Universidad Politécnica de Madrid
Facultad de Informática

Master Thesis
MASTER IN ARTIFICIAL INTELLIGENCE RESEARCH

bayeclass: an R package for learning Bayesian network classifiers. Applications to neuroscience

Author: Bojan Mihaljevic
Supervisor: Pedro Larrañaga Múgica
Supervisor: Concha Bielza Lozoya

July 2013

Acknowledgements

First, I want to thank my supervisors Concha Bielza and Pedro Larrañaga for their trust, for all the valuable advice, and for the patience during the execution of this Master's Thesis.

I would like to thank Javier deFelipe and Ruth Benavides-Piccione from the *Laboratorio Cajal de Circuitos Corticales*(CSIC) for their contributions to this work.

This Master's Thesis would have not been possible without the financial support of the Spanish Economy and Competitiveness Ministry through Cajal Blue Brain (C080020-09) and TIN2010-20900-C04-04 projects.

I want to thank my colleagues from the *Computational Intelligence Group* and the *Máster Universitario en Inteligencia Artificial* for their company and guidance. Also to the rest of my friends, be they in Madrid or somewhere else. Finally, I want to thank my parents and my sister for their love and support.

Resumen

El aprendizaje automático proporciona herramientas para el análisis automatizado de datos. Un tipo de algoritmo de aprendizaje automático, el clasificador supervisado, aprende la correspondencia entre las características descriptivas de un objeto y el conjunto de clases posibles, a partir de un conjunto de datos.

Los clasificadores supervisados basados en redes Bayesianas son particularmente útiles. Existen muchos algoritmos para el aprendizaje de tales clasificadores. Sin embargo, sólo dos de ellos están disponibles en software de libre acceso. El entorno de software R es el sistema de código abierto líder para la computacion estadística. Proporcionamos una implementación de clasificadores Bayesianos avanzados para entorno R, en forma de un paquete al que llamamos bayesclass.

El clasificador Bayesiano más conocido es el naive Bayes. Este clasificador asume que las características son independientes dada la clase. En muchos ámbitos, es posible obtener una clasificación más precisa relajando de estos supuestos. El modelo semi-naive Bayes elimina los supuestos de independencia condicional dentro de subconjuntos disjuntos de características. El algoritmo de aprendizaje *backward sequential elimination and joining* (BSEJ), tiende a producir modelos semi-naive Bayes con pequeños subconjuntos de características relacionadas, eliminando pocos de los supuestos de independencia del naive Bayes. Extendemos el algoritmo BSEJ con un segundo paso que elimina algunos supuestos de independencia injustificados. Nuestro clasificador supera a la BSEJ y otros cinco clasificadores Bayesianos en un conjunto de bases de datos de referencia.

La clasificación de neuronas es un problema importante en neurociencia. Anteriormente, 42 expertos clasificaron un conjunto de neuronas de acuerdo a una taxonomía propuesta. Los expertos no coincidieron en muchos de los términos de la taxonomía. Los datos recogidos permiten construir un modelo computacional objetivo que podría resolver los conflictos asignando etiquetas de clase definitivas. Un clasificador supervisado puede aprender el mapeo entre los descriptores cuantitativos neuronales y las elecciones taxonómicas de los expertos. Un problema es que hay hasta 42 etiquetas de clase por neurona y la etiqueta más votada no es siempre fiable (puede tener pocos votos). Construimos los clasificadores utilizando únicamente las neuronas con etiquetas fiables y así obtenemos una alta precisión en la predicción.

Abstract

Machine learning provides tools for automatized analysis of data. The most commonly used form of machine learning is supervised classification. A supervised classifier learns a mapping from the descriptive features of an object to the set of possible classes, from a set of features-class pairs. Once learned, it is used to predict the class for novel data instances. The Bayesian network-based supervised classifiers are particularly useful. They have a solid theoretical basis in probability theory and provide competitive predictive performance. Many algorithms for learning Bayesian network classifiers exist. However, only two are provided in freely available software. The `R` software environment is the leading open-source system for statistical computing. We provide an implementation of state-of-the-art Bayesian network classifiers for the R environment, in the form of an add-on package called `bayesclass`.

The best-known Bayesian network classifier is the naive Bayes. It assumes that the features are independent given the class. In many domains, more accurate classification is obtained by relaxing these assumptions. The semi-naive Bayes model removes all assumptions of conditional independence within disjoint subsets of features. Its state-of-the-art learning algorithm, the backward sequential elimination and joining (BSEJ) algorithm, tends to produce semi-naive Bayes models with small subsets of related features. Such a model removes only a few of naive Bayes' independence assumptions. We extend the BSEJ algorithm with a second step which removes some of its unwarranted independence assumptions. Our classifier outperforms the BSEJ and five other Bayesian network classifiers on a set of benchmark databases, although the difference in performance is not statistically significant.

Classification of neurons is an important problem in neuroscience. Previously, 42 experts classified a set of interneurons according to a proposed taxonomy. They disagreed on many of the terms of the taxonomy. The gathered data allows for constructing an objective computational model that could resolve the conflicts by assigning the definitive class labels. A supervised classifier can learn the required mapping from quantitative neuronal descriptors to the experts' taxonomical choices. A challenge is that there are up to 42 class labels per neuron and the most voted label is not always reliable (i.e. has many votes). We use only the cells with reliable class labels to build the classifiers and obtain high predictive accuracy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An enormous amount of data is being generated every day. Analysing big data sets is impossible without the help of automated procedures. Machine learning [7, 48] provides these procedures. The most commonly used form of machine learning is supervised classification [7, 15]. Its goal is to learn a mapping from the descriptive features of an object to the set of possible classes, given a set of features-class pairs.

Probabilities play a central role in modern machine learning [7, 31, 48]. Probabilistic graphical models (PGMs) [39] have emerged as a general framework for describing and applying probabilistic models [7]. A PGM allows us to efficiently encode a joint distribution over some random variables by making assumptions of conditional independence. Bayesian networks [51] are the most commonly used kind of graphical models.

A Bayesian network classifier (BNC) [21] is a Bayesian network applied to the classification task. BNCs have many strengths, including: interpretability, possibility of including prior knowledge about a domain, and competitive predictive performance. They have been successfully applied in practice (e.g. [8, 40]).

Many algorithms for learning BNCs from data have been proposed [2, 18]. Only a couple, namely the naive Bayes [47] and the tree augmented naive Bayes [21], are provided in freely available software. `R` [53] is a programming language and an environment for statistical computing and graphics. It is free software, released under the GNU General Public License (GPL) and runs on all common operating systems. It is the leading open-source system for statistical computing [16, 34]. `R` consists of a base distribution and add-on packages, contributed by members of its open-source community. Through the add-on packages, many models for supervised classification are available [42]. However, only a pair of Bayesian network classifiers are provided.

We set out to extend the `R` platform with implementations of state-of-the-art algorithms for learning Bayesian network classifiers from data. We provide our code

in the form of an add-on package called `bayesclass`. Chapter 3 describes the implementation of this package.

The above-mentioned naive Bayes classifier is the best-known Bayesian network classifier. It assumes that the features are independent given the class. These assumptions are violated in many domains and more accurate classification can often be obtained by relaxing them [21]. The semi-naive Bayes [50] model removes all assumptions of conditional independence within disjoint subsets of features. The state-of-the-art algorithm for learning a semi-naive Bayes is the backward sequential elimination and joining (BSEJ) algorithm [50]. This algorithm tends to form small subsets of related features and therefore removes few of the independence assumptions of the naive Bayes [21].

We extend the BSEJ algorithm with a second step which removes some of its unwarranted conditional independence assumptions. We use statistical tests of conditional independence to assess whether an assumption of conditional independence holds in the data. We refer to our proposed classifier as augmented semi-naive Bayes. In Chapter 2 we describe this proposal and report an empirical comparison with the BSEJ algorithm and with five other reference Bayesian network classifiers.

Classification of neurons according to their morphology is an important problem in neuroscience. In a previous study, 42 experts were asked to classify a representative set of interneurons according to a proposed taxonomy. They disagreed on many of the terms of the taxonomy. The gathered data allows for constructing an objective computational model that could resolve the conflicts by assigning the definitive class labels. A supervised classifier can learn the required mapping from quantitative neuronal descriptors to the experts' taxonomical choices. A challenge is that there are up to 42 class labels for each interneuron and the most voted label is not always reliable. We use only the cells with reliable class labels (i.e. those backed by many votes) to build the Bayesian network classifiers. We show that classifiers learned in this way can achieve high predictive accuracy.

This dissertation is organized as follows: in the rest of the present chapter we introduce discrete Bayesian network classifiers and describe the Bayesian network classifiers that are referred to in the remaining chapters; in Chapter 2, we present the augmented semi-naive Bayes classifier; in Chapter 3 we describe the `bayesclass` package for learning Bayesian network classifiers; in Chapter 4 we describe the application of Bayesian network classifiers to the classification of interneurons into morphological classes; and finally, in Chapter 5, we sum up with conclusions and an outline of future work.

1.1 Fundamentals

We use upper-case letters to denote variables (X) and lower-case letters (x) to denote variable values. We use boldface letters to denote multidimensional vectors of variables ($\mathbf{X}$) and variable values ($\mathbf{x}$). Let $\mathbf{X} = (X_1, \ldots, X_n)$ be a vector of n discrete predictor random variables or features, with $x_i \in \{1, \ldots, r_i\}$ and let C the the class variable, with $c \in \{1, \ldots, r_c\}$. The supervised classification problem consists in inducing from a random sample $\mathcal{D} = \{(\mathbf{x}^{(\mathbf{1})}, c^{(1)}), \ldots, (\mathbf{x}^{(\mathbf{N})}, c^{(N)})\}$, of size N, a classifier able to assign labels to new instances given by the values of their features. A *Bayes classifier* assigns an instance $\mathbf{x}$ to the most probable class, i.e.

$$c^* = \arg\max_c p(c|\mathbf{x}) = \arg\max_c p(c, \mathbf{x}).$$

A Bayesian network classifier [21] uses a Bayesian network [**?**] to encode $p(c, \mathbf{x})$. A Bayesian network consists of two components: a directed acyclic graph (DAG) $\mathcal{G} = (\mathbf{X}, \mathbf{A})$ and a set of parameters $\boldsymbol{\Theta}$. The nodes $\mathbf{X}$ in $\mathcal{G}$ correspond to random variables and the arcs represent direct dependencies between them. The graph structure encodes conditional independence assumptions about the variables: a variable X is independent of its non-descendants given $\mathbf{Pa}(X)$, its parents in $\mathcal{G}$. From there it follows that

$$p(c, \mathbf{x}) = p(c|\mathbf{pa}(c)) \prod_1^n p(x_i|\mathbf{pa}(x_i)).$$

The number of parameters is then exponential in the size of the conditioning sets $\mathbf{Pa}(X)$ rather than in n. Many Bayesian network classifiers use the following factorization

$$p(c, \mathbf{x}) = p(c) \prod_1^n p(\mathbf{x}|c),$$

that is, they fix the class as the root of the network (i.e., $\mathbf{Pa}(C) = \emptyset$). They differ in how they factorize $p(\mathbf{x}|c)$, that is, in the assumptions of class-conditional independence among the features they make. For example, the naive Bayes (Section 1.3) assumes that all features are conditionally independent given the class:

$$p(\mathbf{x}, c) \propto p(c) \prod_{i=1}^n p(x_i|c).$$

The parameters of a Bayesian network, $\boldsymbol{\Theta}$, quantify conditional probability distributions implied by $\mathcal{G}$. That is, $\boldsymbol{\theta}_{X_i|\mathbf{Pa}(X_i)} = p(X_i \mid \mathbf{Pa}(X_i))$ and $\boldsymbol{\Theta} = (\boldsymbol{\theta}_{X_1|\mathbf{Pa}(X_1)}, \ldots, \boldsymbol{\theta}_{X_n|\mathbf{Pa}(X_n)})$. Once a structure is fixed, we can estimate $\boldsymbol{\Theta}$ from $\mathcal{D}$. Standard methods include maximum likelihood and Bayesian estimation. Using a Dirichlet prior with all hy-

perparameters equal to α, the Bayesian estimate of $p(X_i = k|\mathbf{Pa}(X_i) = j)$ is

$$\frac{N_{ijk} + \alpha}{N_{.j.} + r_i\alpha}, \tag{1.1}$$

where N_{ijk} is the frequency in $\mathcal{D}$ of cases with $X_i = \mathrm{k}$ and $\mathbf{Pa}(X_i) = j$, and $N_{.j.}$is the frequency of cases with $\mathbf{Pa}(X_i) = j$. Maximum likelihood estimation can be seen as a special case of Equation 1.1 where $\alpha = 0$.

1.2 Learning Bayesian Networks from Data

Learning Bayesian networks from data is a two-step procedure: structural learning and parameter fitting. These two steps correspond to the identification of the graph $\mathcal{G}$ and the parameters of local probability distributions, $\mathbf{\Theta}$, respectively. The methods for structural learning can be divided into two categories:

- Constraint-based methods, which use tests of conditional independence to find conditional independences between groups of variables.
- score+search methods, which assign a score to each candidate network and use a search algorithm to explore the space of networks (DAGs). Since learning the optimal Bayesian network is NP-hard [11], heuristic search algorithms are generally used. Typical scores include the log-likelihood and the penalized log-likelihood (e.g. the Bayesian information criterion).

1.2.1 Tests of Independence and Conditional Independence

Two statistics are commonly used to test for conditional independence of variable sets $\mathbf{X}$ and $\mathbf{Y}$ given $\mathbf{Z}$: the mutual information $\mathbf{X}$ and $\mathbf{Y}$ given $\mathbf{Z}$, $I(\mathbf{X}; \mathbf{Y}|\mathbf{Z})$, and Pearson's χ^2 statistic [49]. The χ^2 statistic and $2NI(\mathbf{X}; \mathbf{Y}|\mathbf{Z})$ asymptotically follow the $\chi^2_{(r_\mathbf{X}-1)(r_\mathbf{Y}-1)r_\mathbf{Z}}$ distribution, where $r_\mathbf{X} = \prod_{X\in\mathbf{X}} |X|$, and analogously for $r_\mathbf{Y}$ and $r_\mathbf{Z}$. We can use this distribution to obtain a p-value for the hypothesis of conditional independence. When $\mathbf{Z} = \emptyset$ then $r_\mathbf{Z} = 1$ and we are testing for (unconditional) independence.

