0.017	0.1496 ± 0.017	60

Table 9: WDL of ESKDB compared with XGBoost and RF. The value in boldface is statistically significantly better.

Classifier	RMSE	0/1 Loss
ESKDB$_{M}$ vs. RF	**40-0-32**	30-2-40
ESKDB$_{M}$ vs. XGBoost$_{default}$	38-1-33	30-3-39
ESKDB$_{M}$ vs. XGBoost$_{tuned}$	27-0-45	26-2-44
ESKDB$_{HDP}$ vs. RF	**42-0-30**	36-2-34
ESKDB$_{HDP}$ vs. XGBoost$_{default}$	**44-0-28**	36-1-35
ESKDB$_{HDP}$ vs. XGBoost$_{tuned}$	33-1-38	29-2-41

be stored in memory and on-disk. **Table 9** gives a similar story with ESKDB finishing just behind a tuned version of XGBoost.

Figure 15 shows the comparison between ESKDB$_{HDP}$ with XGBoost$_{default}$ and XGBoost$_{tuned}$ in detail (ESKDB$_{HDP}$ wins above the diagonal line). This plot is interesting in that it shows the diversity of results obtained by XGBoost and ESKDB, with the results quite spread on either side of the diagonal line. This tends to indicate that there is some important benefit in having ESKDB available because even if the average RMSE is better for a tuned version of XGBoost, there are still many datasets for which ESKDB obtains a significant improvement over it. One point stands out particularly on the right scatter plot with XGBoost$_{tuned}$ obtaining almost a 0.2 improvement over ESKDB$_{HDP}$, which corresponds to the "tic-tac-toe" datasets. This toy dataset is very particular as it contains all the end-game boards of a 'tic-tac-toe' game, and only a single instance for each one (9 attributes, with 3 possible values – empty, cross or circle). It requires deep structures to represent it, with many combinations of 3 attributes to represent all combinations of aligned crosses. Unfortunately, for this dataset, SKDB

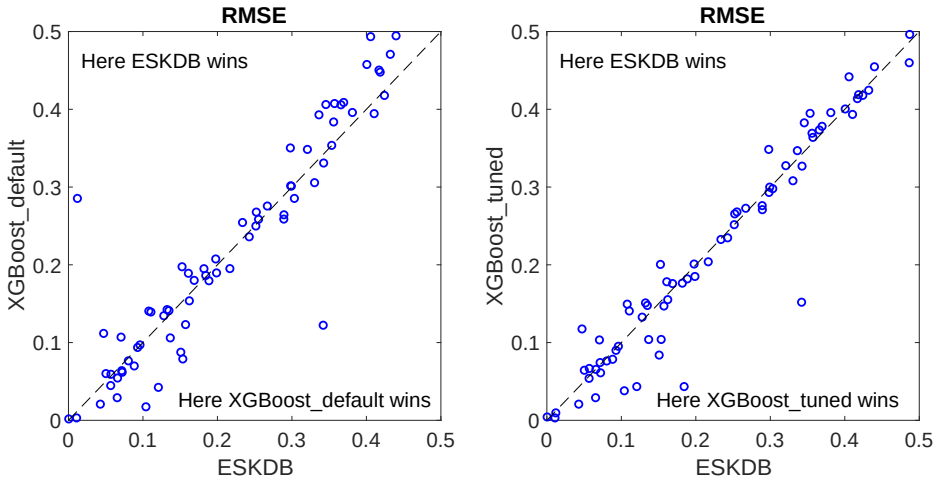

Figure 15: Scatter plot of ESKDB$_{HDP}$ with XGBoost$_{default}$ and XGBoost$_{tuned}$ on RMSE

Table 10: ESKDB compared with RF and XGBoost without "tic-tac-toe" dataset

Classifier	RMSE	0/1 Loss
RF	0.2304 ± 0.016	0.1573 ± 0.018
XGBoost$_{default}$	0.2303 ± 0.018	0.1584 ± 0.018
ESKDB$_{HDP}$	0.2210 ± 0.016	0.1535 ± 0.017
XGBoost$_{tuned}$	0.2191 ± 0.017	0.1517 ± 0.017

chooses to use $k = 2$, making it impossible to obtain accurate estimates of the probabilities. If we remove that particular synthetic data, ESKDB$_{HDP}$ gets extremely close to XGBoost$_{tuned}$ – see **Table 10** – which shows the high relevance of our proposed approach.

4.3.8 ESKDB on the fully discretized data

Strictly speaking, attribute discretisation should only use the training dataset. However, in many cases, data are discretised before training, that is, discretisation also uses the information of test data to find the cut points. Now we compare the difference between these two approaches for ESKDB. **Figure 16** is the scatter plot of ESKDB compared with ESKDB_whole, which

is discretisation on the whole dataset. As shown in **Table 11**, the averaged RMSE of ESKDB and ESKDB_whole are 0.2248 and 0.2204, respectively. Even though discretisation on the whole dataset gives a slightly better result, but it is a kind of cheating of using the test. Although no information about the test is used in ESKDB, it has little difference from ESKDB_whole.

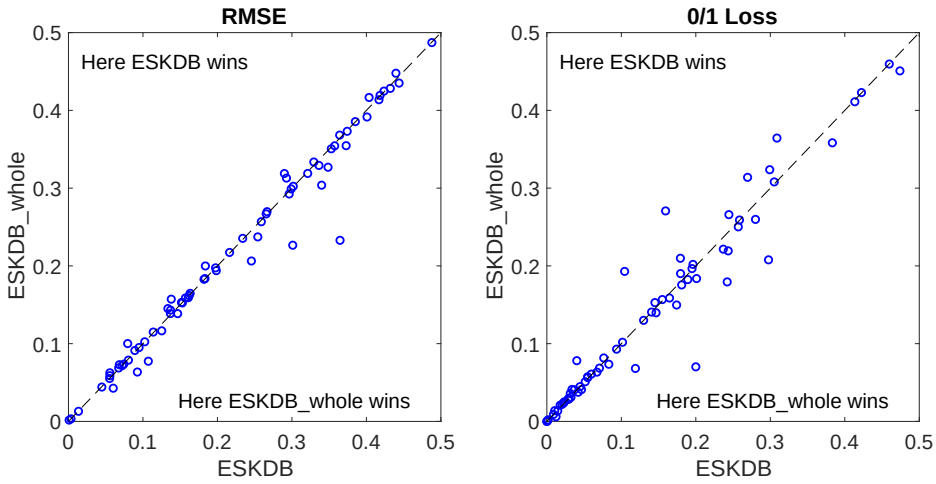

Figure 16: Scatter plot of discretization on only training set and the whole dataset.

Table 11: ESKDB compared with ESKDB learned on the fully discretized data.

Classifier	RMSE	0/1-loss
ESKDB	0.2248 ± 0.016	0.1565 ± 0.018
ESKDB_whole	0.2204 ± 0.016	0.1552 ± 0.017

4.3.9 Running Time

In this section, we compare the training times for classifiers mentioned in the experiment part of ESKDB. Table 12 lists the averaged results over all the datasets. As can be seen from this table, it takes less than 5 seconds for NB, KDB, AODE and the default version of XGBoost to be learned. Both of TAN and KDF need 5.7 seconds, and RF needs 6.8 seconds to be

Table 12: Averaged training time (seconds) for all the datasets

Classifier	RMSE	0/1 Loss	Time (s)
XGBoost$_{default}$	0.2288	0.1565	2.7
NB	0.2753	0.2074	3
KDB	0.2557	0.1851	3.6
AODE	0.2498	0.1821	5
TAN	0.2568	0.1858	5.7
KDF	0.2512	0.1858	5.7
RF	0.2314	0.1563	6.8
SKDB	0.2418	0.1706	13.7
ESKDB$_M$	0.2252	0.1565	25
XGBoost$_{tuned}$	0.2179	0.1496	60
ESKDB$_{HDP}$	0.2227	0.1536	190

learned. The parameter-tuned XGBoost needs 60 seconds to be learned, which is around 22 times longer than XGBoost$_{default}$. This indicates that the parameter tuning for XGBoost is time-consuming. ESKDB$_M$ with 10 SKDBs needs 25 seconds, which is 1.8 times longer than SKDB. HDP makes ESKDB 7.6 times longer than M-estimation because of the 1,000 iterations of Gibbs sampling used. It can be seen from this table that ESKDB_M is 2.4 times faster than XGBoost_tuned and ESKDB_HDP is 3x slower than XGBoost_tuned, which we regard as a good result given the amount of engineering that went into that classifier. However, like RF, ESKDB scales linearly with the ensemble size (10 used here).

4.4 Summary

In this chapter, we described the details of the ESKDB algorithm, including the motivation, the algorithm details and the experiment results. Experimental results show that ESKDB clearly outperforms all existing BNC algorithms (including ensembles such as KDF and AODE) both in terms of accuracy, probability estimation and functionally, as being able to handle numerical attributes. We then show that ESKDB obtains better-smoothed probabilities than Random Forest, which is critical for situations where the confidence in the predictions is important. We finish by showing that ESKDB performs competitively to XGBoost, depending on how much time is used to tune

XGBoost's hyper-parameters.

