UNIVERSITY OF CRETE
DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF SCIENCES AND ENGINEERING

Efficient and Accurate Feature Selection, with Extensions for Multiple Solutions and to Big Data

by

Giorgos Borboudakis

PhD Dissertation

Presented

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

Heraklion, February 2019

UNIVERSITY OF CRETE

DEPARTMENT OF COMPUTER SCIENCE

Efficient and Accurate Feature Selection, with Extensions for Multiple Solutions and to Big Data

PhD Dissertation Presented

by **Giorgos Borboudakis**

in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

APPROVED BY:

Author: Giorgos Borboudakis, PhD Candidate, University of Crete

---

Supervisor: Ioannis Tsamardinos, Professor, University of Crete

---

Committee Member: Gregory F. Cooper, Professor, University of Pittsburgh

---

Committee Member: Dimitris Plexousakis, Professor, University of Crete

---

Committee Member: Gavin Brown, Professor, University of Manchester

---

Committee Member: Vassilis Christophides, Professor, University of Crete

---

Committee Member: Isabelle Guyon, Professor, University of Paris-Saclay

---

Committee Member: Jan Lemeire, Professor, Vrije Universiteit Brussel

---

Department Chairman: Angelos Bilas, Professor, University of Crete

Heraklion, February 2019

This thesis is dedicated to my parents Nikos and Stamatia, and my sisters Kelly and Christina.

Acknowledgments

Firstly, I would like to thank my supervisor Prof. Ioannis Tsamardinos for his guidance, patience and support. I owe many things and skills I learned our collaboration, and would definitely not be where I am if it wasn't for him.

Besides my supervisor, I would also like to thank the rest of my thesis committee, Prof. Gregory F. Cooper, Prof. Dimitris Plexousakis, Prof. Vassilis Christophides, Prof. Gavin Brown, Prof. Isabelle Guyon and Prof. Jan Lemeire, for all their helpful comments and suggestions.

I thank my labmates for working with me and for all the fun we had during the last few years. Special thanks to Sofia, Vincenzo, Paul, Michail, and Kleanthi for working closely with me and and for helping me grow as a researcher. I also thank Prof. Froudakis and his team, Prof. Mouchtaris and Tasos Alexandridis, as well as Prof. Christophides, Prof. Pratikakis and Pavlos Katsogridakis for our collaborations.

I am grateful to all of my friends, for all the good and bad times we shared during the last years. The list is in alphabetical order (and most likely non-exhaustive, apologies for that): Alex T., Alexandros P., Alexis B., Alexis S., Amin, Anna, Andreas, Argiro, Athineou, Christos S., Christos T., Danielle, Efi, Flavius, Florin, George P., George T., Giannis Pap., Giannis Pan., Glykeria, Ioulia, Iraklis, Iulia, Jenny, Jordan, Jorge, Klio L., Klio V., Konstantina, Kostas K., Kriton, Krystallia, Laertis, Lampis, Lisa, Maria M., Maria P., Marina, Michalis, Mihai, Milosz, Myrto, Nasos, Olina, Paul, Popi, Quim, Roberta, Serafim, Sofia, Sorina, Stan, Stefanos, Tasos A., Tasos T., Thodoris, Tommy, Vincenzo, Vlad, Xanthos, Zack.

Last, but definitely not least, I thank my family for always being there for me.

Abstract

The problem of feature selection can be defined as identifying a minimal subset of the input variables that is optimally predictive for an outcome variable of interest. Feature selection is a common component in supervised machine learning pipelines, and an essential tool when the goal of the analysis is knowledge discovery. The latter is especially important in domains such as molecular biology and life sciences, where a researcher is interested in understanding the problem under study and not necessarily in the resulting predictive model.

Feature selection is challenging: it has been shown to be NP-hard, and thus most approaches rely on approximations to solve it efficiently. There exist many different approaches to the feature selection problem, trading off generality (e.g., what types of data and outcomes they can handle), computational cost, and theoretical properties of optimality. Stepwise selection methods (e.g., forward-backward selection) are quite general and optimal for a large class of distributions, but are computationally expensive. Sparsity-based methods (e.g., LASSO) are computationally efficient for some problems (e.g., classification and regression) and slow for others (e.g., time-course data), and have strong theoretical guarantees. Information theoretic approaches are computationally fast, but not as general (they only handle discrete data) and with weaker theoretical guarantees. Another challenge is to scale feature selection methods to very large datasets, which may contain millions of samples and variables. Existing approaches are either too slow or perform poorly in terms of predictive performance. Finally, most methods do not deal with the presence of multiple solutions to the feature selection problem, which often exist in real data. For example, it has been shown molecular data often contain multiple solutions, possibly due to the redundancy present in the underlying biological system. Therefore, although identifying a single solution is adequate for predictive purposes, it is not sufficient when the focus is on knowledge discovery. On the contrary, reporting a single solution and claiming that there are no other solutions is misleading.