1.2.2 Learning Bayesian Network Classifiers

Learning Bayesian network classifiers is a special case of Bayesian network learning. The typical goals of the latter include the discovery of knowledge and a good approximation of the underlying distribution, while in the former case we are primarily interested in learning a model that maximizes predictive performance. It seems that

the two goals do not imply each other so the two procedures can differ. For example, when learning a Bayesian network classifier it generally makes sense to omit some features from the model as long as that improves predictive performance. Similarly, it makes sense to use the predictive accuracy of a network as its score. These operations seem to make little or no sense in general Bayesian network learning. Generally, any operation that improves predictive accuracy appears to be useful in the context of Bayesian network classifier learning.

Predictive Accuracy as Network Score

The predictive accuracy of a network structure $\mathcal{G}$ can be estimated with cross-validation [9]. The network's parameters are estimated from the training set and the performance of the resulting network determined against the validation set. The average performance over the test sets gives the cross-validated estimate.

Restricted Network Structures

Most of the algorithms for learning Bayesian network classifiers impose constraints on the network structure. The network structure of the tree augmented naive Bayes (Section 1.6), for example, consists of a tree in the features subgraph and an arc from the class node to each feature (the class has no parents). Most methods fix the class as the root of the network. Given the restrictions on network structure, general search heuristics are often not suitable and specific search algorithms are used for learning Bayesian network classifiers.

1.3 Naive Bayes

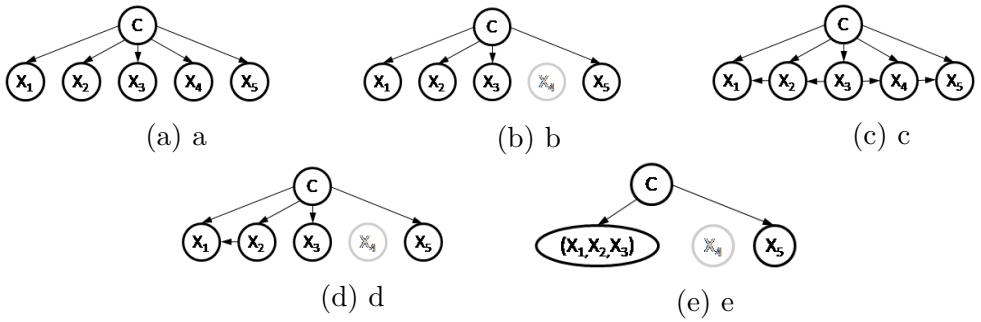
The naive Bayes model [47] assumes that all features are independent given the class:

$$p(c|\mathbf{x}) \propto p(c) \prod_{i=1}^{n} p(x_i|c)$$

See Figure 1.1a for the corresponding network structure. When all variables are discrete, decision surface of the naive Bayes is a hyperplane [52]. Nonetheless, it shows competitive performance in several domains [30]. Its low variance (due to its simplicity) can be useful when N is small or n is high compared to N.

1.3.1 Weighted Naive Bayes

Extending the naive Bayes with additional parameters can improve its accuracy. The attributed weighted naive Bayes [27] is a simple filter method for setting feature



(a) a (b) b (c) c (d) d (e) e

Figure 1.1: a) Naive Bayes (Section 1.3); b) Selective naive Bayes (Section 1.4); c) Tree augmented naive Bayes (Section 1.6); d) Selective tree augmented naive Bayes (Section 1.6.1); e) Semi-naive Bayes (Section 1.5)

weights, obtaining

$$p(c|\mathbf{x}) \propto p(c) \prod_{i=1}^{n} p(x_i|c)^{w_i}. \tag{1.2}$$

A feature weight $w_i \in [0-1]$ is inversely proportional to the dependence of X_i on the rest of features. A features's dependency is estimated by constructing unpruned decision trees and looking at the depth at which attributes are tested in the tree. A bagging procedure is used to stabilize the estimates. The weight of a feature X_i is given as

$$w_i = \frac{\sum_{j=1}^{M} \frac{1}{\sqrt{d_{ij}}}}{M},$$

where M is the number of bagged trees and d_{ij} the minimum depth that X_i is tested at in j-th tree ($d_{ij} = 0$ if X_i does not appear in j-th tree). Implicit feature subset selection is performed as a weight $w_i = 0$ omits the effect of X_i on prediction. The size of the bootstrap sample and the number of samplings (i.e. trees) are the parameters of this method.

The adjusted probability naive Bayes classifier (APNBC) [59] assigns a numeric weight $w_c \in [0, \infty)$ to each class. Upon classification, the class posterior is multiplied by this weight. Then, we have

$$p(c|\mathbf{x}) \propto w_c p(c) \prod_{i=1}^{n} p(x_i|c).$$

The weights are found with a hill-climbing search that maximizes resubstitution accuracy. All weights are initialized to 1, and at each step a weight w_c^* which maximizes resubstitution accuracy is identified. If w_c^* improves resubstitution accuracy (see be-

low), it is incorporated into the classifier (i.e. $w_c = w_c^*$) and the process is repeated; otherwise the search halts.

Weights are continuous values so the search space is infinite. However, only $2N_m$ candidate weights that are computed at each step, where N_m is the number of misclassified instances. For each misclassified instance $\boldsymbol{x}^{(i)}$, candidate weights are computed for its true class $c^{(i)}$ and the class assigned to it by the classifier, $a^{(i)}$, as follows:

$$w_{ic^{(i)}} = \frac{w_{a^{(i)}} p(a^{(i)}|\boldsymbol{x}^{(i)})}{p(c^{(i)}|\boldsymbol{x}^{(i)})} + \epsilon, \tag{1.3}$$

and

$$w_{ia^{(i)}} = \frac{w_{c^{(i)}} p(c^{(i)}|\boldsymbol{x}^{(i)})}{p(a^{(i)}|\boldsymbol{x}^{(i)})} - \epsilon,$$

where ϵ is a small positive value. The best weight, w_c^*, is the $w_{ic^{(i)}}$ or $w_{ia^{(i)}}$ which maximizes resubstitution accuracy (in case of ties, the weight which least differs from current weight is selected).

Before being included in $\mathbf{w}$ (i.e. before setting $w_c = w_c^*$), w_c^* is replaced by a midpoint between its value and a "critical" value b. If $w_c^* > w_c$ (i.e. w_c^* was obtained with Equation 1.3) then b is the highest value less than w_c^* such that setting $w_c = b$ results in a higher classification error than $w_c = w_c^*$. Analogously, when $w_c^* < w_c$, b is the lowest value greater than w_c^* which results in a higher classification error than w_c^*.

To manage the risk of overfitting due to the maximization of resubstitution accuracy (instead of an honest estimate of accuracy), w_c^* is only included in $\mathbf{w}$ if it leads to an improvement in accuracy that is unlikely to have been obtained by chance. If it is not, then the search is halted. The probability of the improved accuracy, acc_{impr}, is the probability of observing it after N trials when the true probability of the underlying Binomial distribution is acc_{curr}, the current accuracy. The threshold α for determining whether this probability is higher than chance is a parameter of the APNBC method.

1.4 Selective Naive Bayes

Redundant features degrade the predictive performance of the naive Bayes [43]. Pruning the feature set can alleviate this problem [43]. If we denote the pruned feature set with $\mathbf{X}_F, F \subseteq \{1, \ldots, n\}$, then

$$p(c|\mathbf{x}) \propto p(c)p(\mathbf{x}_F|c) = p(c) \prod_{i \in F} p(x_i|c).$$

See Figure 1.1b for the network structure of the selective naive Bayes. Finding the optimal feature subset is an instance of the feature subset selection task. There are two main approaches to this task: the filter approach and the wrapper approach. The filter approach selects features without considering their effect on the classifier. It usually scores feature subsets using statistics, such as the mutual information, computed from the empirical distribution. Most filter methods do not account for correlations between features but tend to be robust to overfitting [56]. The wrapper approach [38] uses the classifier as a black box to score feature subsets. Any measure of predictive performance, such as accuracy or the AUC, can be used as a score. A wrapper typically uses a resampling technique, such as cross-validation, to assess the predictive performance of a classifier; it is therefore, in general, more computationally costly than the filter. In the case of Bayesian network classifiers, accuracy is estimated as explained in Section 1.2.2

A major issue in feature subset selection is how to search the space of feature subsets since its cardinality is 2^n. Except for a small n, we need to use a search heuristic. Coarse heuristics, such as the forward search (explained below), are faster and tend to be more robust to overfitting than more sophisticated ones [26].

1.4.1 The Forward Sequential Selection Algorithm

The forward sequential selection (FSS) algorithm [43] is a wrapper algorithm for learning a selective naive Bayes. It uses the forward selection heuristic to traverse the search space: starting from an empty set, features are progressively incorporated into the classifier. The search stops when there is no improvement in accuracy.

1.4.2 The Filter Forward Sequential Selection Algorithm

The filter forward sequential selection (FFSS) algorithm [8] is a filter approach for learning a selective naive Bayes. It incorporates in the models all features that are not independent from the class, as deemed by the test of independence based on mutual information (see Section 1.2.1) with threshold α. The threshold α is a parameter of this method.

1.5 Semi-naive Bayes

Naive Bayes's assumptions of conditional independence are violated in many domains and more accurate classification can often be obtained by dispensing with assumptions unwarranted by the data [21]. A common approach to this is to augment the structure of naive Bayes with arcs between features, obtaining an augmented naive Bayes model [21]. The semi-naive Bayes [50] is an example of such a model. It

assumes that correlations exist only within disjoint subsets of features. No independence assumptions are made within a feature subset, i.e., each feature depends directly on every other. This means that the structure of a naive Bayes is augmented with an arc between every pair of features in the same feature subset. For simplicity, we depict the a subset of related features with a compound node corresponding to the Cartesian product of the features within the subset (see Figure 1.1e). The semi-naive Bayes model might omit some of the features (i.e. it performs embedded feature subset selection). According to the semi-naive Bayes model,

$$p(c|\mathbf{x}) \propto p(c) \prod_{j \in Q} p(\mathbf{x}_{S_j}|c), \tag{1.4}$$

where $S_j \subseteq \{1, \ldots, n\}$ is the j-th feature subset, $Q = \{1, \ldots, K\}$ is the set of indices of feature subsets, and the following conditions hold: $\cup_{j \in Q} S_j \subseteq \{1, 2, ..., n\}$ and $S_j \cap S_l = \emptyset$, $j \neq l$.

The number of possible partitions of the feature set into disjoint subsets grows faster than exponential in n. That justifies the use of heuristics for learning a semi-naive Bayes from data.

1.5.1 The Backward Sequential Elimination and Joining Algorithm

The backward sequential elimination and joining (BSEJ) algorithm [50] is probably the best-known method for learning a semi-naive Bayes from data. Its bias component of error is low [64] which suggests that it is suitable when N is large. It uses a greedy search which, starting from the structure of a naive Bayes (where each feature is a singleton feature subset), considers two operations in each step:

a) Removing a feature X_i from the model

b) Creating a new subset of related features $\mathbf{X}_{S_k}$ by merging two subsets, $\mathbf{X}_{S_j}$ and $\mathbf{X}_{S_i}$, $i \neq j$

A cross-validation estimate of predictive accuracy is used to evaluate the candidate operations and the better one is chosen. If no operation improves the accuracy of the current structure, the search stops.

1.5.2 The Filter Forward Sequential Selection and Joining Algorithm

The filter forward sequential selection and joining (FFSSJ) ([8]) algorithm uses a variant of the forward search to learn a semi-naive Bayes. Starting from a network

structure which contains no features (i.e just the class node), it evaluates two ways of including a feature X_i in the model:

a) As independent of other features

b) As related to every feature X_j in a subset of related features $\mathbf{X}_{S_k}$

Each of the operations produces a candidate subset of related features $\mathbf{X}_{S_{new}}$. The $\mathbf{X}_{S_{new}}$ produced by a) is a singleton $\{X_i\}$ and that produced by b) is $X_i \cup \mathbf{X}_{S_k}$. Each $\mathbf{X}_{S_{new}}$ is scored with the p-value of the test of independence of $\mathbf{X}_{S_{new}}$ and C (see Section **??**) and the best $\mathbf{X}_{S_{new}}$ is selected. If its x-value lower than a threshold α then it is incorporated into the model; otherwise the search stops.

1.6 Tree Augmented Naive Bayes

The tree augmented naive Bayes (TAN) [21] is a well-known Bayesian network classifier which outperforms naive Bayes in many domains [21]. It augments the naive Bayes with a tree in the features subgraph. That is, it conditions every feature except one (the root of the tree) on exactly one other feature (see Figure 1.1c for the network structure). According to the TAN model,

$$p(c|\mathbf{x}) \propto p(c)p(x_r|c)\prod_{i\neq r} p(x_i|x_{j(i)}, c),$$

where X_r is the root of the tree. The augmenting tree which maximizes the likelihood of the TAN is found efficiently using a variant of Chow-Liu's [12] algorithm given in [21]. First, class-conditional mutual information $I(X_i; X_j|C)$ is computed for every pair of features. A complete graph, $\mathcal{G}^{CL} = (\mathbf{X}, \mathbf{E})$, is then built, with class-conditional mutual information values $I(X_i; X_j|C)$ as edge weights. Then, the maximum weighted spanning tree (MWST) is found using by applying Kruskal's algorithm [41] on $\mathcal{G}^{CL}$. The MWST tree is directed by choosing a root node (does not matter which one) and directing the arcs away from it. This gives the directed augmenting tree which is added to the structure of a naive Bayes. The TAN does not perform feature subset selection and always it always augments the naive Bayes with $n-1$ arcs.

1.6.1 Selective Tree Augmented Naive Bayes

The selective tree augmented naive Bayes (STAN) algorithm [8] is a a variant of TAN which performs feature subset selection and may augment naive Bayes with less than $n-1$ arcs. Before learning the augmenting tree(s), STAN filters out features and

inter-feature dependencies (i.e. its $\mathcal{G}^{CL}$ is not necessarily complete). If all direct dependencies between some pair of feature sets are filtered out (their edges omitted from $\mathcal{G}^{CL}$), then a single spanning tree cannot be built, as no edges exist between these sets of features (nodes) in $\mathcal{G}^{CL}$. In this case, a maximum spanning tree is obtained over each connected set of features. Each of these trees is then directed as in the case of TAN and then added to the naive Bayes network structure (see Figure 1.1d). Then, according to the STAN model,

$$p(c|\mathbf{x}) \propto p(c) \prod_{i \in R} p(x_i|c) \prod_{i \in F \setminus R} p(x_i|x_{j(i)}, c),$$

where $\mathbf{X}_F, F \subseteq \{1, \ldots, n\}$ are the features that are included in the model, $R \subseteq F$ are the roots of the trees, and $\{X_{j(i)}\} = \mathbf{Pa}(X_i) \setminus C$, $i \notin R$, are the parent features of X_i.