Our ESKDB algorithm can be used either with simple Laplace-type smoothing (such as Laplace and M-estimation) or with our improved HDP estimator. This choice mostly trades off running time vs quality of the estimates: ESKDB$_{HDP}$ runs approximately 8x slower than ESKDB$_{M}$ but is able to have significantly better RMSE than an untuned XGBoost. It is also important here to underline that both versions of ESKDB can run without having to load the whole dataset in memory, in contrast to RF and XGBoost.

Chapter 5

HGS Smoothing Algorithm for Decision Trees

As we discussed in Chapter 1, many classification tasks require accurate class probability estimates rather than class labels. We also mentioned in Chapter 2 the reasons why decision trees cannot provide accurate class probability estimates and introduced the existing methods. In this chapter, we propose a new and more efficient hierarchical probability smoothing algorithm called Hierarchical Gradient Smoothing (HGS) for decision trees.

5.1 Motivation

One might ask why would one need a decision tree when the Random Forest model is much better than a decision tree for many areas. Decision trees are still seeing use in online, non-stationary and embedded contexts, as well as for interpretability. More recently, with the advent of many more learning tasks, such as online learning, or learning where the inference system have low computational resources, a single decision tree is seeing a resurgence. Extremely fast decision trees are one of the top performers for high-data-throughput contexts (Manapragada et al., 2018). Random forests and gradient boosted trees have relatively high computational demands in inference, and thus may not be suitable for wearable or embedded IOT (Internet of Things) applications. So, the problem of making a single tree perform well in inference arises, and one can ask does a single decision tree beat a random forest with 10 trees. Moreover, trees also serve as one of the few global models considered

to be interpretable, an increasingly important requirement in applications (Murdoch et al., 2019). Thus, quality single decision tree built efficiently have many uses.

The Probability Estimation Tree (PET) is a generalisation of a single decision tree by taking the observed relative frequencies at a leaf node as the class probability estimates for any test examples that fall into this leaf. However, this method may lead to unreliable estimates when the number of training examples associated in a leaf is small (Zadrozny and Elkan, 2001a).

Simple probability smoothing techniques, such as Laplace smoothing and M-estimation, have long been used to improve PETs' class probability estimates by making the estimates at the leaves less extreme. However, they ignore the broader context of any leaf node, especially crucial in cases where the datasets are imbalanced.

Hierarchical smoothing has gained attention in the community in recent years. It assumes that the class probability of the leaf node depends on the probabilities of its parents in some hierarchy. To our knowledge, M-branch smoothing (Ferri et al., 2003) is the first and only one hierarchical method for PETs. It smooths the leaf node to its direct parent using M-estimation, with the parent also been smoothed recursively until the root node reached. The results demonstrate that M-branch performs better than M-estimation. Hierarchical Dirichlet Process (HDP) (Petitjean et al., 2018) can also be used to smooth the probability at the leaves with its parent, partially mimicking what is done in M-branch, but it uses fully Bayesian inference. A decision tree can be turned into an HDP model tree with each node in the tree associated with a Dirichlet Process (DP). Similar HDP smoothing methods allow Bayesian network classifiers (Petitjean et al., 2018) and language models (Shareghi et al., 2017a) to get state-of-the-art probability estimates, but this has not been applied to decision trees.

Provost and Domingos (2003) believe that a thorough study of what are the best smoothing methods for decision trees would be a successful contribution to machine learning research, which is also the main aim of this research. In this chapter, we first apply the advanced HDP smoothing method of (Petitjean et al., 2018) to decision trees and then propose a novel hierarchical smoothing approach called Hierarchical Gradient Smoothing (HGS) as an alternative. HGS smooths leaf nodes up to all the ancestors, instead of recursively smoothing to the parent used by HDP. HGS is made

faster by efficiently optimising the Leave-One-Out Cross-Validation (LOOCV) loss measure using gradient descent, instead of sampling used in HDP. An extensive set of experiments are conducted on 143 datasets showing that our HGS estimates are not only more accurate but also do so within a fraction of HDP time. Besides, HGS makes a single tree almost as good as a Random Forest with 10 trees. For applications that require more interpretability and efficiency, a single decision tree plus HGS is more preferred.

5.2 HGS Algorithm

Like all other hierarchical smoothing methods, HGS considers that the class probability estimate of a leaf node is related to the probability estimates of all parent nodes on the branch that contains the leaf. Each parent node has a weight parameter to control the degree to which the probability estimates are backed off to the parent. However, unlike HDP and M-branch smoothing, HGS smooths the leaf node to all the ancestor nodes at one time, rather than smoothing to the direct parent node recursively. This makes HGS faster than HDP and M-branch and also allows global optimisation of hyper-parameters.

HGS is different from HDP and M-branch. **Figure 17** can more intuitively express the differences between them. It can be seen from this figure that HDP and M-branch both smooth the leaf node (node 4) to an upper parent node (node 2), then the parent node (node 2) also needs to be smoothed to a higher node (node 1) until the root node is reached. Each node has a concentration parameter to control the smoothness, which is c for HDP and m for M-branch. However, unlike HDP and M-branch smoothing, HGS smooths the class probability estimate on a leaf node (node 7) to all ancestor nodes on the branch (node 3 and node 1) at one time, instead of only to the nearest parent node recursively. Each parent node has a weight parameter α to control the degree to which the probability estimates are backed off to the parent. The one-time smoothing makes HGS faster than HDP and M-branch and also allows global optimisation of hyper-parameters.

The probability smoothing formula for leaf l and class k using HGS is as follows,

$$\hat{\theta}_{l,k}^{HGS} = \frac{n_{l,k} + \sum_{p \in \text{anc}(l)} \alpha_p \hat{\theta}_{p,k}}{n_{l,\cdot} + \sum_{p \in \text{anc}(l)} \alpha_p}, \tag{45}$$

where $n_{l,k}$ is the data count for class k at node l and $n_{l,\cdot}$ is the total count.

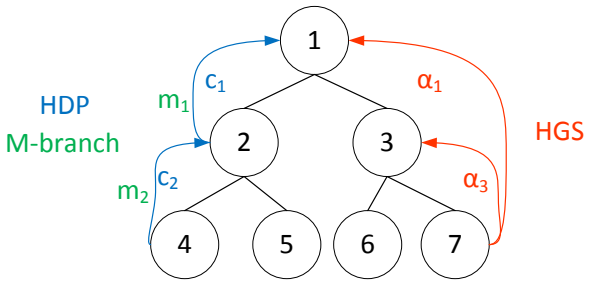

Figure 17: The difference between HGS, HDP, and M-branch. HGS is represented by red, HDP and M-branch by blue and green, respectively.

anc(l) represents all the parent nodes on the branch containing the leaf node l, which are called ancestors of l. α_p is the weight parameter for ancestor node p and the probability estimate of p is $\hat{\theta}_{p,k} = \frac{n_{p,k}}{n_{p,\cdot}}$. The term $\sum_{p \in \text{anc}(l)} \alpha_p \hat{\theta}_{p,k}$ in the numerator is the weighted combined probability of all the ancestors. The term $\sum_{p \in \text{anc}(l)} \alpha_p$ in the denominator is their sum.

5.2.1 Working with LOOCV

Let α denote the vector that contains all the weight parameter αs with the size being the number of internal nodes in the tree. How can one set the α properly, for instance, how should it be optimised?

Before going into the details of how to set α, it is worth briefly introducing LOOCV and incremental LOOCV. LOOCV is a special case of k folds cross-validation, where k equals to the number of training examples N. In each fold of the cross-validation, an example will be treated as a test example while the others are the training examples. LOOCV could be sped up using incremental LOOCV (Kohavi, 1995). The idea is instead of training a model during each fold of the cross-validation, first, train a model on the full dataset, then delete the one example that is left out, test on that example, then insert it into the model again. This delete-test-insert phase is repeated for each of the N folds. Incremental LOOCV can be conducted on any algorithm that supports incremental learning, allowing for dynamically adding or removing examples from the model. Decision tree learning is such an algorithm. Note incremental LOOCV means the model structure remains unchanged.

If one looks at a cross-validation in LOOCV, a test example at leaf l with true class k should be left out from the tree, which means the data count of both l and anc(l) should be reduced by 1. The total count becomes

$n_{l,\cdot} = n_{l,\cdot} - 1$. This Leave-One-Out (LOO) probability estimate for this test with class k becomes

$$\theta_{l,k}^{LOO} = \frac{n_{l,k} - 1 + \sum_{p \in \text{anc}(l)} \alpha_p \theta_{p,k}^{LOO}}{n_{l,\cdot} - 1 + \sum_{p \in \text{anc}(l)} \alpha_p}. \tag{46}$$

Here $\theta_{p,k}^{LOO} = \frac{n_{p,k}-1}{n_{p,\cdot}-1}$. $n_{l,k} \geq 1$ must be satisfied so that there is at least one example to moved out. For other classes $c \in \mathcal{K}, c \neq k$, the probability estimate is formed without subtracting one in the numerator.