In this thesis, we focus on forward-backward selection and propose several extensions to tackle the above challenges. Forward-backward selection was chosen because of its theoretical properties and generality, as it is applicable to different predictive tasks and data types. We provide a unified view of several classes of feature selection methods, such as sparsity-based, information theoretic, statistical and causal-based methods, and show that they are instances or approximations of stepwise selection methods. This allows one to translate and use techniques (such as the ones proposed in this thesis) between differ-

ent approaches to the feature selection problem. Then, we propose a heuristic inspired by causal modeling to speed-up the forward-backward selection algorithm, while preserving its theoretical properties. In experiments we show that this leads to a speed-up of 1-2 orders of magnitude over the standard forward-backward selection algorithm, while retaining its predictive performance. Afterwards, we extend the algorithm for Big Data settings, enabling it to scale to data with tens of millions of samples and variables. In a comparison with alternative methods from the same algorithmic family, we show that the proposed method significantly outperforms all competitors in terms of running time, being the only method that is able to terminate on all datasets, and without sacrificing predictive performance. Furthermore, in a comparison with information theoretic methods we show that, although computationally slower, it is able to produce significantly better predictive models. Finally, we deal with the multiple feature selection problem. We show that the existing taxonomy of features is misleading when multiple solutions are present, and propose an alternative taxonomy that takes multiplicity into account. Then, we consider several definitions of statistical equivalence of feature sets, as well as methods to test for equivalence of feature sets. Afterwards, we propose a general strategy to extend forward-backward type methods for identifying multiple, statistically equivalent solutions. We provide conditions under which it is able to identify all equivalent solutions. In a comparison with the only alternative method with the same theoretical guarantees, we show that it produces similar results while being computationally faster.

Περίληψη

Το πρόβλημα της επιλογής μεταβλητών μπορεί να οριστεί ως η ανακάλυψη ενός ελάχιστου υποσυνόλου των μεταβλητών εισόδου που είναι βέλτιστα προβλεπτικό για κάποια μεταβλητή ενδιαφέροντος. Η επιλογή μεταβλητών συνηθίζεται να χρησιμοποιείται σε αναλύσεις μηχανικής μάθησης και είναι βασικό εργαλείο όταν ο στόχος της ανάλυσης είναι η ανακάλυψη γνώσης. Αυτό είναι ιδιαίτερα σημαντικό σε τομείς όπως η μοριακή βιολογία και οι επιστήμες της ζωής, όπου ένας ερευνητής ενδιαφέρεται να κατανοήσει το πρόβλημα που μελετάει και όχι απαραίτητα για το προγνωστικό μοντέλο που προκύπτει.

Η επιλογή μεταβλητών είναι δύσκολη: έχει αποδειχθεί ότι είναι NP-σκληρή, και για αυτό οι περισσότεροι αλγόριθμοι είναι προσεγγιστικοί για να είναι υπολογιστικά αποδοτικοί. Υπάρχουν πολλές διαφορετικές προσεγγίσεις στο πρόβλημα επιλογής μεταβλητών, οι οποίες διαφέρουν στο πόσο γενικές είναι (π.χ. τι τύπους δεδομένων και μεταβλητών μπορούν να χειριστούν), στο υπολογιστικό τους κόστος, καθώς και στις θεωρητικές τους ιδιότητες. Οι μέθοδοι βηματικής επιλογής (stepwise selection) είναι αρκετά γενικές και βέλτιστες για μια μεγάλη κατηγορία πιθανοτικών κατανομών, αλλά είναι υπολογιστικά ακριβές. Οι μέθοδοι που βασίζονται σε αραιότητα (sparsity) (π.χ. LASSO) είναι υπολογιστικά αποδοτικές για ορισμένα προβλήματα (π.χ. ταξινόμηση και παλινδρόμηση) και χρονοβόρες για άλλα, (π.χ. δεδομένα χρόνου) και έχουν ισχυρές θεωρητικές εγγυήσεις. Οι προσεγγίσεις βασισμένες στη θεωρίας πληροφορίας είναι υπολογιστικά γρήγορες, αλλά όχι τόσο γενικές (χειρίζονται μόνο διακριτά δεδομένα) και με ασθενέστερες θεωρητικές εγγυήσεις. Μια άλλη πρόκληση είναι να κλιμακωθούν οι μέθοδοι επιλογής μεταβλητών για δεδομένα μεγάλου όγκου, τα οποία μπορεί να περιέχουν εκατομμύρια δείγματα και μεταβλητές. Οι υπάρχουσες προσεγγίσεις είτε είναι πολύ αργές, είτε έχουν κακή απόδοση ως προς την προβλεπτική τους ικανότητα. Τέλος, οι περισσότερες μέθοδοι δεν λαμβάνουν υπόψιν τους την παρουσία πολλαπλών λύσεων, οι οποίες συχνά υπάρχουν σε πραγματικά δεδομένα. Για παράδειγμα, είναι γνωστό πως τα μοριακά δεδομένα συχνά περιέχουν πολλαπλές λύσεις, πιθανώς λόγω του πλεονασμού που υπάρχει στο υποκείμενο βιολογικό σύστημα. Επομένως, παρόλο που ο εντοπισμός μιας λύσης είναι επαρκής για τον σκοπό της πρόβλεψης, δεν αρκεί για την ανακάλυψη γνώσης. Αντιθέτως, η αναφορά μίας και μόνης λύσης και ο ισχυρισμός πως δεν υπάρχουν άλλες λύσεις είναι παραπλανητική.