The first step of the STAN algorithm is feature subset selection. It selects all features that are not independent from the class, as deemed by the test of independence based on mutual information (see Section 1.2.1) with threshold α. This is the same as running the filter forward sequential selection (FFSS) algorithm (see Section 1.4.2). In the second step, STAN discards dependencies that are not warranted by the data. That is, if the hypothesis of class-conditional independence of two features is not rejected at some threshold α, then this dependence will not be included in $\mathcal{G}^{CL}$. The authors develop a heuristic for testing conditional independence: if X_i and X_j are independent given some class value c (as deemed by the test of independence based on mutual information) then X_i and X_j are considered as conditionally independent given C. Finally, the augmenting tree(s) are obtained by running Kruskal's algorithm on $\mathcal{G}^{CL}$[1], i.e., in the same way as in the TAN algorithm. Even though $\mathcal{G}^{CL}$ might not be connected (there might be no edges between some subgraphs), running Kruskal's algorithm over the entire graph will return the MWST for each of the connected subgraphs [48, p. 912].

1.7 Other Bayesian Network Classifiers

There are many other Bayesian network classifiers which are not used in this dissertation. For example, the k-dependence Bayesian classifier [57] extends the TAN to allow a feature to have up to k parent features. The SuperParent-One-Dependence Estimator (SPODE) [36] is a special case of TAN where a single feature is parent of all other features. The NBTree [37] is a hybrid classifier combining the naive Bayes and decision trees. It partitions the training data using a tree structure and

[1]The authors of [8] run Kruskal's algorithm on each connected component of $\mathcal{G}^{CL}$, which is equivalent to what is described here.

establishes a local naive Bayes with non-tested variables in each leaf. We refer the interested reader to reviews by [18] and [2] and the empirical comparison by [64].

Chapter 2

Augmented Semi-naive Bayes

As we have seen, the semi-naive Bayes (Section 1.5) removes some of naive Bayes' conditional independence assumptions. It assumes that correlations exist only inside disjoint subsets of features and these correlations are "full", i.e., each feature depends directly on every other. The backward sequential elimination and joining (BSEJ) algorithm induces semi-naive Bayes classifiers which are competitive with other Bayesian network classifiers [50, 64]. This algorithm does not, in general, form large subsets of related features, since these lead to a poor cross-validated estimate of prediction accuracy [21]. Therefore, the BSEJ tends to partitions the feature set into many small subsets. Since the subsets are conditionally independent given the class, it captures only a few correlations among the features.

We set out to extend the BSEJ algorithm with a second step which removes some of its unwarranted (by the data) independence assumptions. Our procedure is inspired by the selective tree augmented naive Bayes algorithm [8].

We perform an empirical comparison of our proposal with the BSEJ algorithm and with five other Bayesian network classifiers. The rest of this chapter is organized as follows: Section 2.1 describes the proposed extension of the BSEJ algorithm; Section 2.2 reports the empirical evaluation of our proposal; and, finally, Section 2.3 provides some concluding remarks.

2.1 Augmented Semi-Naive Bayes

We set out to improve the predictive accuracy of a semi-naive Bayes by correlating some of its disjoint (and conditionally independent) feature subsets. Just before outputting the final semi-naive Bayes model, the BSEJ algorithm considers correlating each pair of feature subsets (see Section 1.5.1, list item b)) and finds that no correlation improves its estimate of accuracy. We incorporate several (up to $K-1$, where K is the number of disjoint feature subsets) of those correlations in the model, as

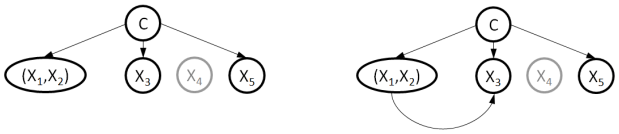


Figure 2.1: a) Semi-naive Bayes (Section 1.5); b) Augmented semi-naive Bayes

long they are warranted by the data. This way, a structure that was not considered by the BSEJ is obtained. The computational cost of this augmenting step is small compared to that of the BSEJ algorithms, which is wrapper-based.

The structure of an augmented semi-naive Bayes (ASB) is a forest (a set of trees) over the subsets of related features (see Fig. 2.1). Then,

$$p(c, \mathbf{x}) = p(c) \prod_{i \in R} p(\mathbf{x}_{S_i}|c) \prod_{i \in Q \setminus R} p(\mathbf{x}_{S_i}|\mathbf{x}_{j(i)}, c),$$

where Q and S_i are defined as in Equation (1.4), $R \subseteq Q$ are indices of feature subsets that are root(s) of the augmenting trees(s), and $\{\mathbf{X}_{j(i)}\} = \mathbf{Pa}(X_{S_i}) \setminus C$.

The augmenting trees are obtained by maximizing the likelihood of the ASB model. The procedure is similar to that used by the STAN algorithm but adapted to add arcs between non-singleton feature subsets. A node Q_j of the complete graph, $\mathcal{G}^{CL} = (\mathbf{Q}, \mathbf{E})$, corresponds to subset of related features $\mathbf{X}_{S_j}$ and the weights are the conditional mutual information values $I(\mathbf{X}_{S_i}; \mathbf{X}_{S_j}|C)$. If the hypothesis of class-conditional independence of two disjoint subsets of features, $\mathbf{X}_{S_i}$ and $\mathbf{X}_{S_j}$, is not rejected at some threshold α, then the corresponding edge will be omitted from $\mathcal{G}^{CL}$. We use the test of conditional independence based on mutual information (see Section 1.2.1). The maximum weighted spanning tree(s) (MWSTs) are obtained by running Kruskal's algorithm on $\mathcal{G}^{CL}$ and they are then directed as described in Section 1.6. The directed MWST trees T encode the augmenting dependencies between feature subsets and specify how to relate the individual features that they comprise. For each arc from a node Q_i to a node Q_j in $T_i \in T$, we include in the semi-naive Bayes an arc from each $X_l \in \mathbf{X}_{S_i}$ to each $X_k \in \mathbf{X}_{S_j}$. See Algorithm 1 for pseudo-code of the full procedure.

The χ^2 approximation used in the test of conditional independence is not reliable when there are little cases in the contingency table over the variables being tested ($\mathbf{X}_{S_i}$, $\mathbf{X}_{S_j}$, and C, in this case) [1]. In order to avoid spurious arcs due to low reliability of the χ^2 approximation, we omit from $\mathcal{G}^{CL}$ an edge Q_i–Q_j whenever the asymptotic test of independence for $\mathbf{X}_{S_i}$ and $\mathbf{X}_{S_j}$ given C in not reliable. Following [63], we consider a χ^2 approximation to be reliable if the average cell count in the contingency table is at least 5.

Our procedure differs from the STAN algorithm in four aspects: it does not

perform feature subset selection; it can relate non-singleton feature subsets; it uses a more standard conditional independence test; and it verifies that the conditional independence test is reliable (assumes independence if it is not).

Algorithm 1 Augmented semi-naive Bayes

1. $\mathcal{G}^{SB} \leftarrow$ a semi-naive Bayes network structure
2. $\mathbf{S} \leftarrow (S_1, \ldots, S_K) : S_j$ is a set of features correlated in $\mathcal{G}^{SB}$
3. **for all** $j = 1, \ldots, K$ **do**
4. $\quad r_{S_j} \leftarrow \prod_{l \in S_j} r_l$
5. **end for**
6. $\mathcal{G}^{CL} \leftarrow (\mathbf{Q}, \mathbf{E})$, a complete undirected graph
7. Weight of $e(i, j)$, the edge between i and j, $\leftarrow I(\mathbf{X}_{S_i}; \mathbf{X}_{S_j}|C)$
8. **for all** $i, j = 1, \ldots, K, i < j$ **do**
9. $\quad$ **if** $\frac{N}{r_{S_j} r_{S_i} r_c} \geq 5$ and p-value of $2NI(X_{S_i}; X_{S_j}|C)$ from $X^2_{(r_{S_i}-1)(r_{S_j}-1)r_c} > \alpha$ **then**
10. $\quad\quad$ remove edge $e(i, j)$ from $\mathbf{E}$
11. $\quad$ **end if**
12. **end for**
13. $T \leftarrow$ MWST(s) obtained by running Kruskal's algorithm on $\mathcal{G}^{SB}$
14. $T' \leftarrow$ for each $T \in \mathbf{T}$ choose a root node at random and direct edges away from it
15. **for all** i, j such that arc$(i, j) \in T'$ **do**
16. $\quad$ augment $\mathcal{G}^{SB}$ with arcs from each X_l in $\mathbf{X}_{S_i}$ to every X_k in $\mathbf{X}_{S_j}$
17. **end for**

2.2 Experimental Evaluation

2.2.1 Setup

We compare the augmented semi-naive Bayes (ASB) algorithm to six reference algorithms for learning Bayesian network classifiers. Two of those algorithms learn a selective naive Bayes (SNB) [43] model. The forward sequential selection (FSS) algorithm [43] performs a greedy search guided by predictive accuracy while the filter forward sequential selection (FFSS) omits from the model the features that are deemed independent of the class by the χ^2 independence test. Besides SNB, we consider the naive Bayes (NB), the tree augmented naive Bayes (TAN), the selective tree augmented naive Bayes (STAN), and the backward sequential elimination and joining (BSEJ) algorithm.

Table 2.1: Data sets. #Instances column displays the number of complete instances.

No.	Data set	#Features	#Instances	#Classes
1	Balance Scale	4	625	3
2	Breast Cancer (Wisconsin)	9	683	2
3	Car	6	1728	4
4	Chess (kr vs. kp)	36	3196	2
5	Dermatology	34	358	6
6	Ecoli	7	336	8
7	House Voting 84	16	232	2
8	Ionosphere	34	351	2
9	Lymphography	18	148	4
10	Molecular Biology (Promoters)	57	106	2
11	Molecular Biology (Splice)	61	3190	3
12	Primary Tumor	17	132	22
13	Tic-tac-toe	9	958	2
14	Wine	13	178	3

We compare the classifiers over 14 natural domains from UCI repository [6] (see Table 2.1). Prior to classifier comparison, we removed incomplete rows and discretized numeric features with the MDL method [17].

For the BSEJ and the FSS, we used 5-fold stratified cross-validation to estimate predictive accuracy. For statistical tests of (conditional) independence we used a significance level of 0.05 and applied the criterion of χ^2 approximation reliability. In FFSS, if a test of independence of X_i and C is not reliable, then independence is assumed and X_i is omitted from the model. For STAN, we used the same test of conditional independence as for ASB. Laplace's correction of maximum likelihood was used to estimate parameters. We estimated predictive accuracy of the classifiers with 5 repetitions of 5-fold stratified cross-validation.

2.2.2 Results

Following [22], we performed Friedman's test [19, 20] and Iman and Devenport's correction [35] to compare the classifiers over all the data sets. Our proposal outperforms the other methods (see Table 2.2 for Friedman's ranks) although the difference is not statistically significant[1].

[1]The p-value from both Friedman's and Iman and Davenport's test was 0.2

Table 2.2: Average Friedman's ranks. Lower ranking means better performance. ASB = augmented semi-naive Bayes, STAN = selective tree augmented Bayes, FSS = forward sequential selection, BSEJ = backward sequential elimination and joining, FFSS = filter forward sequential selection, NB = naive Bayes, TAN = tree augmented naive Bayes.

Algorithm	Friedman's ranks
ASB	3.11
STAN	3.96
FSS	5.21
BSEJ	3.57
FFSS	4.35
NB	3.53
TAN	4.25
p-value$_{\text{Friedman}}$	0.20
p-value$_{\text{Iman-Davenport}}$	0.20

The ASB significantly[2] improves on BSEJ on four data sets (car, chess, ionosphere, and tic-tac-toe. See Table 2.3 for accuracies.). The BSEJ outputs a model similar to the NB on those data sets (e.g. on ionosphere it removes a single feature and accounts for one interaction) while the ASB heavily augments the BSEJ (e.g. on ionosphere it builds a full tree among feature groups). This shows that useful interactions missed by BSEJ can be recovered by ASB.

There is no significant difference between ASB and BSEJ on the remaining data sets. The ASB degrades BSEJ on only three data sets and the degradation is minor (by at most 0.6% accuracy). On four data sets the ASB model is identical to the BSEJ. One of those data sets - primary tumor - has many classes (22) and not many cases (132). This yields the conditional independence test unreliable for every pair of features and therefore no arcs are be added. On the other three data sets, lowering the significance threshold would have would have produced augmented BSEJ models (i.e. arcs would have been added).

2.3 Concluding Remarks

We have presented the augmented-semi naive Bayes (ASB) algorithm, a method for removing some of the unwarranted independence assumptions of a semi-naive Bayes model. The ASB is computationally inexpensive compared to the BSEJ algorithm

[2]According to Wilcoxon's signed rank test at 5% significance level

Table 2.3: Estimated accuracies (in %) of the compared classifiers. The best performing classifiers on a data set are marked in bold. Some data set names are shorter than in Table 2.1 but the order is the same. ASB = augmented semi-naive Bayes, STAN = selective tree augmented Bayes, FSS = forward sequential selection, BSEJ = backward sequential elimination and joining, FFSS = filter forward sequential selection, NB = naive Bayes, TAN = tree augmented naive Bayes.