If one performs an incremental LOOCV on the tree, the examples in every leaf $l \in \mathcal{L}$ with every class $k \in \mathcal{K}$ need to be left out once. The LOOCV measure using log loss of all the examples in the tree becomes

$$LOOCV(\alpha) = \frac{1}{N} \sum_{l \in \mathcal{L}} \sum_{k \in \mathcal{K}} n_{l,k} \cdot \log \left(\frac{1}{\theta_{l,k}^{LOO}} \right). \tag{47}$$

and note we have also tested a squared error loss $(1 - \theta_{l,k}^{LOO})^2$ yielding similar results. For more information about loss functions please refer to (Shen, 2005).

5.2.2 Parameter Learning for HGS

Now a Gradient Descent algorithm can be performed on the $LOOCV(\alpha)$ cost to optimize the parameters α. The gradient of each α_p is

$$\frac{\partial}{\partial \alpha_p} LOOCV(\alpha) = \frac{1}{N \ln 2} \sum_{l \in \text{des}(p)} \sum_{k \in \mathcal{K}} \beta_{l,k}, \tag{48}$$

where $\beta_{l,k}$ is referred to

$$\beta_{l,k} = \frac{n_{l,k} \cdot \left(\theta_{l,k}^{LOO} - \theta_{p,k}^{LOO} \right)}{\left(n_{l,\cdot} - 1 + \sum_{p \in \text{anc}(l)} \alpha_p \right) \cdot \theta_{l,k}^{LOO}}. \tag{49}$$

$\text{des}(p)$ represents the descendent leaves under p.

5.2.3 Algorithm Description

The HGS algorithm $HGS(\mathcal{T}, b, v)$ (**Algorithm 16**) takes a decision tree T, a learning rate b and a precision parameter ϵ as inputs, and a HGS smoothed

tree as output. It has three steps in total. First, initialise α to be the vector of parameters with the length to be the number of internal nodes and all the values to be 1. Second, conduct a standard gradient descent algorithm to get the optimised parameters α. Last, traverse the tree top-down in level-order to calculate the HGS smoothed probability estimates $\hat{\theta}_{l,k}^{HGS}$ for all the leaves. The top-down traverse method used here is the same as the first tree top-down traverse method in **Algorithm 18** (line 1–13), except that the probability estimates are calculated using **Equation 45** in line 9.

Algorithm 16: $HGS(\mathcal{T}, b, v)$

Input : a tree $\mathcal{T}$ with a set of leaves $\mathcal{L}$
Input : a learning rate b
Input : a precision parameter ϵ
Output : a smoothed tree $\mathcal{T}^s$

1 Initialize α to be a vector with all values to be 1;
2 $\alpha_{best} \leftarrow gradientDescent(\mathcal{T}, \alpha, b, \epsilon)$;
3 $\mathcal{T}^s \leftarrow traverseTreeTopDown(\mathcal{T}, \alpha_{best})$;
4 **return** $\mathcal{T}^s$

Algorithm 17: $gradientDescent(\mathcal{T}, \alpha, b, \epsilon)$

Input : a decision tree $\mathcal{T}$, an initialized vector α, a learning rate b, a cost difference threshold ϵ
Output : an optimized vector α

1 $cost \leftarrow calculateGradientsAndCost(\mathcal{T}, \alpha)$;
2 $costDiff = Double.max$;
3 **while** $costDiff > \epsilon$ **do**
4 **for** *each node* $p \in \mathcal{T}$ *and* $p \notin \mathcal{L}$ **do**
5 `// internal nodes`
6 $\alpha_p := \alpha_p - b\frac{\partial}{\partial \alpha_p} LOOCV(\alpha)$;
7 **end**
8 $cost^{'} \leftarrow calculateGradientsAndCost(\mathcal{T}, \alpha)$;
9 $costDiff \leftarrow cost - cost^{'}$;
10 $cost \leftarrow cost^{'}$;
11 **end**
12 **return** α.

The second step in the HGS algorithm uses a standard gradient descent algorithm to optimise the parameters, as shown in **Algorithm 17**. Gradient descent needs many iterations to reduce the cost until the cost difference

between two iterations is less than a given ϵ. **Algorithm 18** is called in every iteration to calculate the cost and gradients by going through the tree twice. First, traverse the tree top-down to calculate the LOO estimate $\theta_{p,k}^{LOO}$ using **Equation 46** and the cost $LOOCV(\alpha)$ using **Equation 47** (line 1–13). $\alpha_p^* = \sum_i \alpha_i$ and $\theta_{p,k}^* = \sum_i \alpha_i \theta_{i,k}^{LOO}$. Second, traverse the tree bottom-up to calculate the gradients for each internal node level by level (line 14–24). Last, return the cost.

Algorithm 18: *calculateGradientsAndCost*$(\mathcal{T}, \alpha)$

Input : a tree $\mathcal{T}$ with depth d and leaves $\mathcal{L}$
Input : a vector α
Output : *cost*

1 `/* Traverse 1: Calculate cost top-down. */`
2 *cost* $\leftarrow 0$;
3 **for** $h \leftarrow 0$ *to* d **do**
4 **for** *each node* p *in level* h *and each class* $k \in \mathcal{K}$ **do**
5 **if** $p \notin \mathcal{L}$ **then**
6 calculate $\theta_{p,k}^{LOO}$, $\alpha_p \theta_{p,k}^{LOO}$, α_p^* and $\theta_{p,k}^*$;
7 **else**
8 calculate $\theta_{p,k}^{LOO}$ using **Equation 46**;
9 *cost* $\leftarrow$ *cost* $+ n_{p,k} \cdot \log \frac{1}{\theta_{p,k}^{LOO}}$;
10 **end**
11 **end**
12 **end**
13 `/* Traverse 2: update gradients bottom-up. */`
14 **for** $h \leftarrow d$ *to* 0 **do**
15 **for** *each node* p *in level* h *and each class* $k \in \mathcal{K}$ **do**
16 **if** $p \notin \mathcal{L}$ **then**
17 Calculate $\frac{\partial}{\partial \alpha_p} LOOCV(\alpha)$ using **Equation 48**;
18 **else**
19 Calculate $\beta_{p,k}$ using **Equation 49**;
20 **end**
21 **end**
22 **end**
23 **return** *cost* $\leftarrow \frac{cost}{N \cdot In2}$

The complexity of HGS smoothing on a decision tree is $O(I \cdot S \cdot K)$, where S is the total number of nodes and K is the number of classes. We call I the number of iterations for gradient descent. In practice, we use the standard stopping criterion corresponding to an improvement of less than ϵ. Each

iteration of gradient descent has a complexity that is linear to the size of the tree S (total number of nodes).

5.3 Experiment Results

The aim of this section is to show the performance of HGS smoothing compared with other existing smoothing methods for decision trees. **Section 5.3.1** gives the general experimental settings. The remaining sections then detail individual experiments.

5.3.1 Experiment Design and Setting

Design: An extensive set of experiments are conducted on 143 standard datasets from the UCI archive (Lichman, 2013), where 20 datasets have more than 10,000 instances, 52 datasets have between 1,000 and 10,000, and 71 datasets have less than 1,000 instances. **Table 13** summarises the characteristics of each dataset, including the name, number of instances and attributes. Numerical datasets do not need to be discretised. A missing value is treated as a separate attribute value. The datasets can be found and downloaded from Github [1].

Evaluation Measure: The results are assessed by RMSE and error rate, among which RMSE is the most important measure because it measures how well-calibrated the probability estimates are. RMSE is the square root of the Brier score in discrete contexts, which is used to measure how well-calibrated the probability estimates are. The error rate is not our focus compared to RMSE. For both RMSE and error rate, the smaller, the better. Win-Draw-Loss (WDL) is reported when comparing the RMSE and error rate of two models. A one-tail binomial sign test is used to determine the significance of the results. A difference is considered to be significant if $p \leq 0.05$.

Software: To ensure reproducibility of our work and allow other researchers to build on our research easily, we have made our source code for HGS smoothing on decision trees available on Github [2].

Compared models and parameters: In our experiment, different probability smoothing methods are compared for C4.5 decision trees without pruning, including Laplace smoothing, M-estimation, M-branch, HDP and HGS. Here

[1] https://github.com/icesky0125/dataset-and-raw-results-for-HGS-paper
[2] https://github.com/icesky0125/DecisionTreeSmoothing

M-estimation is used with a back-off strategy, which means when the leaf node has no data, back off the probability estimates to its nearest non-empty ancestor. The validation method is also included with 50% of the data to learn the tree structure and the remaining to learn the parameters. All the methods are conducted using 10-fold cross-validation.

Table 13: Datasets for decision tree smoothing.