Για τη διπλωματική εργασία εστιάζουμε σε άπληστες μεθόδους επιλογής με-

ταβλητών τύπου “forward-backward” και προτείνουμε διάφορες επεκτάσεις για την αντιμετώπιση των παραπάνω προκλήσεων. Επιλέξαμε αυτή την κατηγορία μεθόδων λόγω των θεωρητικών ιδιοτήτων και της γενικότητάς τους. Δείχνουμε πως αλγόριθμοι διαφόρων κατηγοριών, όπως αυτοί που βασίζονται στην αραιότητα, στη θεωρία πληροφορίας, στη στατιστική ή στη θεωρία αιτιότητας, είναι ειδικές περιπτώσεις ή προσεγγίσεις μεθόδων βηματικής επιλογής. Αυτό επιτρέπει την μετάφραση και χρήση τεχνικών (όπως αυτές που προτείνονται σε αυτή τη διατριβή) μεταξύ διαφορετικών κατηγοριών αλγορίθμων. Στη συνέχεια, προτείνουμε ένα ευριστικό, εμπνευσμένο από αιτιατή μοντελοποίηση, για να επιταχύνουμε τον αλγόριθμο επιλογής forward-backward selection , διατηρώντας τις θεωρητικές ιδιότητές του. Σε υπολογιστικά πειράματα δείχνουμε ότι αυτό οδηγεί σε επιτάχυνση 1-2 τάξεων μεγέθους, διατηρώντας παράλληλα την προβλεπτική του ικανότητα. Στη συνέχεια, επεκτείνουμε τον αλγόριθμο για τις δεδομένα μεγάλου όγκου, επιτρέποντάς του να χειριστεί δεδομένα με δεκάδες εκατομμύρια δείγματα και μεταβλητές. Σε μια σύγκριση με αλγορίθμους από την ίδια αλγοριθμική οικογένεια, δείχνουμε ότι η προτεινόμενη μέθοδος περνά σημαντικά τον ανταγωνισμό όσον αφορά το χρόνο λειτουργίας, έχει την ίδια προβλεπτική ικανότητα, και είναι η μόνη μέθοδος που μπορεί να τερματίσει σε όλα τα σύνολα δεδομένων. Επιπλέον, σε μια σύγκριση με μεθόδους βασισμένες στη θεωρίας πληροφορίας, δείχνουμε ότι, αν και υπολογιστικά βραδύτερη, είναι σε θέση να παράγει σημαντικά καλύτερα προγνωστικά μοντέλα. Τέλος, ασχολούμαστε με το πρόβλημα ανακάλυψης πολλαπλών λύσεων. Δείχνουμε ότι η υπάρχουσα ταξινόμηση χαρακτηριστικών είναι παραπλανητική όταν υπάρχουν πολλές λύσεις και προτείνουμε μια εναλλακτική ταξινόμηση που λαμβάνει υπόψη την ύπαρξη πολλαπλών λύσεων. Στη συνέχεια, εξετάζουμε αρκετούς ορισμούς της στατιστικής ισοδυναμίας συνόλων μεταβλητών, καθώς και μεθόδους ελέγχου της ισοδυναμίας συνόλων μεταβλητών. Έπειτα, προτείνουμε μια γενική λύση για την επέκταση των μεθόδων τύπου forward-backward για τον εντοπισμό πολλαπλών, στατιστικά ισοδύναμων λύσεων και παρέχουμε συνθήκες υπό τις οποίες είναι σε θέση να ανακαλύψει όλες τις ισοδύναμες λύσεις. Σε μια σύγκριση με τη μόνη εναλλακτική μέθοδο με τις ίδιες θεωρητικές εγγυήσεις, δείχνουμε ότι παράγει παρόμοια αποτελέσματα ενώ είναι υπολογιστικά ταχύτερη.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 The Feature Selection Problem