No.	Data set	ASB	STAN	FSS	BSEJ	FFSS	NB	TAN
1	Balance Scale	72.9±2.5	73.2±2.9	**73.6±2.2**	72.8±2.3	73.3±2.3	73.3±2.3	73.2+-2.9
2	Breast Cancer	97.1±1.1	97.1±1.1	96.9±1.4	**97.5±1.0**	**97.5±1.0**	97.5±1.0	97.1±1.1
3	Car	93.3±1.6	93.5±1.5	70.0±0.1	90.0±1.8	85.1±1.7	85.3±1.4	**94.1±1.6**
4	Chess	**94.1±1.1**	92.6±0.8	**94.1±1.0**	92.2±1.1	87.8±1.4	87.8±1.4	92.4±0.9
5	Dermatology	**98.2±1.5**	98.0±1.6	95.1±3.4	**98.2±1.5**	98.0±1.6	98.0±1.6	97.1±1.7
6	Ecoli	**85.7±3.4**	**85.7±3.4**	83.4±2.8	**85.7±3.4**	**85.7±3.4**	**85.7±3.4**	84.5±3.2
7	House Voting 84	94.3±2.8	92.9±2.8	**97.0±2.4**	91.2±4.5	91.3±4.5	91.2±4.4	93.6±2.7
8	Ionosphere	92.0±3.7	91.9±3.7	90.7±3.6	90.7±3.8	90.7±4.1	90.7±4.1	**92.2±3.1**
9	Lymphography	**85.4±6.1**	82.7±5.6	78.4±7.3	85.0±6.5	82.7±7.1	84.6±6.2	83.4±6.0
10	Promoters	89.8±6.4	90.5±5.0	84±11.2	89.8±6.4	90.5±5.0	**91.7±6.2**	48.7±1.2
11	Splice	94.9±0.7	95.0±0.8	93.5±0.8	**95.5±0.8**	95.4±0.9	95.5±0.8	52.5±0.3
12	Primary Tumor	46.5±9.4	21.3±2.0	42.5±7.7	46.5±9.4	21.3±2.0	**48.3±9.3**	41.6±8.0
13	Tic-tac-toe	75.3±3.2	74.8±2.9	69.6±3.4	71.7±3.7	70.4±3.9	70.4±3.8	**75.8±2.9**
14	Wine	98.7±1.6	98.7±1.6	95.4±2.9	**98.9±1.4**	**98.9±1.4**	**98.9±1.4**	96.9±2.6

used for learning semi-naive Bayes models. Our experiments show that ASB improves BSEJ in some domains without degrading it others. The ASB outperformed BSEJ and five other Bayesian network classifiers on 14 benchmark data sets, although the improvement in performance is not statistically significant. Further experiments, over more data sets, might give more conclusive results. Since ASB seems to improve BSEJ, it might be interesting to extend the approach into augmenting other Bayesian network classifiers learned by maximizing predictive accuracy, such as the forward sequential selection algorithm for learning a selective naive Bayes.

Chapter 3

bayesclass: an R Package for Learning Bayesian Network Classifiers

Recently, many of the developments of software for graphical models (of which Bayesian network are a special case) have happened within the R platform, either in form of interface to an existing software or through new R packages [33]. The grahical models taskview in R at http://cran.r-project.com/web/views/gR.html lists around 30 packages. It is expected that this number will grow considerably, and the packages will be extended and modified.

Regarding Bayesian network classifiers, only the naive Bayes and the tree augmented naive Bayes are currently available in R. We provide an implementation of state-of-the Bayesian network classifiers in the form of an add-on package called bayesclass. More precisely, we implement the classifiers described in Section 1 and the augmented semi-naive Bayes classifier, described in Section 2. Besides algorithms for learning Bayesian network classifiers from data, bayesclass provides network structure plotting and easy assessment of predictive performance, thanks to its integration with other add-on packages.

This chapter is structured as follows: Section 3.1 introduces the R software environment; Section 3.2 surveys the currently available implementations of Bayesian network classifiers; Section 3.3 describes the main functionalities and the implementation of the bayesclass package; Section 3.4 showcases some of the functionalities; and finally, Section 3.5 sums up with a brief discussion of the implementation and on outline of future work.

3.1 The R Environment for Statistical Computing

R [53] is a programming language and an environment for statistical computing and graphics. It is free software, released under the GNU General Public License (GPL) and runs on all common operating systems. It is the leading open-source system for statistical computing [16, 34]. R consists of a base distribution and add-on packages, contributed by members of its open-source community. The base distribution contains R's basic functonality, such as the plotting functions and statistical models. Add-on packages extend R with diverse functionalities, such as graph handling, machine learning algorithms, and advanced plotting capabilities. Both the base distribution and the add-on packages are distributed through the Comprehensive R Archive Network (CRAN) [1]. Currently, there are 4689 packages on CRAN (only around 30 of those belong to the base distribution).

3.2 Discrete Bayesian Network Classifiers in R

Only the naive Bayes and the tree augmented naive Bayes are currently available in R.

3.2.1 The bnlearn Package

bnlearn [58] is a package for learning general Bayesian networks from data. It can learn the naive Bayes and tree augmented naive Bayes. It provides maximum likelihood and Bayesian parameter estimation of parameters. Cross-validation can be used to estimate the predictive performance of an algorithm or of a network structure. The network structures can be inspected by plotting or by viewing structured text output. A model can be easily modified by the user (e.g. add/remove/reverse arc). Learning algorithms can be forced to include or exclude certain arcs (black and white lists). `bnlearn` only operates on complete data.

3.2.2 Other Implementations

The `e1071` [45] and `CORElearn` [55] packages provide implementations of the naive Bayes. They only provide parameter learning and prediction.

3.3 the bayesclass Package

The main functionalities provided by the `bayesclass` package are the following:

[1] http://cran.r-project.org

- Ten learning algorithms corresponding to those described in Section 1 and the Augmented semi-naive Bayes, described in Section 2.
- The χ^2 asymptotic test of conditional independence for vectors of variables, described in Section 1.2.1
- Cross-validated estimate of predictive performance of a network structure, described in 1.2.2. It is straightforward to perform a paired comparison of learning algorithms.
- Network structure plotting, via the `Rgraphviz` package.
- Maximum likelihood and parameter estimation, via the `gRain` package

3.3.1 Implementation

`bayesclass` is complete implemented in the `R` programming language. Many of the underlying functionalities comes from other add-on `R` packages. We here describe the tasks for which `bayesclass` relies on those packages.

Graph Manipulation and Algorithms

The `graph` package [24] provides the basic class definitions and functionality, such as graph construction, node/edge/arc addition/removal, and so on. The `Rgraphviz` package [25] provides plotting functionality. Different layout algorithms are provided and many details, such as node plotting, line type and color, can be controlled by the user. The `RBGL` package [10] provides an interface to many graph algorithms (such as shortest path, connectivity etc).

Predictive Performance Assessment

The `caret` package [42] provides resampling estimation of predictive performance. Available metrics are accuracy and Cohen's kappa index. Custom metrics can also be implemented. Resampling techniques include bootstrapping and stratified k-fold cross-validation.

Bayesian Networks Parameter Estimation and Inference

The `gRain` package [32] implements the estimation of parameters from data. Both maximum likelihood and Bayesian estimation are provided. In the latter case, a single hyperparameter α has to be used for all parameters in the network. `gRain` implements exact inference, class prediction, and posterior probability computation. It can handle incomplete data during both parameter learning and inference.

3.4 Sample Session

3.4.1 Learning a Bayesian Network Classifier from Data

Once the `bayesclass` package is loaded, the `car` data set can be loaded by calling the `data` function.

```
library(bayesclass)
data(car)
str(car)
```

```
'data.frame': 1728 obs. of  7 variables:
 $ buying  : Factor w/ 4 levels "low","med","high",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ maint   : Factor w/ 4 levels "high","low","med",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ doors   : Factor w/ 4 levels "2","3","4","5more": 1 1 1 1 1 1 1 1 1 1 ...
 $ persons : Factor w/ 3 levels "2","4","more": 1 1 1 1 1 1 1 1 1 2 ...
 $ lug_boot: Factor w/ 3 levels "big","med","small": 3 3 3 2 2 2 1 1 1 3 ...
 $ safety  : Factor w/ 3 levels "high","low","med": 2 3 1 2 3 1 2 3 1 2 ...
 $ class   : Factor w/ 4 levels "unacc","acc",..: 1 1 1 1 1 1 1 1 1 1 ...
```

The `car` data frame contains seven discrete variables, stored as factors. We will predict the `class` variable and use the remaining variables as predictive features.

In order to learn a Bayesian network classifier we need to call the `bnc` function and supply the data set as the first argument. If we do not specify which column corresponds to the class, `bnc` will assume that we want to predict the last column. The `learner` argument of `bnc` tells it which learning algorithm to use.

```
tan.car <- bnc(car, learner = "tan", smooth = 0.01)
```

The `bnc` function returns an object of class `bayesclass`. We can view its network structure with the `plot` function.

```
plot(tan.car)
```

This produces the plot in Figure 3.1a. We can see that the structure corresponds to a tree augmented naive Bayes model. We can view the associated probability tables by accessing the `cptlists` element of the `bayesclass` object. To view the conditional probability distributions for the `buying` feature, we need to access the first element of `cptlists`.

```
tan.car$cptlist[[1]]
```

```
        class
buying   unacc    acc      good     vgood
  low   0.2132 0.2318 0.6664253 0.5997847
  med   0.2215 0.2995 0.3332851 0.3999077
  high  0.2678 0.2812 0.0001448 0.0001538
  vhigh 0.2975 0.1875 0.0001448 0.0001538
```

If the learning algorithm accepts some specific parameters, we specify them in the `lrn_args` argument of the `bnc` function. The following produces a selective tree augmented naive Bayes with $\alpha = 0.01$ threshold for tests of (conditional) independence.

```
stan.car <- bnc(car, learner = "stan", lrn_args = list(alpha = 0.1),
                     smooth = 0.01)
```

Plotting the network structure we can see that it is more sparse than the TAN model (Figure 3.1b). Infact, the STAN model omits the `doors` feature, probably due to the low α threshold for statistical tests. We can obtain get a list of all the features in a model with the `features` function.

```
features(stan.car)
```

```
[1] "buying"   "maint"    "persons"  "safety"   "lug_boot"
```

3.4.2 Predictive Performance Assessment

A `bayesclass` object can be used to obtain class or probability predictions. For the latter, we need to set the `result` argument of the `predict` function to `"prob"`. The following prints out the class posterior probabilities for the first five instances in the `car` data set.

```
pred.tan <- predict(tan.car, car, result = "prob")
head(pred.tan)
```

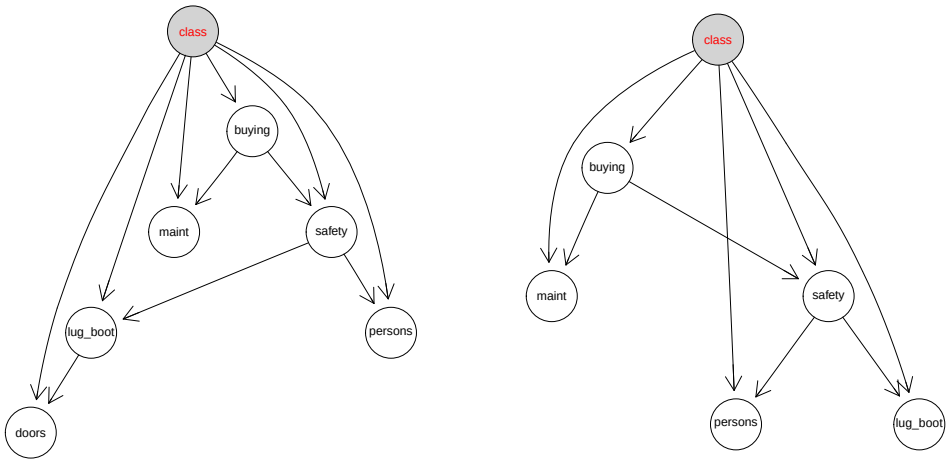
```
      unacc       acc      good     vgood
[1,]     1 1.639e-08 9.852e-06 1.722e-05
[2,]     1 2.377e-09 3.337e-12 9.870e-06
[3,]     1 7.375e-09 1.335e-08 2.367e-12
[4,]     1 2.591e-08 1.750e-05 2.795e-08
[5,]     1 9.777e-09 1.247e-08 2.246e-08
[6,]     1 1.338e-08 1.167e-08 1.103e-11
```

To obtain the predicted classes, we would just need to omit the `result` argument.

We can estimate the predictive performance of a model with resampling using the `assess` function. The following runs two repetitions of stratified 5-fold cross-validation for the tree augmented naive Bayes.

```
pred_performance <- assess(tan.car, car, k = 5, repeats = 2)
pred_performance
```

```
   learner smooth Accuracy  Kappa AccuracySD KappaSD
1   tanbc   0.01   0.9427 0.8764   0.008508 0.01837
```



(a) (b)

Figure 3.1: Network structures learned for the `car` data set. a) Tree augmented naive Bayes; b) Selective tree augmented naive Bayes with $\alpha = 0.01$

The two leftmost columns of the output specify the learning algorithm: the TAN with a smoothing value of 0.01. The remaining columns provide the estimates of accuracy and Cohen's kappa index.

We can do a paired comparison of two learning algorithms with the `assess` function. We just need to supply a list of `bayesclass` objects as the first argument to `assess`. The following compares the TAN algorithm and the STAN algorithm with an $\alpha = 0.01$.

```
compare <- assess(list(stan.car, tan.car), car, k = 5, seed = 0)
```

```
    alpha learner smooth Accuracy  Kappa AccuracySD KappaSD
1   0.1  stanbc   0.01   0.9184 0.8268    0.02284 0.04490
2    NA   tanbc   0.01   0.9433 0.8776    0.01264 0.02703
```

The `seed` argument controls random number generation and can be used to obtain reproducible results.

3.5 Concluding Remarks

The `bayesclass` extends the `R` platform with state-of-the-art Bayesian network classifiers. A minor drawback of the current implementation is that inference with the `gRain` package is very slow. The renders prediction and estimation of performance quite slow. Also, some of our algorithms are implemented in a naive way and can be further optimized.

We plan to keep extending the `bayesclass` package. More discrete Bayesian network classifiers are likely to be added in the future. We plan to implement Bayesian network classifiers which can handle real-valued features and also to allow more flexible prior probabilities for the parameters.

Chapter 4

Bayesian Network Classifiers for Discriminating between Cortical Interneurons

The problem of classifying and naming neurons has been a topic of debate for over 100 years. Nevertheless, a satisfactory consensus remains to be reached, even for restricted neuronal populations such as the GABAergic interneurons of the cerebral cortex [13]. A community-based approach towards establishing a common taxonomy was adopted in [13]. The proposed taxonomy categorizes (cortical GABAergic) interneurons according to six features of axonal arborization. One feature (that we will refer to as F5) is the interneuron type - a name commonly associated in literature with a certain pattern of axonal arborization. Other features correspond to four simple arborization patterns which are useful for interneuron classification (features F1 to F4), and the quality of the digital reconstruction of the axon - whether it is clear enough to identify the remaining five features (feature F6). 42 expert neuroscientists were asked to classify a representative set of interneurons according to the taxonomy. There was little inter-expert agreement on feature F5, as several interneuron types were found to be rather confusing (some were found to be easily distinguishable, nonetheless). The agreement was high for features F1, F2, F3, F6, and, to a lesser degree, F4.