Domain	Case	Att	Domain	Case	Att
skin-segmentation	245,057	3	vowel	990	13
localization	164,860	7	mammographic	961	5
ipums	88,443	60	tic-tac-toe	958	9
OneBig	68,000	20	annealing	898	38
connect-4	67,557	42	vehicle	846	18
shuttle	58,000	9	energy	768	8
adult	48,842	14	pid	768	8
tamilnadu-electri	45,781	3	blood	748	4
bank-marketing	45,211	16	breast-cancer-w	699	9
nomao	34,465	118	credit-approval	690	15
kr-vs-k	28,056	6	balance-scale	625	4
letter-recog	20,000	16	syscon	600	60
magic	19,020	10	indianLiverPatient	583	10
eeg-eye-state	14,980	14	wdbc	569	30
gas-drift	13,910	128	chess	551	39
nursery	12,960	8	climateSimulation	540	20
sign	12,546	8	cylinder-bands	540	39
pendigits	10,992	16	spectrometer	531	101
HumanActivity	10,299	561	meta-data	528	21
artificial-character	10,218	7	dresses-sales	500	12
thyroid	9,169	29	thoracic-surgery	470	16
pioneer	9,150	36	saheart	462	9
mushroom	8,124	22	arrhythmia	452	262
ringnorm	7,400	20	wholesaleChannel	440	7
twonorm	7,400	20	wholesaleRegion	440	7
musk	6,598	167	house-votes-84	435	16
satellite	6,435	36	user-knowledge	403	5

continued ...

. . . continued

Domain	Case	Att	Domain	Case	Att
first-order-theorem	6,118	51	horse-colic	368	21
turkiye-student	5,820	32	dermatology	366	34
optdigits	5,620	64	movement-libras	360	90
texture	5,500	40	ionosphere	351	34
page-blocks	5,473	10	spectf-heart	349	44
wall-following	5,456	24	bupa	345	6
phoneme	5,438	7	leaf	340	15
banana	5,300	2	primaryTumor	339	17
waveform-5000	5,000	40	ecoli	336	7
wineQualityWhite	4,898	11	soybean-large	307	35
wilt	4,839	5	habermans	306	3
spambase	4,601	57	cleveland	303	13
abalone	4,177	8	hungarian	294	13
sick	3,772	29	statlog-heart	270	13
kr-vs-kp	3,196	36	spect-heart	267	22
splice	3,190	60	bankruptcy	250	6
splice-c4	3,177	60	Audiology	226	69
hypothyroid	3,163	25	new-thyroid	215	5
madelon	2,600	500	glass	214	9
seismic-bumps	2,584	18	seeds	210	7
ozone-onehr	2,536	72	sonar	208	60
image-segment	2,310	19	autos	205	25
cardiotocography	2,126	35	wpbc	198	33
mfeat-fourier	2,000	76	parkinsons	195	22
mfeat-mor	2,000	6	flags-colour	194	29
multiple-features	2,000	649	flags-religion	194	29
steel-plates-faults	1,941	33	planning-relax	182	12
car-evaluation	1,728	6	robotFailureLp5	164	90
leaves-margin	1,600	64	hayes-roth	160	4
leaves-shape	1,600	64	hepatitis	155	19
leaves-texture	1,599	64	teaching-assistant	151	5
wine-quality-red	1,599	11	iris	150	4
semeion	1,593	256	lymphography	148	18

continued . . .

. . . continued

Domain	Case	Att	Domain	Case	Att
volcanoes	1,520	3	echocardio	131	6
amazon	1,500	10000	voice	126	310
yeast	1,484	8	inflammations	120	6
cmc	1,473	9	robot-failure-lp4	117	90
flare	1,389	12	appendicitis	106	7
banknote	1,372	4	breast-tissue	106	9
hill-valley	1,212	100	promotor	106	57
cnae-9	1,080	856	zoo	101	16
qsar-bio	1,055	41	blogger	100	5
german	1,000	20	fertility-diagnosis	100	9
led	1,000	7	leukemia-haslinger	100	50
german-credit	1,000	24			

5.3.2 HDP Parameter Tuning

This section tunes the parameters of HDP on decision trees. As we have discussed in **Section 2.3.3**, the two most critical parameters of HDP smoothing for tree-structured classifiers in (Petitjean et al., 2018) are *iteration* and *tying*. *iteration* is the cycles that Gibbs sampling needs to sample the parameters. The sampling process can be accelerated by tying some nodes together to share the same parameter value. *tying* is the parameter used to control which nodes are tied together.

Table 14 is the averaged results on 143 datasets using the four strategies with $iteration = 1000$. The error bars of these four strategies are all the same, which are 0.012 on RMSE and 0.015 on error rate respectively. As can be seen from this table that $SINGLE$ and $LEVEL$ give very similar results. The variance is relatively low when the number of parameters is small.

Table 15 tests the *iteration* parameter with the *tying* fixed to $SINGLE$. 1,000 iterations achieves better average RMSE results and is about 3.5 times faster than 5,000 iterations. In the following experiments, we set HDP using $iteration = 1,000$ and $tying = SINGLE$.

Table 14: Averaged results of the four tying strategies.

tying	RMSE	Error Rate	Runtime (s)
SINGLE	0.2436	0.2078	4.9
LEVEL	0.2493	0.2087	4.7
PARENT	0.2525	0.2090	5.2
NONE	0.2533	0.2091	6.7

Table 15: Averaged results of the two iterations.

iteration	RMSE	Error Rate	Runtime (s)
1000	0.2436	0.2078	4.9
5000	0.2450	0.2074	17.6

5.3.3 HGS Parameter Tuning

HGS is basically a parameterless algorithm. The only two parameters are the learning rate b and the minimum cost difference threshold ϵ between two iterations needed in the gradient descent algorithm (Ruder, 2016). The learning rate controls how big the steps of the gradient descent. With a high learning rate, we can cover more ground each step, but we risk overshooting the lowest point since the slope of the hill is constantly changing. A lower learning rate is more precise, but calculating the gradient is time-consuming, so it takes a long time to get to the bottom. In this experiment, we tried different values $b = 0.01, 0.001$ and $\epsilon = 0.001, 0.0001$ and found that their results were all the same with only slight differences in training time. The RMSE is 0.2410, and the error rate is 0.2059. Based on these results, in the following experiment we choose the standard values of $b = 0.01$ and $\epsilon = 0.0001$.

5.3.4 HGS vs. Single Layer Smoothing Methods

This part of the experiment is to evaluate the advantages of HGS smoothing over other single-layer smoothing methods (including MLE, Laplace smoothing, and M-estimation), whose class probability estimation depends only on leaf nodes.

Table 16 shows the Win-Draw-Loss between HGS and MLE, Laplace and M-estimation, from which it can be seen that in 143 datasets, HGS is superior to all other methods in error rate, and is significantly better than other methods in RMSE. Meanwhile, it can be seen that M-estimation

gets better RMSE than MLE and Laplace, but the classification accuracy measured by the error rate is not as good as them.

Table 16: WDL of HGS compared with MLE, Laplace and M-estimation (Stat. sig. $p < 0.05$ results are depicted in boldface).

Method	RMSE	Error Rate
HGS vs. MLE	**108-2-33**	69-22-52
HGS vs. Laplace	**111-4-28**	68-22-53
HGS vs. M-estimation	**98-4-41**	66-23-54

Table 17 lists the averaged results on 143 datasets for HGS, MLE, Laplace and M-estimation. The error bars of all these models are 0.012 on RMSE and 0.015 on error rate, respectively. The standard deviation of them on RMSE are 0.150, 0.133, 0.142 and 0.132, respectively. It can be seen from this table that HGS achieved the best RMSE and error rate, which means HGS makes decision trees get both very accurate probability estimates and classification accuracy. It can also be seen that even HGS is a hierarchical smoothing method, but it has the same learning time as the single-layer methods, indicating the high efficiency of HGS.

Table 17: Averaged results of all the methods.

Methods	RMSE	Error Rate	Runtime (s)
MLE	0.2596	0.2093	1.1
Laplace	0.2499	0.2093	1.1
M-estimation	0.2485	0.2068	1.1
HGS	**0.2410**	**0.2059**	**1.1**

To show the comparison between HGS and other methods more intuitively, we present the scatter plots of these methods on both RMSE and error rate. The circles on the diagonal indicate that HGS is superior on these datasets. As can be seen from **Figure 18**, HGS works better than MLE for data sets with higher RMSE, and MLE works better for data sets with smaller RMSE. This can be explained by the fact that generally, larger data sets tend to have smaller RMSE values. Simple MLE methods can yield better results when the data volume is large enough, but for small datasets, probability smoothing can effectively improve probability estimates. **Figure 19** and **Figure 20** are the scatter plots of HGS compared with Laplace and M-estimation, respectively.