Given the lack of consensus on interneuron types, it was concluded that a common nomenclature is currently untenable. However, a solution based on the axonal feature-based taxonomy was envisioned. A computational model, which would learn to map quantitative descriptions of interneurons into the experts' taxonomical classification choices, could be used to objectively classify interneurons. The data gathered in [13] allows machine learning [3] to be applied to produce such models. One family of machine learning algorithms - supervised classifiers [15] - learn a mapping from

inputs (e.g. neuronal descriptors) to outputs (e.g. taxonomical categories), given a set of input-output pairs, called the training set. A conventional supervised classifier requires a single output label for every training example. Each interneuron (input) is, however, associated with a taxonomical feature (output) through up to 42 labels, each due to a vote from one of the experts. The popular approach for obtaining a unique label, known as majority vote, is to select the most commonly assigned one [54]. [13] followed this approach to build supervised classifiers for the six axonal features. They obtained low predictive accuracy for axonal features F5 and F4, fairly high for F1, F2, and F3, and very high for F6. Since some categories of features F4 and F5 were confusing to the experts, it is possible that the majority votes in these cases were often unreliable indicators of true morphological properties (i.e. they might be backed by few votes). Therefore, we use only instances with reliable labels, that is, those whose majority vote count is above some threshold t, for learning supervised classifiers. A high t yields reliable labels but reduces the amount of data. Also, some taxonomical categories are omitted from the data when a high t is used as they have few reliable examples. Therefore, the classification task can be notably different between one vote count threshold t and another. We experiment with different thresholds t and produce supervised classifiers for each of them.

As supervised classifiers we use the ten Bayesian network classifiers described in Section 1. We use the thresholded majority vote approach for classifying cells according to the morphological features F1, F2, F3, F4, and F5 ([13] showed that predicting F6 is almost trivial). Furthermore, we use the simple axonal features F1 to F4 to predict the interneuron type, something that has not been studied yet.

The rest of this chapter is structured as follows: Section 4.1 describes the axonal feature-based taxonomy for cortical GABAergic interneurons; Section 4.2 describes how the was obtained and preprocessed; Section 4.3 explains the methodology used for supervised classification; Section 4.4 reports the experimental results; and finally, Section 4.5, sums up with conclusions and an outline of future work.

4.1 Background

4.1.1 The Neuron Classification Problem

Neurons have highly diverse morphological, molecular, and physiological features [5]. The scientific community has access to a vast amount of data about these features but it lacks a classification system to organize the available knowledge. One population of neurons, the GABAergic interneurons of the cortex, are remarkably variable. This variability makes it hard to unambiguously identify a set of features which define an interneuron subtype. The Petilla terminology [5] provided a nomenclature for characterizing interneurons according to their morphological, physiological, and

molecular properties. However, it does not identify features which unambiguously define an interneuron type.

4.1.2 An Axonal Features-based Taxonomy

As an alternative to the morphological characterization of the Petilla terminology, [13] proposed a pragmatic taxonomy for cortical GABAergic interneurons. It classifies interneurons according to six features of their axonal arborization. The first and second features (F1 and F2) are the laminar and columnar reaches of the axon, respectively. They distinguish between cells whose axon is either confined to a single cortical layer or column (intralaminar and intracolumnar axons) or not (translaminar and transcolumnar). The third feature (F3) is the position of the dendritic arbor relative to the axonal arbor: it is either centered or displaced. The forth feature (F4) is specific to the axons of translaminar and displaced interneurons: they either mainly ascend towards the cortical surface (ascending), mainly descend towards the white matter (descending), or both ascend and descend (both). The fifth feature (F5) is the interneuron type. Ten type names have been proposed to the experts in an experiment conducted by [13]: arcade (AR), Cajal-Retzius (CR), chandelier (CH), common basket (CB), common type (CT), horse-tail (HT), large basket (LB), Martinotti (MA), neurogliaform (NG), and other (OT). Finally, the sixth feature (F6) is in fact a feature of the digital reconstruction of the axon: whether it is clear enough to identify the remaining five features (characterized) or not (uncharacterized).

To study the potential of this taxonomy for fostering consensus among neuroscientists, [13] asked 42 experts to classify a representative set of interneurons according to the 6 features of the taxonomy. The experts agreed on the definitions of features F1, F2, and F3: when classifying a cell according to one of those features, most ascribed it to the same category. They agreed partially on F4, reaching consensus on categories ascending and descending while disagreeing on category both. Regarding interneuron type, they found some types to be easily distinguishable and others rather confusing. High-consensus types include chandelier and Martinotti, while low-consensus ones include arcade, Cajal-Retzius, common basket, large basket, and common type cells. Regarding F6, 90% of the votes was for category characterized, rendering the observed agreement high.

Supervised Classification

[13] learned a supervised classifier for each axonal feature. They used ten classifiers available in the Weka[29] data mining software package, including the discrete naive Bayes [47], an implementation of the C4.5 decision tree learner, and the random forest technique. They computed $2,886$ morphological variables with the Neurolucida (MicroBrightField) software. Besides considering the full predictor set, for each

axonal feature they considered predictor subsets obtained with the ranking based on gain-ratio (they used 500 top-ranked predictors) and with the correlation-based feature selection technique [28] which selects relevant and non-redundant predictors. They obtained a high accuracy, 99.17%, for F6, and fairly high accuracies for F1, F2, F3: 85.48%, 81.33%, and 73.86%, respectively. Low accuracies for were obtained for F4 and F5: 60.17%, and 62.24%, respectively. They estimated accuracy using leave-one-out cross-validation.

4.2 Materials

The materials we use were collected and used in [13]. They consist of 320 interneurons and the terminological choices regarding those neurons taken by 42 expert neuroscientists. The pool includes interneurons from different areas and layers of the cerebral cortex of the mouse, rat, rabbit, cat, monkey and human.

241 of the cells were retrieved from the Neuromorpho.org [4] website and their three-dimensional digital reconstructions are available. We used those reconstructions to obtain quantitative data about neuronal morphologies. With this data we train supervised classifiers that can classify a neuron on the basis of its morphological characteristics. The remaining 79 cells were obtained by scanning drawings from published papers. We use these cells, along with the 241 digitally reconstructed ones, to study if a neuron can be automatically classified into the correct interneuron type using its simple axonal features as predictors.

4.2.1 Morphometric Parameters

We computed the following parameters for both the axonal and the dendritic arbors:

- Branched structure analysis. Number of endings, number of nodes (branching points), total and mean dendritic tree length; total, mean, median and standard deviation of segments' length. The maximum number of segments in a path between a root and an ending.

- Convex hull analysis. Area and perimeter of the 2D convex hull; volume and surface of the 3D convex hull.

- Sholl analysis. The number of intersections of the processes with concentric spheres centred at the soma. The radii of the spheres range from 60 μm to 960 μm for the axon and 300 μm for the dendrites, increasing by 60 μm for each next enclosing sphere. In addition to the intersections, we measure the number of endings, number of the nodes and the length of the arbor within a sphere but outside its inner sphere.

- Fractal analysis. The fractal dimension computed using the box-counting method [44].
- Vertex analysis. The number of nodes of each of the following three types: V_a (nodes whose both child segments are terminal); V_b (nodes with one terminal child segment); and V_c (nodes with no terminal child segments). Also the ratio $\frac{V_a}{V_b}$.
- Branch angle analysis. The mean, standard deviation, and median of planar, local and spline angles of the bifurcations.
- Fan-in analysis. Torsion ratio (a measure of length loss upon fan-in analysis).
- Polar histogram. Length according to direction of growth, measured for each of the 36 10-degree-wide angle intervals between 0 and 360 degrees.

Additionally, two more parameters were measured for the dendrites: the number of segments rooted at the soma and the number of branching points in which these segments end. All together, we quantified the axon with 128 parameters and the dendrites with 86, giving a total of 214 morphometric parameters.

Axonal Features F1 and F2

An expert identified a subset of the 214 morphometric parameters as suitable predictors for the simple axonal features F1 and F2. The quantities correspond 57 metrics that describe the axon: axonal length, number of endings, mean segment length, torsion ratio, 2D and 3D convex hull measures, Sholl analysis of intersections, and polar histogram analysis.

4.2.2 Data Preprocessing

Prior to computing morphometric measurements, we noticed that the digital reconstructions of several cells had more than one axon, a situation impossible in practice as neurons have a single axon. In most cases the cause appeared to be the omission of one or more axonal fragments from the reconstruction, which caused the "splitting" of the axon into disconnected parts. For 36 such cases, an expert deemed the missing fragments as easily restorable and draw them manually using the Neurolucida (MicroBrightField) software. For 4 interneurons, the missing axonal fragments were considered either too large or too many and those cells were subsequently omitted from analyses involving morphometric data. This effectively reduced our data set of valid digitally reconstructed cells to 237 instances.

4.3 Methodology

4.3.1 Discretization

The axonal morphometric measurements that we use as predictor variables are real-valued. On the other hand, our Bayesian network classifiers assume discrete predictive variables. Learning such classifiers from morphometric data requires that continuous predictors first be converted to categorical ones.

The process of transforming quantitative data to qualitative (i.e. categorical) data is known as discretization [62]. It consists in the partitioning a continuous domain into a finite number of non-overlapping intervals that are treated as categorical values. It can reduce noise if it is present in the data ([23]) and for the naive Bayes it often yields better accuracy than when normal distribution of predictors is assumed ([14]).

We use the Weighted Proportional k-Interval Discretization (WPKID) method ([61]) for discretization. WPKID adapts the better-known PKID discretization method ([60]) for small data sets. A recent study ([23]), which did not consider WPKID, found PKID to be the best discretization method for naive Bayes in terms of predictive accuracy and Cohen's kappa.

The WPKID is a variant of equal-frequency discretization (see e.g., [14]) where the the number of bins (intervals) is a function of data set size:

$$sb = N$$
$$s = b + 30$$

where b is the number of bins and N the size of the data set.

We do not discretize all of our data as a preprocessing step, that is, prior to classifier induction and evaluation. Instead, we only discretize the training data, mapping the real-valued test data to the intervals formed on training data.

4.3.2 Thresholded Majority Vote Label

A problem domain in supervised learning is modelled with a vector of n predictive variables $\mathbf{X} = (X_1, \ldots, X_n)$ and a class variable $C \in \Omega_c = \{1, \ldots, r_c\}$. We have a data set with N instances $\mathcal{D}_v = \{(\mathbf{x}^{(1)}, \mathbf{v}^{(1)}), \ldots, (\mathbf{x}^{(N)}, \mathbf{v}^{(N)})\}$. The true class label of the instances in $\mathcal{D}_v$ is unknown. Instead, some of M experts provide a single class label for each instance $\mathbf{x}^{(i)}$ in $\mathcal{D}_v$. This is encoded this with a vector of vote counts for each label: $\mathbf{v}^{(i)} = (v_1^{(i)}, \ldots, v_{r_c}^{(i)}) \in \{0, \ldots, M\}^{r_c}$.

We form a data set $\mathcal{D} = \{(\mathbf{x}^{(1)}, c^{(1)}), \ldots, (\mathbf{x}^{(N)}, v^{(N)})\}$, with a single class label per instance, by choosing the *single* most voted class for each instance in $\mathcal{D}_v$ (if there

is a tie for the most voted class then we exclude the instance from $\mathcal{D}$):

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, c^{(i)} = r_k), \quad \forall i : v_k^{(i)} > v_j^{(i)}, j \neq k\}.$$

We can then form subsets of $\mathcal{D}$ such that all data instances have a minimum of label reliability. We define label reliability as the number of voters (experts) that selected the most common label, $v_{c^{(i)}}^{(i)}$, of a data instance $(\mathbf{x}^{(i)}, c^{(i)})$. Then, we form a subset $\mathcal{D}^t$, where t is the label reliability, as

$$\mathcal{D}^t = \{(\mathbf{x}^{(i)}, c^{(i)}), \quad \forall i : v_{c^{(i)}}^{(i)} > t\}.$$

When discussing a data subset $\mathcal{D}^t$ we will refer to t as the *vote threshold* of $\mathcal{D}^t$.

Thresholded Majority for Several Axonal Features

When using all axonal features as predictive variables, we need a single value for each of them. Axonal feature F4 is not applicable to all cells (only to translaminar and displaced cells) and imposing the same threshold on F4 as on other axonal features would omit cells to which F4 is not applicable from our data set. The reason behind this is that an expert only categorizes a cell according to an axonal feature when he/she finds that feature to be applicable. Therefore, even when most experts think that F4 is not applicable to an interneuron, its most voted category will be either ascending, descending, or both (provided they have at least one vote). The number votes for the ascribed label will be low (because the majority did not vote) and the cell will be discarded at a high vote threshold. We avoid discarding such cells by making explicit the opinions of experts who consider F4 as inapplicable. We replace their missing votes with votes for a dummy category.

4.4 Experimental Evaluation

4.4.1 Classifier Parameters and Predictive Performance Evaluation

For the learning algorithms which maximize cross-validation estimate of accuracy (the FSS and BSEJ algorithms), we set the number of folds to $k = 5$, and use a single run of stratified cross-validation (using several repetitions can be very costly). For the APNBC algorithm, we set the α parameter to $\alpha = 1$, so that any improvement in resubtitution accuracy is considered as unlikely to have been obtained by chance (this is the same as not performing the test). For all statistical tests, except that of the APNBC, we use $\alpha = 0.05$ as threshold. For the AWNB algorithm we use

bootstrap samples of size $\frac{N}{2}$ and create 20 such samples. We estimate the parameters of the Bayesian networks the Laplace correction for maximum likelihood. We used 5 repetitions of 5-fold stratified cross-validation to estimate predictive performance.

4.4.2 Predicting Axonal Features Independently

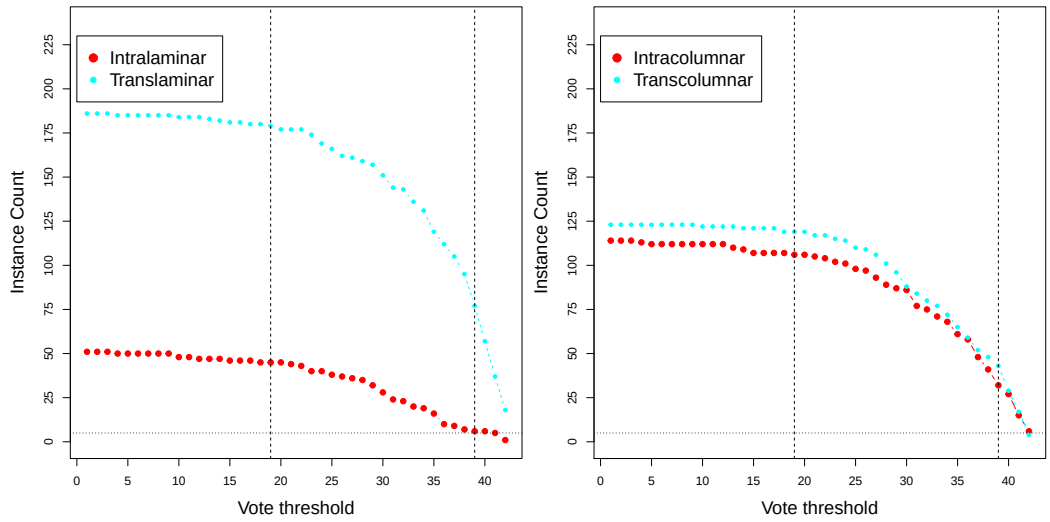
Vote Thresholds

Vote thresholds can be in the range between 0 and 41. Choosing a low threshold would be similar to applying the majority vote rule (no threshold), while a high one might leave us with too few cases. The number of instances available at some threshold t differs among the axonal features (see Figure 4.1) so we consider different vote threshold ranges for different features. As the lower bound on a threshold range for an axonal feature we choose some low threshold which produces a data set $\mathcal{D}^t$ that is roughly different from the full data set. We choose the upper bound rather arbitrarily but impose the condition that it yields at least five instances of at least two different classes. For axonal features F1, F2, and F3, we use the range of 19 to 39. As Figure 4.1 shows, a threshold $t < 19$ would produce a data set similar to the full data set, and a $t > 39$ would leave us with less than 5 `intralaminar` cells. For axonal feature F4 we use the range 11 to 39. For axonal feature F5 we use the range 13 to 27. We can see that for $t < 13$ there is practically no difference with the full data set. At threshold $\mathcal{D}^{27}$ there are 5 `CB`, 1 `CH`, 1 CT, 3 `HT`, 10 `LB` and 20 `MA` cells.