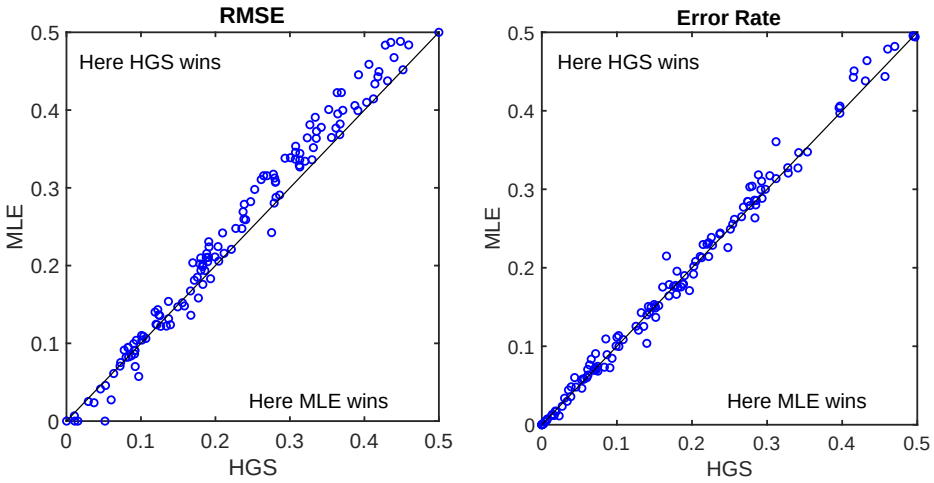

Figure 18: HGS vs. MLE.

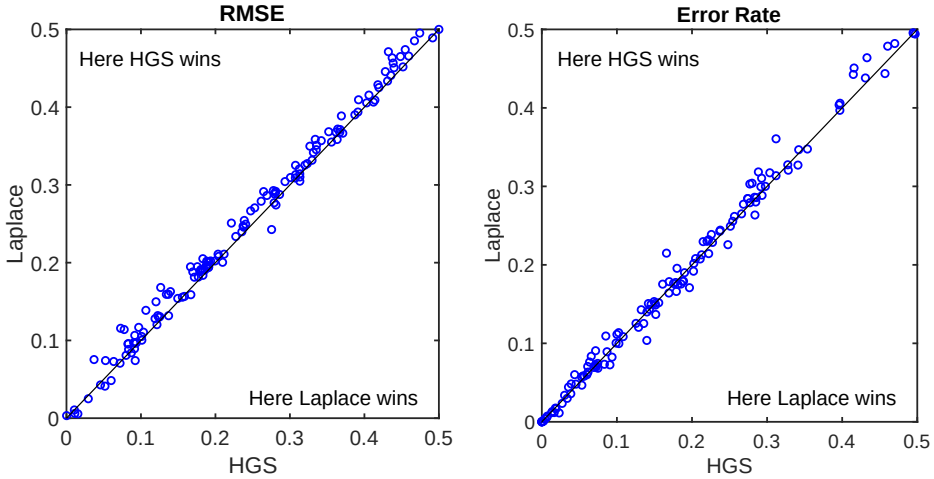

Figure 19: HGS vs. Laplace.

5.3.5 HGS vs. Hierarchical Smoothing Methods

This part of the experiment is to evaluate the advantages of HGS smoothing for decision trees compared with other hierarchical probability smoothing methods, including HDP and M-branch.

Table 18 shows the Win-Draw-Loss between HGS and M-branch and HDP. It can be seen from this table that HGS is significantly better than HDP and M-branch in RMSE. HGS makes more than 90 out of 143 datasets

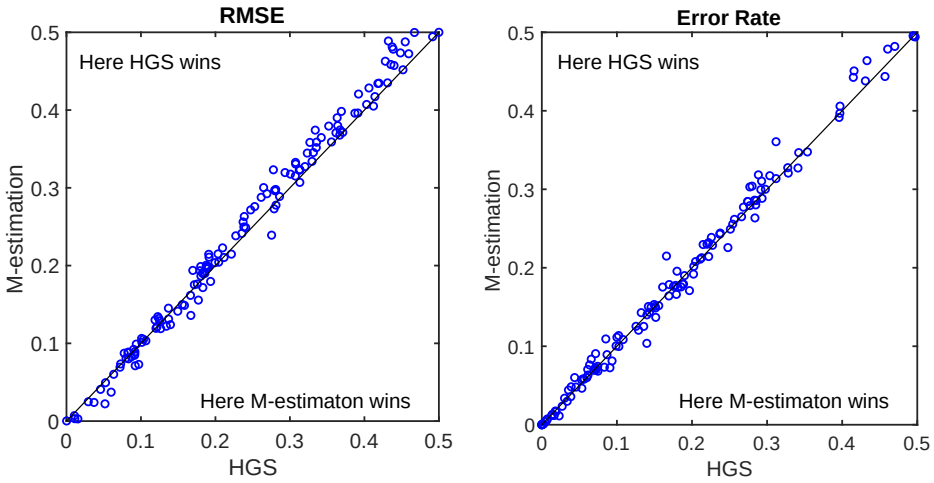

Figure 20: HGS vs. M-estimation.

better than them.

Table 18: WDL of HGS compared with M-branch and HDP (Stat. sig. $p < 0.05$ results are depicted in boldface).

Method	RMSE	Error Rate
HGS vs. M-branch	**96-3-44**	59-32-52
HGS vs. HDP	**92-1-50**	64-21-58

Table 19 shows the averaged RMSE and error rate of HGS, M-branch and HDP on 143 datasets. HGS obtained the best performance but was 5 times faster than M-branch and 9 times faster than HDP. Although Petitjean et al. (2018) suggested that the HDP estimate might be more accurate if more samples were taken, it would also mean longer times.

Table 19: Averaged results of HGS and the hierarchical smoothing methods.

Methods	RMSE	Error Rate	Runtime (s)
HDP	0.2436	0.2078	4.9
M-branch	0.2428	0.2062	9.3
HGS	**0.2410**	**0.2059**	**1.1**

Figure 21 and **Figure 22** are the scatter plots of HGS versus HDP and M-branch smoothing on RMSE and error rate. These plots show that HGS gets good improvement over HDP and M-branch. **Figure 23** shows the comparison of HDP with M-branch. The result of M-branch and HDP are

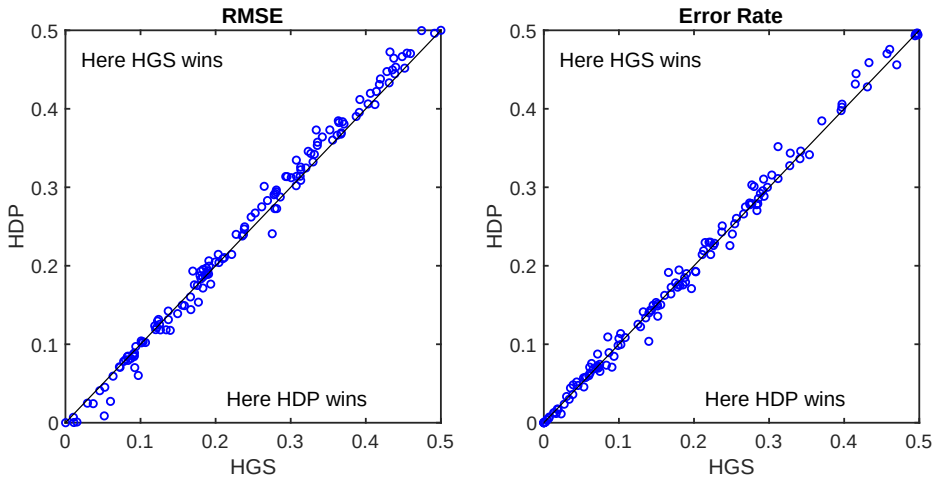

Figure 21: HGS vs. HDP.

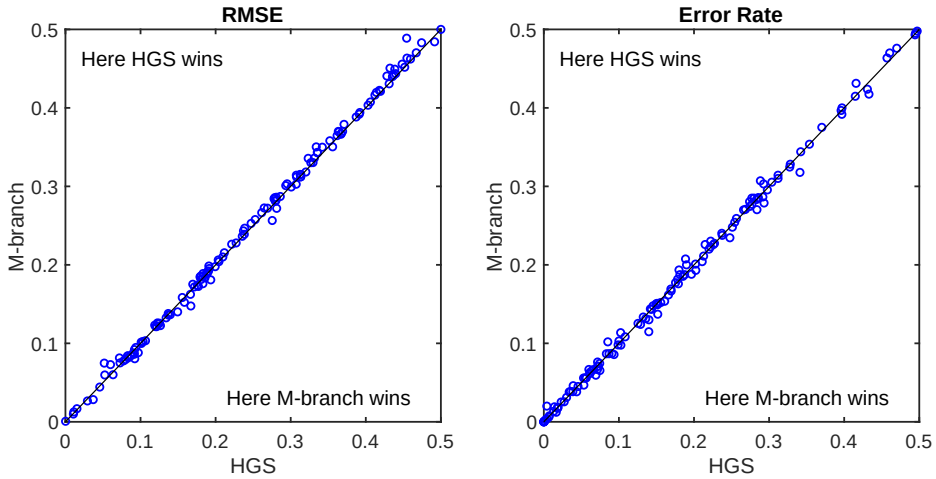

Figure 22: HGS vs. M-branch.

similar since M-branch and HDP is actually hierarchical M-estimation with each base rate to be the parent probability estimate.