The above-mentioned ranges contain 131 different thresholds. That would produce 131 data sets. On each we would evaluate 10 classifiers with repeated cross-validation, which can be time consuming. In order to reduce the runtime of the experiments, we only use every second threshold in a range.

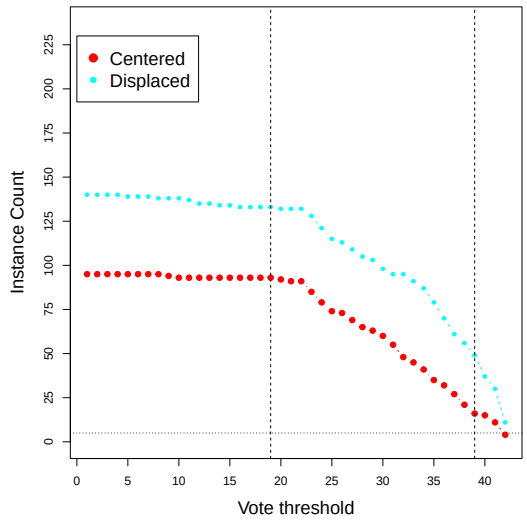
Interneuron Type

The best predictive accuracy is achieved by TAN (see Figure 4.2e) at vote threshold 25 and it equals 77%. At this threshold there are 22 MA, 12 LB, 11 CB, 4 HT, 1 CH, and 1 CT cells. To further analyse the discrimination among classes, we compute the sensitivity and specificity for each class. First, we estimate TAN's confusion matrix by averaging its confusion matrices over 5 runs of 5-fold cross-validation. Then, we compute the sensitivities and the specificities from the averaged confusion matrix. We can see that MA, HT, and CB have relatively high sensitivity and specificity (all above 0.80, see Table (4.1)). LB's sensitivity is somewhat lower but still relatively high. The sensitivity of CH and CT is 0 because there is a single instance of each of those classes.

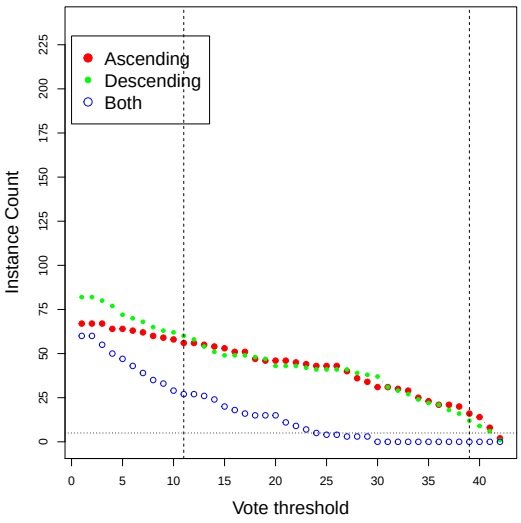


(a) F1

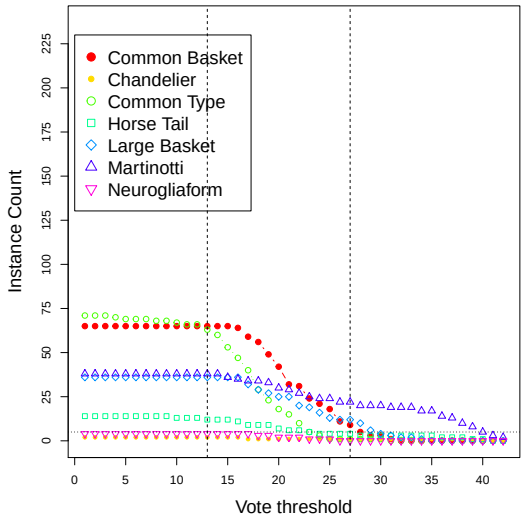
(b) F2



(c) F3



(d) F4.



(e) F5

Figure 4.1: Class count per vote threshold. The dotted horizontal line is at $y = 5$. The dashed horizontal lines indicate the boundaries of vote threshold ranges.

Table 4.1: Sensitivity and specificity of the TAN classifier for predicting interneuron type using morphometric parameters as predictors. Vote threshold 25 is applied to F5. Interneuron type name abbreviations: CB = common basket, CH = chandelier, HT = horse-tail, LB = large basket, MA = Martinotti.

	CB	CH	CT	HT	LB	MA
Sensitivity	0.8000	0.0000	0.0000	0.800	0.6800	0.8400
Specificity	0.9143	0.9795	0.9846	1.000	0.8800	0.9000

Simple Axonal Features

Axonal feature F1 is unbalanced – there are much more translaminar than intralaminar cells (see Figure 4.1a). Therefore, several classifiers seem to be achieving high accuracy by always predicting the translaminar class, as suggested by their 0 kappa index (see e.g. the FSS classifier on threshold 37, Figure 4.2a and Figure 4.3a). The APBNC classifier achieves the highest accuracy (93.92%), and its kappa (0.55) indicates that it can correctly identify some of the intralaminar cells. The highest accuracy is achieved at threshold 39, with 77 translaminar and 6 intralaminar cells.

For the axonal feature F2, the best accuracy (94.02%) is achieved by the FSS classifier on threshold 39 (see Figure 4.1b), where there are 27 intralaminar and 29 transcolumnar cells. The estimated kappa is 0.88.

For the axonal feature F3, the best accuracy (92.98%) is achieved by the APNBC classifier on threshold 39 (see Figure 4.1c). On this threshold, there are 15 centered and 37 displaced cells. The kappa is 0.83.

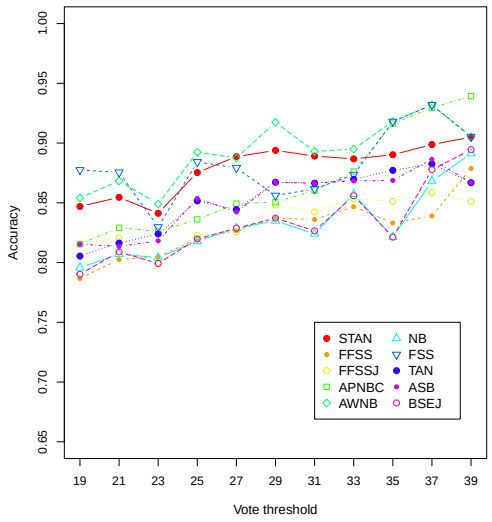
For the axonal feature F4 the highest accuracy is 87.60%, reached by the TAN at threshold 35 (see Figure 4.1d). At this threshold there are 21 ascending and 21 descending cells (no both cells). The sensitivity and specificity of the model are both 0.88. The kappa is 0.71.

4.4.3 Simple Axonal Features as Predictors of Interneuron Type

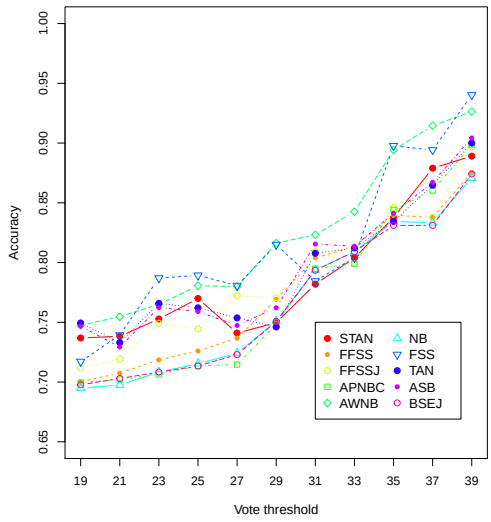
It is interesting to know whether simple axonal features are useful for predicting neuronal type name. If they are, then it might be possible to improve automatic classification of interneurons by predicting simple axonal features from data and using those predictions to (help) discriminate among interneuron types.

We use a single predictor variable for each simple axonal feature. We assign values to these variables using the thresholded majority vote (see Section 4.3.2 and Section 4.3.2).

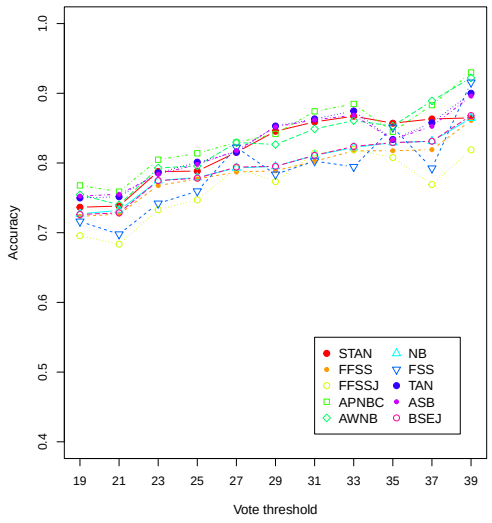
Since are we only using experts' terminological choices to build the classifiers, we



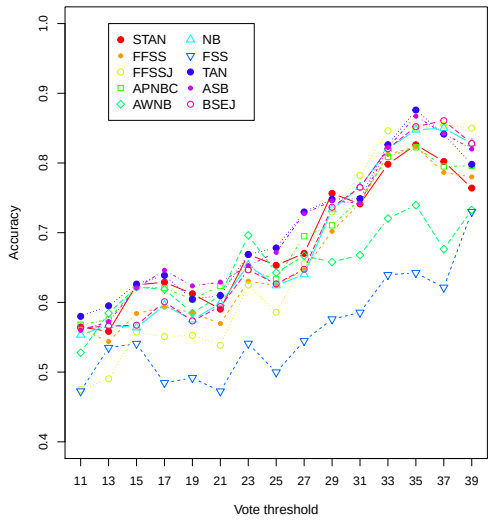
(a) F1



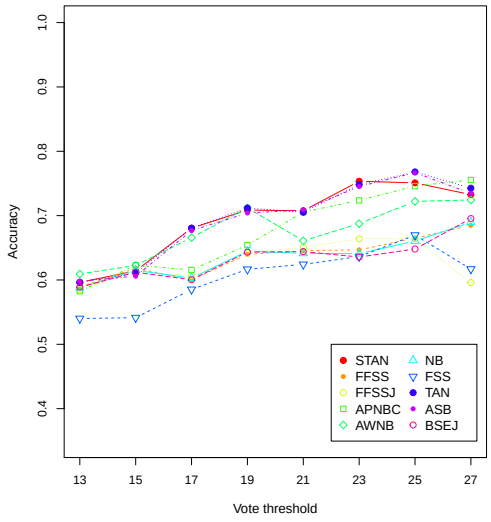
(b) F2



(c) F3

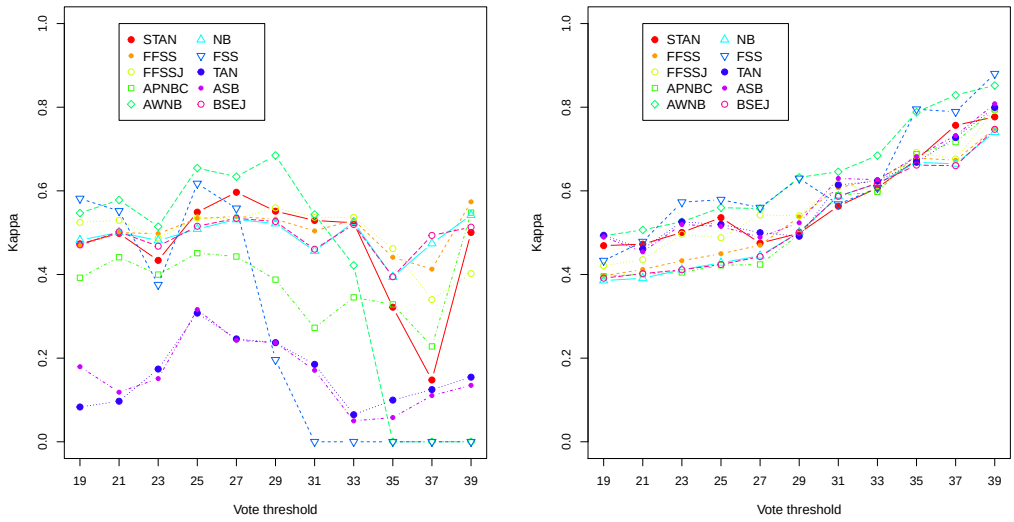


(d) F4



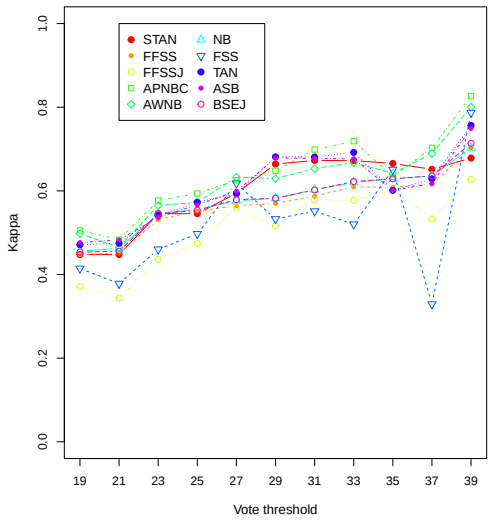
(e) F5

Figure 4.2: Classifiers' accuracy for predicting axonal features individually from morphological data. Note that the minimum value on the y-axis and the range of

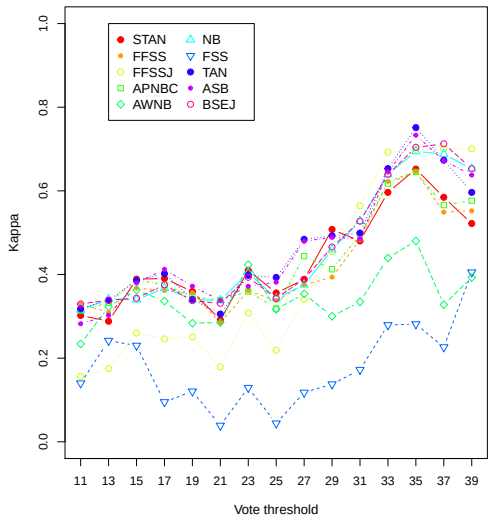


(a) F1

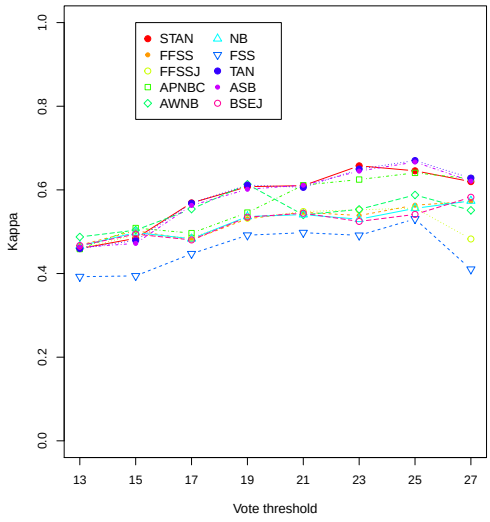
(b) F2



(c) F3



(d) F4



(e) F5

Figure 4.3: Classifiers' kappa for predicting axonal features individually from morphological data. Note that the range of the x-axis can differ between graphs.

may include in our data sample the 79 neurons scanned from publications and the 4 cells with poor digital reconstructions (this corresponds to the full data sample). This gives us a data sample notably larger than the one corresponding to (correctly) digitally reconstructed, analysed in the previous section. We will consider the full data set and the data subset corresponding to digitally reconstructed cells.