To compare the three hierarchical smoothing methods more intuitively, we take the RMSE of HGS as the benchmark for each dataset and subtract RMSE of M-branch and HDP, and drew **Figure 24**. The datasets are arranged in descending order of their sizes from left to right. The points in the grey area represent the datasets that perform better with HGS. The lower the point is, the stronger the advantage of HGS is. The following conclusions can be

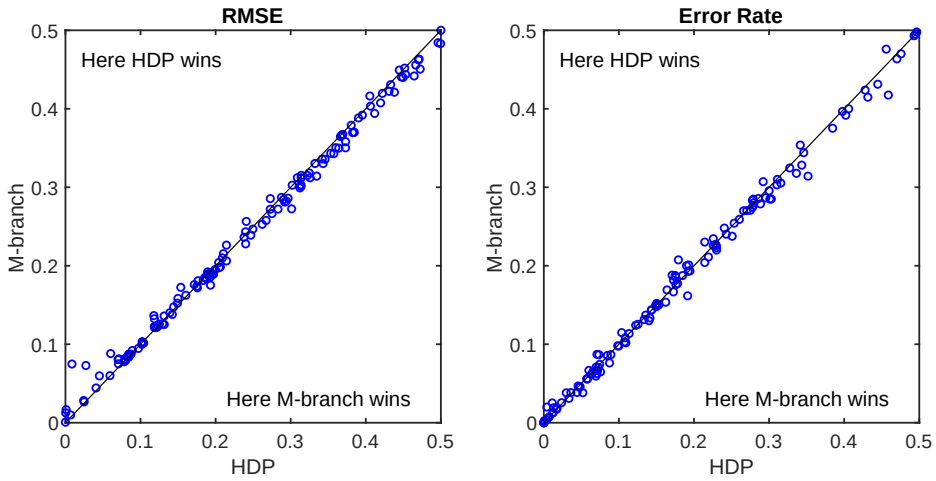

Figure 23: HDP vs. M-branch.

drawn. First, there are more points in the grey area, which indicates that HGS makes the majority of the datasets with better estimates. Second, the "stars" points are mostly centred around the line $y = 0$, while o points are more diffuse. This means HDP has high variance compared with M-branch. Last, among the top 20 largest datasets with more than 10,000 examples on the far left of the figure, HDP makes 14 out of them performs better than HGS and Mbranch. This indicates that HDP is more helpful on large datasets.

5.3.6 Smoothing vs. Data Size

Based on the intuition that large datasets tend to fit well and have smaller RMSE, people can get the following guidance on smoothing and data sizes. MLE does not work well on small datasets because of data sparsity problem (See **Figure 18**). Laplace makes small datasets better but large datasets worse (See **Figure 19**). M-estimation makes RMSE more stable no matter how big the dataset is (See **Figure 20**). HDP is particular better on large datasets but worse on small ones (See **Figure 21**). M-branch and HGS have little relation with data size (See **Figure 22**).

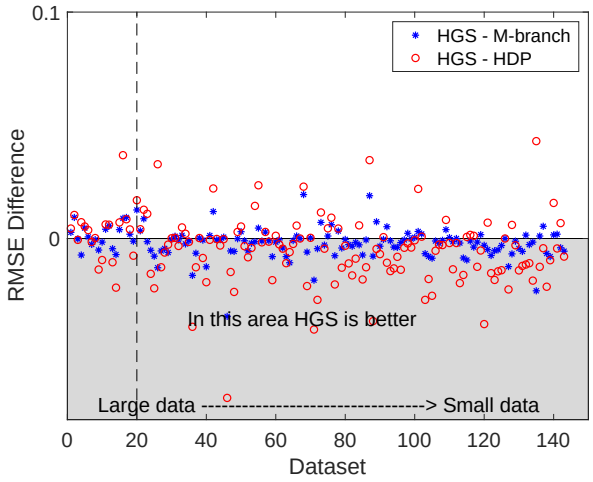

Figure 24: Compare the probability estimates of HGS with HDP and M-branch. The X-axis represents the datasets arranged from large to small in terms of data size. The Y-axis represents the RMSE difference between each method and HGS, which is the lower, the better. First, the majority of the points are below the line $y = 0$, which means HGS performs better on most of the datasets. Second, for the top 20 large datasets HDP is better than M-branch and HGS.

5.3.7 HGS vs. Validation

In this section, we compare HGS smoothing with the validation method, where 50% of the training data are used to build the decision tree model and the remaining 50% of the data are used to learn the class probability estimates at leaves using frequencies. **Table 20** shows the averaged results over 143 datasets in RMSE and error rate. As can be seen from the table, HGS smoothing performs better in the decision tree than the validation method.

Table 20: Averaged results of HGS and the validation method.

Methods	RMSE	Error Rate	Runtime (s)
Validation	0.2653	0.2353	0.7
HGS	**0.2410**	**0.2059**	**1.1**

5.3.8 Running Time vs. Data Size

One of the most important motivations of HGS is its efficiency, i.e. running time. It is interesting to investigate the running times based on different data size. **Figure 25** is the running time versus data sizes plot for HGS and HDP evaluated on all the datasets. We take the log of both the data sizes and the running times to make the figure more intuitive. It is evident that the blue $*$ are almost always lower than the red circles, which indicates that the training time of HGS on datasets of different sizes is basically shorter than that of HDP.

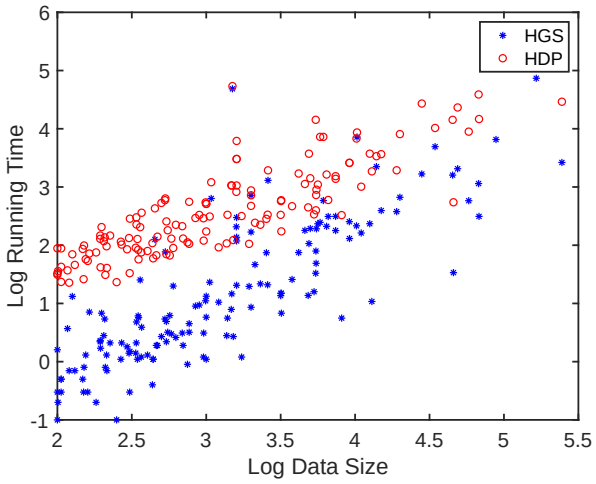

Figure 25: Training time comparison according to log data size.

5.3.9 Smoothing vs. Pruning

Many studies suggest to skip the pruning process and apply probability smoothing techniques to decision trees to obtain better probability estimates (Elkan, 2001; Han, 2011; Zadrozny and Elkan, 2001b). This part of the experiment is to verify this conclusion further. Reduced Error Pruning (REP) is used after the tree grew, then use MLE to get the probability estimates. **Table 21** is the WDL of above mentioned smoothing methods compared with pruning. First, pruning does not help on probability estimation because its aim is to improve the classification accuracy rather than the probability estimates. Pruning with MLE does not improve the estimate compared to using MLE alone. Laplace, M-estimation and HDP obtain similar error rate

versus pruning, but the RMSE has improved a lot. HGS makes trees improved significantly compared with pruning both on RMSE and error rate.

Table 21: WDL for smoothing compared with pruning (Stat. sig. $p < 0.05$ results are depicted in boldface).

vs. pruning+MLE	RMSE	Error Rate
MLE	70-4-69	**84-11-48**
Laplace	**92-1-50**	**84-11-48**
M-estimation	**92-2-49**	**84-11-48**
HDP	**93-2-48**	**84-9-50**
M-branch	**120-1-22**	**92-11-40**
HGS	**123-1-19**	**95-12-36**

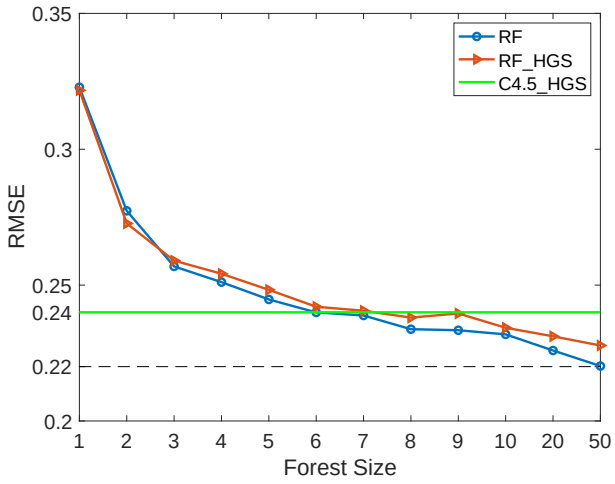

Figure 26: HGS Smoothing on Random Forest in RMSE.