We impose a vote threshold on all five axonal features. We arbitrarily choose to fix 21 as vote threshold for the simple axonal features while varying the vote threshold for the neuronal type within the range used when predicting interneuron type from morphological data: 13 to 27 (see Figure 4.4 for class counts).

Full Data Set

Best accuracy is at threshold 27, by TAN, and it is 79.46% (see Figure 4.5a). MA, HT, and NG types are identified with maximum sensitivity (see Table 4.2), LB with medium sensitivity and, unlike when learning from morphological data, CB with relatively low sensitivity (0.36). CB cells were mostly misclassified as NG.

Table 4.2: Sensitivity and specificity of the TAN classifier for predicting interneuron type using simple axonal features as predictors. Both 2D and 3D cells are used, with threshold 21 for simple axonal features and threshold 27 for interneuron type. Interneuron type name abbreviations: CB = common basket, CH = chandelier, CT = common type, HT = horse-tail, LB = large basket, MA = Martinotti, NG = neurogliaform.

	CB	CH	CT	HT	LB	MA	NG
Sensitivity	0.36	0.4	0	1	0.63	1	1
Specificity	0.97	0.99	1	0.97	0.97	0.97	0.88

Digitally Reconstructed Cells

Best accuracy, 88.10%, is achieved by the FSS at threshold 25 for interneuron type (see Figure 4.5c). At this threshold there are 21 MA, 11 LB, 9 CB, 4 HT, 1 CH, and 1 CT cells. FSS can correctly identify MA, HT, LB, and, to a lesser degree, CB cells (see Table 4.3). This data set contains all but 4 of the instances from the data set for which the best prediction on F5 given morphological parameters were obtained (see Section (4.4.2). The accuracy is much higher in this case, reaching 88.10%, as opposed to 77% when using morphological variables as predictors.

Table 4.3: Sensitivity and specificity of the FSS classifier for predicting interneuron type using simple axonal features as predictors. Only 3D cells are used, with threshold 21 for simple axonal features and threshold 25 for interneuron type. Neuronal type names abbreviations: CB = common basket, CHND = chandelier, CT = common type, HT = horse-tail, LB = large basket, MA = Martinotti, NRG = neurogliaform.

	CB	CHND	CT	HT	LB	MA
Sensitivity	0.71	0	0	1	0.93	1
Specificity	0.97	1	1	0.97	0.93	0.99

Simple Axonal Features and Morphometric Measurements as Predictors

Inspecting the data showed that there are many instances with identical simple axonal feature values belonging to different neuronal types. This suggests that a more refined feature space could improve prediction. We therefore augment the feature set with the 214 morphometric parameters. We use the same vote thresholds as previously: in the range from 13 to 27 for interneuron type and vote threshold 21 for simple axonal features.

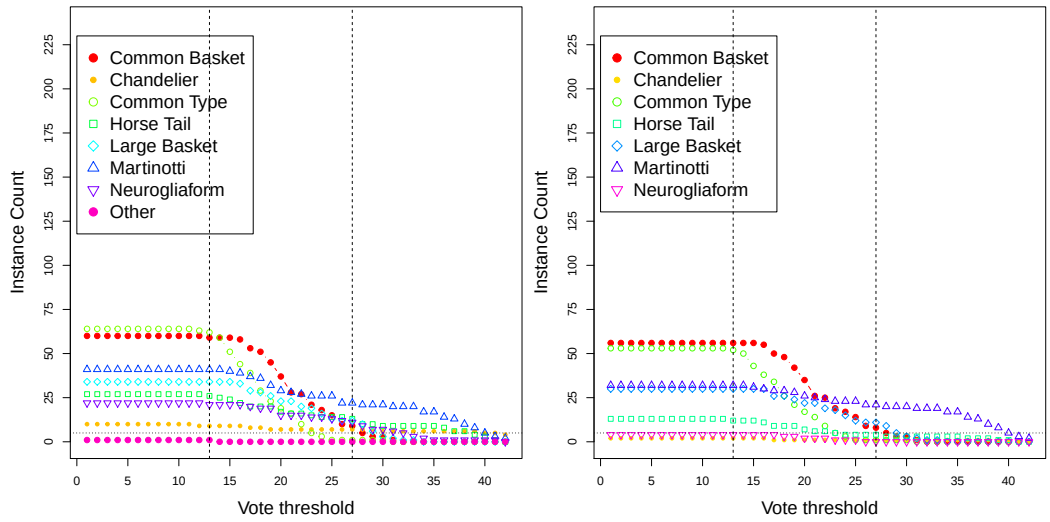
Best predictive performance, 87.86%, is achieved by AWNB at vote threshold 25 (see Figure 4.5e). This is very similar to the best performance when using only simple axonal features as predictors. The sensitivities and specificities also very are similar (see Table 4.4). Using morphometric parameters together with simple features seems to improve accuracy on low thresholds while at high thresholds it degrades the performance of most classifiers (compare Figure 4.5e and Figure 4.5c).

Table 4.4: Sensitivity and specificity of the AWNB classifier for predicting interneuron type using simple axonal features and morphometric parameters as predictors. Vote threshold for the simple axonal features is 21 and for the interneuron type it is 25. Neuronal type names abbreviations: CB = common basket, CH = chandelier, CT = common type, HT = horse-tail, LB = large basket, MA = Martinotti.

	CB	CH	CT	HT	LB	MA
Sensitivity	0.93	0	0	0.7	0.93	0.99
Specificity	0.95	1	1	1	0.98	0.92

4.5 Concluding Remarks

Previous study of supervised classification of interneurons [13] achieved low accuracy for features F5 and F4. Although our results are not directly comparable, as they



(a) Full Data Set (b) Digitally Reconstructed Cells

Figure 4.4: Class count per vote threshold on F5 with vote threshold 21 for F1-F4. The dotted horizontal line is at $y = 5$. The dashed horizontal lines indicate the boundaries of vote threshold ranges. For a), the threshold for simple axonal features is 25 and for b) it is 21.

were obtained from different subsets of the data set used in [13], they show that using reliable cells can improve accuracy.

For the interneuron type, our model can accurately discriminate between reliable examples of the MA, LB, CB, and HT neuronal types. MA and HT are well-defined and therefore useful neuronal types while the usefulness of common and large basket cell types is questioned as these two types are commonly confused by the experts [13]. However, both types have "typical", easily recognizable examples: cells that are assigned to them with high confidence.

Our model could be used to resolve the confusion. As shown, it can correctly discern between the typical (reliable) instances of the CB and LB types. When presented with an ambiguous cell, the model would probably assign it to either LB or CB. Whatever choice the model makes, it will be based on the opinions of many experts.

For feature F4, our model can discriminate between reliable examples of ascending and descending cells. As with interneuron type, this model can be used to resolve the confusion present among experts by ascribing all cells to either the ascending or the descending class. Furthermore, we show that axonal features F1, F2, F3, F4 are can discriminate between interneuron types. Additionally, we show that using simple axonal features together with morphometric parameters helps discriminating

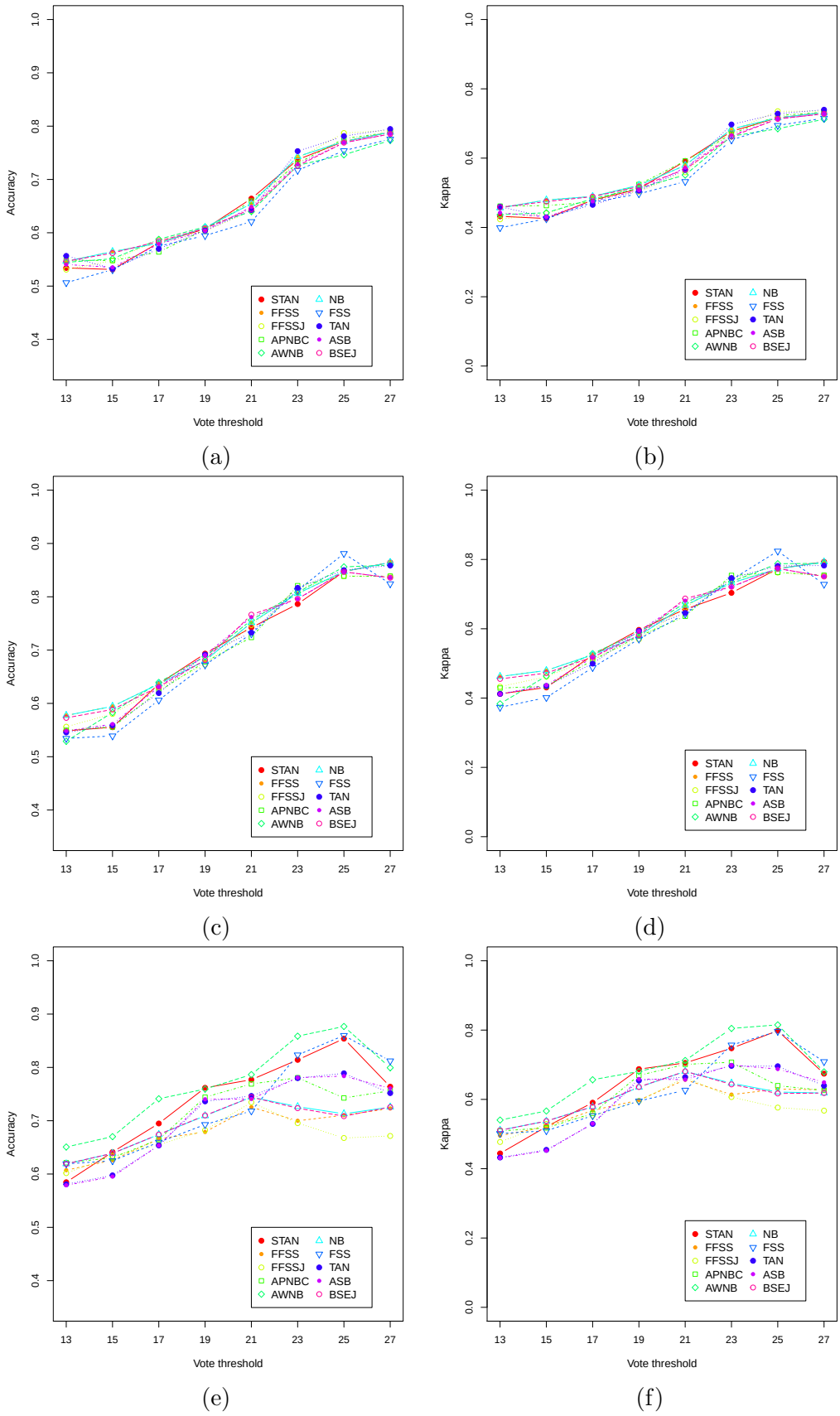


Figure 4.5: Accuracy and Cohen's kappa for predicting interneuron type with simple axonal features as predictors. a) and b) are results obtained with the full data

between interneuron types.

Future work involves the use of predicted values for axonal features F1, F2, F3, and F4 to predict the interneuron type.

Chapter 5

Conclusions

The contributions of this Master's Thesis are three-fold. The first contribution is the implementation of state-of-the-art Bayesian network classifiers for the R environment for statistical computing, through the `bayesclass` add-on package. This package has been submitted to the CRAN repository and is awaiting maintainer's approval for publication. It has been presented with a talk entitled '`bayesclass`: a package for learning Bayesian network classifiers' at the ninth R User Conference, on July 11th 2013, in Albacete, Spain (see http://www.edii.uclm.es/~useR-2013/docs/useR2013_abstract_booklet.pdf).

The second contribution is methodological: the proposal of a novel, promising classifier; the augmented semi-naive Bayes. It will be presented with a talk at the fifteenth Conference of the Spanish Association for Artificial Intelligence, to be held in Madrid, Spain, in September, 2013. The corresponding article is accepted for publication in the associated volume of Lecture Notes in Computer Science. Find the bibliographical information here [46].

The third contribution is the application of the developed tools, both the `bayesclass` package and the augmented semi-naive Bayes classifier, towards solving a challenging problem in neuroscience. Promising results are obtained, indicating that the implemented algorithms and employed methodology are suitable to the task of automatic classification of interneurons.

5.1 Future Work

We plan to keep extending the `bayesclass` package. More discrete Bayesian network classifiers are likely to be added in the future. We plan to implement Bayesian network classifiers which can handle real-valued features and also to allow more flexible prior probabilities for the parameters.

Regarding the augmented semi-naive Bayes classifier, further experiments ought to be carried out to determine if there is statistically significant improvement over the competing algorithms.