5.3.10 HGS on Random Forest

We sought to determine how many trees are needed in RF to beat HGS on a single tree, and the impact of smoothing with RF. Previously, Bostrom (2007) suggested that a non-corrected probability estimate should be used in RF. **Figure 26** shows the RMSE changing with the forest size. RF_HGS represents random forest using HGS smoothing, while RF means no smoothing. C4.5 with HGS smoothing is represented by line $y = 0.24$. This figure shows that HGS makes RF worse after three trees because smoothing can reduce the diversity of RF. A single C4.5 tree using HGS yields RMSE better or

close to RF with 7 trees and comparatively for 10 trees. While single trees with HGS smoothing cannot beat RF, a single tree is preferred if one is more interested in interpretability.

5.4 Summary

In this chapter, we present the hierarchical probability smoothing method HGS for decision trees. This algorithm enables the class probability estimation of the decision tree to be more accurate and efficient than existing single-layer probabilistic smoothing methods (including MLE, Laplace, and m-estimation) and hierarchical methods (including HDP and M-branch). In addition, a single tree plus HGS smoothing performs comparably with a Random Forest with seven trees, but it can be learned and tested much faster and with higher interpretability.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

6.1.1 Conclusion for ESKDB

The ESKDB algorithm is a novel ensemble method for SKDB, where different BNCs are generated by changing the attribute orders and by sampling a discretization. Ablation experiments demonstrated that both sources of stochasticity are needed. This method produces more accurate class probability estimates than the existing BNCs both on RMSE and on 0/1 loss using M-estimation for smoothing. The HDP method of smoothing Petitjean et al. (2018) can be used as well, and unlike early Chinese restaurant implementations of HDP, this has no dynamic memory requirements so scales well. We also developed an improvement for HDP smoothing that is both far faster and more accurate, but it is still a few times slower than M-estimation.

The ensembling strategy used shows interesting departures from well known techniques like RF. First, better smoothing is used in the base classifiers, and we conjecture this is because the base SKDB is itself a combination of simpler classifiers (which are smoothed) using Bayes rule. Second, only 10 ensembles perform well, which we conjecture is due to the reduction of variance already achieved by better smoothing.

The experimental results give guidance on using smoothing and ensembling:

- Single SKDB with HDP smoothing yields more explainable models, and is comparable in performance to default versions of RFs and XGBoost.

- Due to comparative results reported in Duan and Wang (2017), this also means single SKDB with HDP smoothing beats KDF.
- For more efficient and accurate probability estimates, ensembling SKDBs with M-estimation is preferred, and can easily be run out-of-core.
- For best probability estimates, ensembling SKDBs with HDP is recommended.

The diversity needed for ensembles could be further increased by using attribute selection and data re-sampling using bagging. Also, SKDB could be modified to integrate better with ensembling or HDP smoothing, or modified to introduce the conditional mutual information as done for KDF. Thus while our best method, ESKDB with HDP smoothing did not outperform XGBoost on the larger datasets, there is considerable room for improvement, so we view the advances presented here as an important step in developing superior, high performance, scalable classification methods.

6.1.2 Conclusion for HGS

It is well known that probability smoothing beats pruning, and M-branch showed us that hierarchical smoothing can improve again. This thesis, however, develops a new hierarchical algorithm, HGS, and tests out a recent algorithm, HDP smoothing on trees for the first time. The originality of HGS is in removing recursive smoothing and efficient pre-computation of key statistics, which also allow better optimization of hyper-parameters. This experimental evaluation demonstrates three significant contributions.

1. HDP smoothing developed in Petitjean et al. (2018) is shown to be comparable to M-branch, and evidence suggests it is the superior algorithm for large data sets.
2. HGS is an order of magnitude faster than M-branch and HDP smoothing, and significantly better in RMSE.
3. HGS is generally as good as or superior to a random forest with 7 trees and almost as good with 10 trees. This makes HGS a single tree alternative to a random forest with 10 trees or less, and thus suitable in online contexts Manapragada et al. (2018).

6.2 Future Work

6.2.1 Apply HGS smoothing to ESKDB

HDP smoothing has been shown to get the most accurate parameter estimates for Bayesian network classifiers, but it is computationally intensive. Besides, HDP works less effective than HGS on decision trees. Therefore, it would be of interest to apply HGS smoothing to the ESKDB model to see whether HGS could make Bayesian network classifiers get better parameter estimates. Although both decision trees and Bayesian network classifiers can be represented by tree structures, their mechanisms are different. Split attributes of decision trees are more diverse, while each layer of a conditional probability tree of Bayesian network classifier splits on the same parent attribute. This may result in a difference in the application of the HGS algorithm to the Bayesian network classifiers.

6.2.2 Cost-sensitive Learning Decision Trees

Cost-sensitive learning techniques believe a different misclassification error has a different cost. In disease diagnosis, for example, if we misdiagnose a healthy person as a patient, the cost may be some treatment. Whereas, if we misdiagnose a patient as a healthy person, the cost is much higher, it may be the patient's life. Accurate class probability estimation is the key to reduce the misclassification cost in cost-sensitive learning. Probability smoothing techniques, such as Laplace smoothing and M-estimation, have long been used for cost-sensitive decision trees. Now that we have a more efficient probability estimation method for decision trees, the HGS algorithm, applying HGS algorithm to cost-sensitive decision trees will be significant progress in the cost-sensitive learning field.

References

Jacinto Arias, José A Gámez, and José M Puerta. Bayesian network classifiers under the ensemble perspective. In *International Conference on Probabilistic Graphical Models*, pages 1–12, 2018.

Anthony G Barnston. Correspondence among the correlation, rmse, and heidke forecast verification measures; refinement of the heidke score. *Weather and Forecasting*, 7(4):699–709, 1992.

Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1-2): 105–139, 1999.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

Henrik Bostrom. Estimating class probabilities in random forests. In *Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on*, pages 211–216. IEEE, 2007.

Henrik Boström. Calibrating random forests. In *Machine Learning and Applications, 2008. ICMLA'08. Seventh International Conference on*, pages 121–126. IEEE, 2008.

Henrik Boström. Forests of probability estimation trees. *International journal of pattern recognition and artificial intelligence*, 26(02):1251001, 2012.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Glenn W Brier. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3, 1950.

Wray L Buntine and Swapnil Mishra. Experiments with non-parametric topic models. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 881–890. ACM, 2014.

John Burge and Terran Lane. Shrinkage estimator for Bayesian network parameters. In *European Conference on Machine Learning*, pages 67–78. Springer, 2007.

Nitesh V Chawla. Many are better than one: improving probabilistic estimates from decision trees. In *Machine Learning Challenges Workshop*, pages 41–55. Springer, 2005.

Nitesh V Chawla. Many are better than one: Improving probabilistic estimates from decision trees. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 41–55. Springer, 2006.

Nitesh V Chawla and David A Cieslak. Evaluating probability estimates from decision trees. In *American Association for Artificial Intelligence*, 2006.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SigKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

Ye Chen and John Canny. Probabilistic recommendation of an item, September 12 2017. US Patent 9,760,802.

Vincy Cherian and MS Bindu. Heart disease prediction using naive Bayes algorithm and Laplace smoothing technique. *International Journal of Computer Science Trends and Technology (IJCST)*, 5(2), 2017.

C Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3): 462–467, 1968.

Michael Crawford, Taghi M Khoshgoftaar, Joseph D Prusa, Aaron N Richter, and Hamzah Al Najada. Survey of review spam detection using machine learning techniques. *Journal of Big Data*, 2(1):23, 2015.

Rajashree Dash, Rajib Lochan Paramguru, and Rasmita Dash. Comparative analysis of supervised and unsupervised discretization techniques.

International Journal of Advances in Science and Technology, 2(3):29–37, 2011.

Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134 (1):19–67, 2005.

James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *Machine Learning Proceedings 1995*, pages 194–202. Elsevier, 1995.

Zhiyi Duan and Limin Wang. K-dependence Bayesian classifier ensemble. *Entropy*, 19(12):651, 2017.

Charles Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, volume 17, pages 973–978. Lawrence Erlbaum Associates Ltd, 2001.

Bradley J Erickson, Panagiotis Korfiatis, Zeynettin Akkus, and Timothy L Kline. Machine learning for medical imaging. *Radiographics*, 37(2):505–515, 2017.

U Fayyad and K Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *13th International Joint Conference on Artificial Intelligence*, volume 2, pages 1022–1027, 1993.

Usama M. Fayyad and Keki B. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8(1):87–102, January 1992.

C Ferri, P Flach, and J Hernández-Orallo. Decision trees for ranking: effect of new smoothing methods, new splitting criteria and simple pruning methods. *Technical report, DSIC 2003*, 2003.

S Fortmann-Roe. Understanding the bias-variance trade-off.[online] available: http://scott. fortmann-roe. com/docs. *BiasVariance. html [2018, May 22]*, 2012.

Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

Stephen R Garner et al. Weka: The waikato environment for knowledge analysis. In *Proceedings of the New Zealand computer science research students conference*, pages 57–64, 1995.

Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

Robin Genuer. Variance reduction in purely random forests. *Journal of Nonparametric Statistics*, 24(3):543–562, 2012.

Pierre Geurts. Bias vs variance decomposition for regression and classification. In *Data mining and knowledge discovery handbook*, pages 733–746. Springer, 2009.

Gerd Gigerenzer, Ralph Hertwig, Eva Van Den Broek, Barbara Fasolo, and Konstantinos V Katsikopoulos. "a 30% chance of rain tomorrow": How does the public understand probabilistic weather forecasts? *Risk Analysis: An International Journal*, 25(3):623–629, 2005.

Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477):359–378, 2007.

Zhimeng Han. *Smoothing in Probability Estimation Trees*. PhD thesis, University of Waikato, 2011.

Marti A. Hearst. Support vector machines. *IEEE Intelligent Systems*, 13(4): 18–28, July 1998.

Jennifer A. Hoeting, David Madigan, Adrian E. Raftery, and Chris T. Volinsky. Bayesian model averaging: a tutorial (with comments by M. Clyde, David Draper and E. I. George, and a rejoinder by the authors). *Statistical Science*, 14(4):382–417, 11 1999.

Marjan Jahanshahi, Leonora Wilkinson, Harpreet Gahir, Angeline Dharminda, and David A Lagnado. Medication impairs probabilistic classification learning in parkinson's disease. *Neuropsychologia*, 48(4):1096–1103, 2010.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

Liangxiao Jiang, Dianhong Wang, and Zhihua Cai. Scaling up the accuracy of Bayesian network classifiers by m-estimate. In *International Conference on Intelligent Computing*, pages 475–484. Springer, 2007.

Xiaoqian Jiang, Melanie Osl, Jihoon Kim, and Lucila Ohno-Machado. Smooth isotonic regression: A new method to calibrate predictive models. *AMIA Summits on Translational Science Proceedings*, 2011:16, 2011.

Pooria Joulani, Andras Gyorgy, and Csaba Szepesvári. Fast cross-validation for incremental learning. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Szabolcs Kéri, Csaba Szlobodnyik, György Benedek, Zoltán Janka, and Júlia Gádoros. Probabilistic classification learning in tourette syndrome. *Neuropsychologia*, 40(8):1356–1362, 2002.

Ron Kohavi. The power of decision tables. In *European conference on machine learning*, pages 174–189. Springer, 1995.

Ron Kohavi and Mehran Sahami. Error-based and entropy-based discretization of continuous features. In *KDD*, pages 114–119, 1996.

Sotiris Kotsiantis and Dimitris Kanellopoulos. Discretization techniques: A recent survey. *GESTS International Transactions on Computer Science and Engineering*, 32(1):47–58, 2006.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Tim Leathart, Eibe Frank, Geoffrey Holmes, and Bernhard Pfahringer. Probability calibration trees. In *ACML 2017*, pages 145–160, 2017.

David D. Lewis. *Naive Bayes at forty: The independence assumption in information retrieval*, pages 4–15. Springer, 1998.

Pengyu Li, He Zhang, Xuyang Zhao, Cangzhi Jia, Fuyi Li, and Jiangning Song. Pippin: A random forest-based method for identifying presynaptic and postsynaptic neurotoxins. *Journal of Bioinformatics and Computational Biology*, page 2050008, 2020.

Qing Li, Sung Hyon Myaeng, Dong Hai Guan, and Byeong Man Kim. A probabilistic model for music recommendation considering audio features. In *Asia Information Retrieval Symposium*, pages 72–83. Springer, 2005.

M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

David Madigan, Jeremy York, and Denis Allard. Bayesian graphical models for discrete data. *International Statistical Review/Revue Internationale de Statistique*, pages 215–232, 1995.

Chaitanya Manapragada, Geoffrey I. Webb, and Mahsa Salehi. Extremely fast decision tree. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 1953–1962, New York, USA, 2018. ACM. ISBN 978-1-4503-5552-0.

Dragos D Margineantu. Class probability estimation and cost-sensitive classification decisions. In *European Conference on Machine Learning*, pages 270–281. Springer, 2002.

Ana M Martınez, Geoffrey I Webb, Shenglei Chen, and Nayyar A Zaidi. Scalable learning of Bayesian network classifiers. *Journal of Machine Learning Research*, 17:1–35, 2016.

Rory Mitchell and Eibe Frank. Accelerating the xgboost algorithm using gpu computing. *PeerJ Computer Science*, 3:e127, 2017.

W. James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, 2019. ISSN 0027-8424.

Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632. ACM, 2005.

Domingos Pedro. A unified bias-variance decomposition and its applications. In *17th International Conference on Machine Learning*, pages 231–238, 2000.

François Petitjean, Wray Buntine, Geoffrey I Webb, and Nayyar Zaidi. Accurate parameter estimation for Bayesian network classifiers using hierarchical Dirichlet processes. *Machine Learning*, 107(8-10):1303–1331, 2018.

John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

New York Post. *Reliable AI system predicts Trump will win*, 2016. URL https://nypost.com/2016/10/28/reliable-ai-system-predicts-trump-will-win/.

Foster Provost and Pedro Domingos. Well-trained PETs: Improving probability estimation trees. *CDER WorkingPaper, Stern School of Business. New York, NY: New York University*, 2000.

Foster Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine learning*, 52(3):199–215, 2003.

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

J Ross Quinlan. C4.5: Programs for machine learning. *The Morgan Kaufmann Series in Machine Learning, San Mateo, CA: Morgan Kaufmann,| c1993*, 1993.

Jyoti Ramteke, Samarth Shah, Darshan Godhia, and Aadil Shaikh. Election result prediction using twitter sentiment analysis. In *2016 international conference on inventive computation technologies (ICICT)*, volume 1, pages 1–5. IEEE, 2016.

Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Stefan Rüping. Robust probabilistic calibration. In *European Conference on Machine Learning*, pages 743–750. Springer, 2006.

Jarosław Rzeszótko and Sinh Hoa Nguyen. Machine learning for traffic prediction. *Fundamenta Informaticae*, 119(3-4):407–420, 2012.

Mehran Sahami. Learning limited dependence Bayesian classifiers. In *KDD*, volume 96, pages 335–338, 1996.

Dewi Retno Sari Saputro, Purnami Widyaningsih, Feri Handayani, and Nughthoh Arfawi Kurdhi. Prior and posterior Dirichlet distributions on Bayesian networks (bns). In *AIP Conference Proceedings*, volume 1827, page 020036. AIP Publishing LLC, 2017.

Claude E Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

Ehsan Shareghi, Gholamreza Haffari, and Trevor Cohn. Compressed nonparametric language modelling. In *Proc. of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 2701–2707, 2017a.

Ehsan Shareghi, Gholamreza Haffari, and Trevor Cohn. Compressed nonparametric language modelling. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2701–2707, 2017b.

Yi Shen. *Loss functions for binary classification and class probability estimation*. PhD thesis, University of Pennsylvania, 2005.

Ayush Singhal, Pradeep Sinha, and Rakesh Pant. Use of deep learning in modern recommendation system: A summary of recent works. *arXiv preprint arXiv:1712.07525*, 2017.

Yee Whye Teh. Dirichlet process. *Encyclopedia of machine learning*, pages 280–287, 2010.

Yee Whye Teh and Michael I Jordan. Hierarchical Bayesian nonparametric models with applications. *Bayesian nonparametrics*, 1, 2010.

Yong Wang, Julia Hodges, and Bo Tang. Classification of web documents using a naive bayes method. In *Proceedings. 15th IEEE International*

Conference on Tools with Artificial Intelligence, pages 560–564. IEEE, 2003.

Geoffrey I. Webb, Janice R. Boughton, and Zhihai Wang. Not so naive Bayes: Aggregating one-dependence estimators. *Machine Learning*, 58(1):5–24, January 2005.

Wikipedia contributors. Sign test — Wikipedia, the free encyclopedia, 2020. URL https://en.wikipedia.org/w/index.php?title=Sign_test&oldid=956142528. [Online; accessed 1-June-2020].

Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

Lei Xu, Adam Krzyzak, and Ching Y Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE transactions on systems, man, and cybernetics*, 22(3):418–435, 1992.

Bianca Zadrozny and Charles Elkan. Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 204–213. ACM, 2001a.

Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In *ICML*, volume 1, pages 609–616. Citeseer, 2001b.

Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004.

He Zhang, François Petitjean, and Wray Buntine. Bayesian network classifiers using ensembles and smoothing. *Knowledge and Information Systems*, pages 1–24, 2020a.

He Zhang, François Petitjean, and Wray Buntine. Hierarchical gradient smoothing for probability estimation trees. In *Advances in Knowledge Discovery and Data Mining*, pages 222–234, Cham, 2020b. Springer International Publishing. ISBN 978-3-030-47426-3.

Wenliang Zhong and James T Kwok. Accurate probability calibration for multiple classifiers. In *IJCAI*, pages 1939–1945, 2013.

Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman and Hall/CRC, 1st edition, 2012.

Zhi-Hua Zhou. *Ensemble Learning*, pages 411–416. Springer US, Boston, MA, 2015.

Zhi-Hua Zhou and Xu-Ying Liu. On multi-class cost-sensitive learning. *Computational Intelligence*, 26(3):232–257, 2010.