Regarding the classification of GABAergic interneurons, future work involves the use of predicted values for axonal features F1, F2, F3, and F4 to predict the interneuron type.

Appendix A

`bayesclass` Package Documentation

The following is the package documentation for `bayesclass`, produced by typing `help(package='bayesclass')` in `R`. It provides basic information about the package and lists the implemented functions.

```
Information on package 'bayesclass'

Description:

Package:       bayesclass
Title:         Algorithms for learning Bayesian network classifiers
Description:
Version:       0.1
Authors@R:     c(person("Mihaljevic","Bojan",email="b.mihaljevic@fi.upm.es",role=c("aut","cre")),
               person("Bielza","Concha",email="mcbielza@fi.upm.es",role="aut"),
               person("Larra<f1>aga","Pedro",email="pedro.larranaga@fi.upm.es",role="aut"))
Depends:       R (>= 2.1.0),gRain,gRbase(>= 1.6.5),graph,RBGL,Rgraphviz,caret,e1071
License:       MIT
Collate:       'ci_tests.R' 'classifiers.R' 'information_theory.R' 'predict.R' 'package_doc.R'
               'data_structures.R' 'caret_integration.R' 'search.R' 'assessment.R' 'selective.R' 'tan.R'
               'graph.R'
Packaged:      2013-07-03 13:06:28 UTC; CIG
Author:        Mihaljevic Bojan [aut, cre], Bielza Concha [aut], Larra<f1>aga Pedro [aut]
Maintainer:    Mihaljevic Bojan <b.mihaljevic@fi.upm.es>
Built:         R 3.0.0; ; 2013-07-03 14:06:33 UTC; windows

Index:

assess                 Uses cross-validation to assess a set of
                       Bayesian network classifiers
assess_model           Uses cross-validation to assesses a specific
                       Bayesian network classifier
assess_params          Uses cross-validation to assess Bayesian
                       network classifiers on a data set
assess_search_state    A wrapper for 'assess_model'
assess_simple          Returns the classifier's accuracy on a data set
assessment_args        Returns a list of arguments for 'assess_model'
augment_nb             Augments the given network with arcs from the
```

```
                        class node to all others
augmenting_forest       Returns the undirected augmenting forest for a
                        tree augmented naive Bayes.
augmenting_tree         Returns the undirected augmenting tree
bayesclass              Returns a bayesclass object
bayesclass_fit          Learns the parameters of a Bayesian network.
bnc                     Learns a Bayesian network classifier from data
bnc_args                Extract bnc arguments from a named vector
caret_model             Induces a Bayesian network classifier and
                        (optionally) performs data pre-processing
caret_parameters        Returns a data frame with parameters of
                        Bayesian network classifier learners
chisq_cond_independence_test
                        Chi-squared tests of conditional independence
chisq_independence_test.mi
                        Chi-squared independence test based on mutual
                        information (the G2 statistic)
chisq_independence_test.x2
                        Chi-squared independence test based on
                        Pearson's X2 statistic
class_cond_dependencies
                        Returns a graph of pairwise class-conditionally
                        dependent features.
complete_graph          Returns a complete unweighted graph with the
                        given nodes
cond_independence_test
                        Chi-squared tests of conditional independence
cond_independence_test_cond_mi
                        Conditional independence test based on
                        conditional mutual information
conditional_mi          Returns the conditional mutual information
                        I(X;Y|Z).
direct_away             Directs edges away from node 'r'
direct_graph            Directs an undirected graph
directed_acyclic_graph  Directs an undirected graph.
extract_parameters      Extracts model parameters from a list of
                        bayesclass objects
features                Returns the features of a Bayesian network
                        classifier
ffss                    Returns a selective naive Bayes with the
                        features for which the hypothesis of
                        independence is rejected using the chi-squared
                        test of independence.
fss                     Uses the forward sequential selection algorithm
                        the learn a selective naive Bayes
get_graph_weights       Returns graph weights
greedy_search           Performs a greedy search in the space of
                        Bayesian network structures
inclusions              Returns the possible search states obtained by
                        including a single candidate in the model
independence_test       Chi-squared test of independence for sets of
                        variables
learner_function        Returns the function that implements the
                        learning algorithm.
lists_to_dataframe      Converts a list of lists into a data.frame
max_likelihood_tree     Returns the undirected augmenting tree
mi                      Returns the mutual information between two
                        variable sets
naiveBayes              Returns the independence network of a naive
                        Bayes
nvalues                 Returns the number of combinations of the given
```

```
                        variables
pairwise_conditional_mi
                        Returns the graph with feature pairwise
                        class-conditional mutual information
predict.BayesClassifier
                        Return class predictions or the class posterior
                        distribution
predict.bayesclass_fit
                        Return class predictions or the class posterior
                        distribution
revert_arcs             Reverts the arcs of a dag
search_state            Returns a search state
set_graph_weights       Adds weights to graph edges
stan_pairwise_class_conditional_mi
                        Returns the filtered pairwise class conditional
                        mutual information network
stanbc                  Returns the selective tree augmented naive
                        Bayes (Blanco et al., 2005).
tan                     Returns the independence network of a tree
                        augmented naive Bayes
tan_structure           Remove edges for interactions that we are not
                        confident about
tanbc                   Returns the independence network of a tree
                        augmented naive Bayes
tree_augmented          Returns the independence network of a tree
                        augmented naive Bayes
tree_augmented_impl     Returns a tree augmented naive Bayes.
valid_chisq_approximation
                        Check whether the chi-squared approximation is
                        reliable
valid_chisq_approximation.expected
                        Check whether Cochrane's conditions for the
                        reliability of the chi-squared approximation
                        are met
vector_list_2_dataframe
                        Converts a list of vectors into a data.frame
weight_conditional_mi   Sets edge weights to the class-conditional
                        mutual information between feature pairs
```

Bibliography

[1] A. Agresti. *Categorical Data Analysis*. Wiley, 1990.

[2] K.M. Al-Aidaroos, A.A. Bakar, and Z. Othman. Naive Bayes variants in classification learning. In *Proceedings of the International Conference on Information Retrieval Knowledge Management (CAMP-2010)*, pages 276–281, 2010.

[3] Ethem Alpaydin. *Introduction to machine learning*. The MIT Press, 2004.

[4] Giorgio A Ascoli, Duncan E Donohue, and Maryam Halavi. Neuromorpho. org: a central resource for neuronal morphologies. *The Journal of Neuroscience*, 27 (35):9247–9251, 2007.

[5] Giorgio A Ascoli, Lidia Alonso-Nanclares, Stewart A Anderson, German Barrionuevo, Ruth Benavides-Piccione, Andreas Burkhalter, György Buzsáki, Bruno Cauli, Javier Defelipe, Alfonso Fairén, et al. Petilla terminology: nomenclature of features of gabaergic interneurons of the cerebral cortex. *Nature Reviews Neuroscience*, 9(7):557–568, 2008.

[6] K. Bache and M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml. http://archive.ics.uci.edu/ml.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2007. ISBN 0387310738. URL http://www.amazon.com/Pattern-Recognition-Learning-Information-Statistics/dp/0387310738%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0387310738.

[8] R. Blanco, I. Inza, M. Merino, J. Quiroga, and P. Larrañaga. Feature selection in Bayesian classifiers for the prognosis of survival of cirrhotic patients treated with TIPS. *Journal of Biomedical Informatics*, 38(5):376–388, 2005.

[9] Remco Bouckaert. *Bayesian network classifiers in Weka*, 2004.

[10] Vince Carey, Li Long, and R. Gentleman. *RBGL: An interface to the BOOST graph library*. URL http://www.bioconductor.org. R package version 1.36.2.

[11] D.M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

[12] C.K. Chow and C.N. Liu. Approximating discrete probability distrivutions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[13] J. DeFelipe, P.L. Lopez-Cruz, R. Benavides-Piccione, C. Bielza, P. Larrañaga, and et al. New insights into the classification and nomenclature of cortical gabaergic interneurons. *Nature Reviews Neuroscience*, page in press, 2013. URL http://dx.doi.org/10.1038/nrn3444.

[14] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *Machine Learning: Proceedings of the Twelfth International Conference*, pages 194–202, 1995.

[15] R. Duda, P. Hart, and D.G. Stork. *Pattern Classification*. John Wiley and Sons, 2001.

[16] Brian Sidney Everitt. *A handbook of statistical analyses using R*. CRC Press, 2010.

[17] U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-1993)*, pages 1022–1029. Morgan Kaufmann, 1993.

[18] M. Julia Flores, José A. Gámez, and Ana M. Martínez. Supervised Classification with Bayesian Networks: A Review on Models and Applications, 2012.

[19] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32 (200):675–701, 1937.

[20] Milton Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.

[21] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

[22] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044–2064, 2010.

[23] Salvador Garcia, Julian Luengo, Jose Antonio Saez, Victoria Lopez, and Francisco Herrera. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *Knowledge and Data Engineering, IEEE Transactions on*, 25(4):734–750, 2013. ISSN 1041-4347. doi: 10.1109/TKDE.2012.35.

[24] R. Gentleman, Elizabeth Whalen, W. Huber, and S. Falcon. *graph: A package to handle graph data structures*. R package version 1.34.0.

[25] Jeff Gentry, Li Long, Robert Gentleman, Seth Falcon, Florian Hahne, Deepayan Sarkar, and Kasper Daniel Hansen. *Rgraphviz: Provides plotting capabilities for R graph objects*. R package version 2.4.1.

[26] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[27] M. Hall. A decision tree-based attribute weighting filter for naive Bayes. *Knowledge-Based Systems*, 20(2):120–126, 2007.

[28] M.A. Hall. *Correlation-based Feature Selection for Machine Learning*. PhD thesis, Department of Computer Science, University of Waikato, 1999.

[29] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009. ISSN 1931-0145. doi: 10.1145/1656274.1656278. URL http://doi.acm.org/10.1145/1656274.1656278.

[30] D. J. Hand and K. Yu. Idiot's Bayes - not so stupid after all? *International Statistical Review*, 69(3):385–398, 2001.

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

[32] Søren Højsgaard. Graphical independence networks with the gRain package for R. *Journal of Statistical Software*, 46(10):1–26, 2012. URL http://www.jstatsoft.org/v46/i10/.

[33] Søren Højsgaard, David Edwards, and Steffen Lauritzen. *Graphical models with R*. Springer, 2012.

[34] Kurt Hornik, Christian Buchta, and Achim Zeileis. Open-source machine learning: R meets Weka. *Computational Statistics*, 24(2):225–232, 2009.

[35] R.L. Iman and J. M. Davenport. Approximations of the critical region of the friedman statistic. *Communications in Statistics*, 9(6):571–595, 1980.

[36] E.J. Keogh and M.J. Pazzani. Learning the structure of augmented Bayesian classifiers. *International Journal on Artificial Intelligence Tools*, 11(4):587–601, 2002.

[37] R. Kohavi. Scaling up the accuracy of Naive-Bayes classifiers: a decision-tree hybrid. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-1996)*, 1996.

[38] R. Kohavi and G.H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273–324, 1997.

[39] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[40] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. 1997.

[41] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48–50, 1956.

[42] Max Kuhn. Building predictive models in r using the caret package. *Journal of Statistical Software*, 28(5):1–26, 2008.

[43] P. Langley and S. Sage. Induction of selective Bayesian classifiers. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence (UAI-1994)*, pages 399–406. Morgan Kaufmann, 1994.

[44] Benoit B Mandelbrot, Dann E Passoja, and Alvin J Paullay. Fractal character of fracture surfaces of metals. 1984.

[45] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2012. URL http://CRAN.R-project.org/package=e1071. R package version 1.6-1.

[46] Bojan Mihaljevic, Pedro Larrañaga, and Concha Bielza. Augmented Semi-naive Bayes Classifier. In *Lecture Notes in Computer Science*, number 8109. Springer, 2013. in press.

[47] M. Minsky. Steps toward artificial intelligence. *Transactions on Institute of Radio Engineers*, 49:8–30, 1961.

[48] Kevin P Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, 2012.

[49] Radhakrishnan Nagarajan, Marco Scutari, and Sophie Lébre. *Bayesian Networks in R with Applications in Systems Biology*. Springer, 2013.

[50] M. Pazzani. Constructive induction of cartesian product attributes. In *Proceedings of the Information, Statistics and Induction in Science Conference (ISIS-1996)*, pages 66–77, 1996.

[51] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, Palo Alto, CA, 1988.

[52] M. Peot. Geometric implications of the naïve Bayes assumption. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-1996)*, pages 414–419. Morgan Kaufmann, 1996.

[53] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL http://www.R-project.org/. ISBN 3-900051-07-0.

[54] Vikas C Raykar, Shipeng Yu, Linda H Zhao, Gerardo Hermosillo Valadez, Charles Florin, Luca Bogoni, and Linda Moy. Learning from crowds. *The Journal of Machine Learning Research*, 99:1297–1322, 2010.

[55] Marko Robnik-Sikonja and Petr Savicky. *CORElearn: CORElearn - classification, regression, feature evaluation and ordinal evaluation*, 2013. URL http://CRAN.R-project.org/package=CORElearn. R package version 0.9.41.

[56] Y. Saeys, I. Inza, and P. Larra naga. A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517, 2007.

[57] M. Sahami. Learning limited dependence Bayesian classifiers. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-1996)*, pages 335–338, 1996.

[58] Marco Scutari. Learning bayesian networks with the bnlearn r package. *arXiv preprint arXiv:0908.3817*, 2009.

[59] G. I. Webb and M. J. Pazzani. Adjusted probability naïve Bayesian induction. In *Proceedings of the 11th Australian Joint Conference on Artificial Intelligence (AI-1998). Lecture Notes in Computer Science*, volume 1502. Springer, 1998.

[60] Ying Yang and Geoffrey Webb. Proportional k-interval discretization for naive-bayes classifiers. *Machine Learning: ECML 2001*, pages 564–575, 2001.

[61] Ying Yang and Geofrey Webb. Weighted proportional k-interval discretization for naive-bayes classifiers. *Advances in Knowledge Discovery and Data Mining*, pages 565–565, 2003.

[62] Ying Yang, Geoffrey I Webb, and Xindong Wu. Discretization methods. In *Data Mining and Knowledge Discovery Handbook*, pages 101–116. Springer, 2010.

[63] Sandeep Yaramakala and Dimitris Margaritis. Speculative Markov blanket discovery for optimal feature selection. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages pp 809–812. IEEE Computer Society, Washington, DC, USA, 2005.

[64] F. Zheng and G.I. Webb. A comparative study of semi-naive Bayes methods in classification learning. In *Proceedings of the 4th Australasian Data Mining Workshop*, pages 141–156. University of Technology of Sydney, 2005.