In supervised machine learning tasks the goal is to construct a predictive model for a quantity of interest T (also called target or outcome), using a set of predictors (also called features, variables or attributes) $\mathbf{F}$. The process of model construction typically consists of several steps, such as feature extraction and construction (e.g., using dimensionality reduction techniques), data pre-processing (e.g., missing value imputation and standardization), feature selection and model training. Feature selection (also called variable selection) is applied to select a subset of the input features $\mathbf{F}$ for use in model training. The objective of the **feature selection problem** can be defined as *identifying a minimal-size subset of the features that is multivariately optimally predictive for an outcome of interest T* [145]. An introduction to the topic can be found in [63], while a recent survey on feature selection methods is given in [94].

By selecting a subset of the features and removing the remaining ones from consideration, it is often the case that a better model can be learned, especially in high-dimensional settings. This may seem counterintuitive, as an ideal learning algorithm should in principle be able to perform at least as well without applying feature selection, as the information provided by the selected features is already contained in the input data. Indeed, asymptotically (i.e, as sample size tends to infinity) and given a perfect learning algorithm, there is no reason to perform feature selection for predictive purposes. In practice however, by removing irrelevant and redundant features the task of the learning algorithm becomes easier, and thus often leads to better models. This is because a good-quality selection of features facilitates modeling, particularly for algorithms susceptible to the curse of dimensionality.

Another use of feature selection is to reduce the cost of measuring the features to make operational a predictive model. For example, it can reduce the monetary cost and time of measuring the features, or inconvenience to a patient of applying a diagnostic model by reducing the number of medical tests and measurements required to perform on a subject for providing a diagnosis.

In computer science and related fields, the main purpose of feature selection is to aid in the creation of better predictive models, and to reduce the cost of measuring features. We argue however, that often *feature selection is the main objective of an analysis*, and not the learned model. This is particularly true in domains such as molecular biology or life sciences, where the primary goal is to gain understanding and intuition about the problem under study, and *feature selection is used as a tool for knowledge discovery*. This is no accident, as the solution to the feature selection problem has been shown to be directly related to the data-generating causal mechanism [4,86,145]. For example, a medical researcher analyzing their molecular data of cancerous and healthy tissues is not necessarily interested in building a model to classify tissues, as they can already diagnose them on the microscope. Instead, the goal is to identify the features (e.g., gene expressions) that are important for the diagnosis. These gene expressions may improve understanding of the mechanism of the disease, and can lead to the formulation of hypotheses and to further experiments. Similarly, a business executive analyzing subscribers' data, may care more about the features that predict attrition than the actual predictive model. The predictive features may provide understanding into what affects the subscribers' behavior and how they can be influenced through advertisement or promotions.

1.2 Challenges

1.2.1 Hardness of Feature Selection and Approximate Approaches

The feature selection problem is NP-hard[1] [157]. Recently, there have been several approaches to solve the problem exactly (also called the best subset selection problem) for linear models, by formulating it as a mixed integer linear program and solving it using existing off-the-shelf solvers [10,110,130]. Although the results are promising, exact approaches are only able to handle a few hundred or thousand variables at most.

In order to efficiently tackle the problem, most approaches rely on some kind of approximation. The majority of approximate approaches can be roughly categorized into stepwise methods [88,156], sparsity-based methods such as LASSO [139], information-theoretic methods [22] and causal-based methods [4]. Even though they do not solve the exact problem, it can be shown that they still are optimal for a large class of distributions or under certain conditions. For instance, causal-based methods identify the optimal solution (called the Markov blanket of T [86]) in distributions that can be faithfully represented by causal networks (such as Bayesian networks [114,136] and maximal ancestral graphs [126]). Assuming faithfulness of the data distribution has led

[1]The proof is for the decision version of linear regression, and thus the general problem is trivially also NP-hard.

to algorithms that have been proven competitive in practice [4,5,89–91,101,117,146,147]. Furthermore, in a recent comparison between best subset selection, the standard forward selection algorithm (an instance of stepwise methods) and the LASSO, it has been shown that forward selection and best subset selection have almost identical performance, and are competitive with the LASSO [70]. The theoretical properties and empirical evidence encourage further research into approximate methods.

1.2.2 Feature Selection with Big Data

An important problem which has not received much attention, is to devise scalable feature selection methods for Big Data. When applied to Big Data settings, feature selection algorithms need to scale not only to millions of training samples but also millions of variables. Specifically, in the context of Big Data featuring both high dimensionality and/or high sample volume, computations become *CPU-intensive* as well as *data-intensive* and cannot be handled by a single machine; see [13, 14, 163, 166] for the evolution of Big Data dimensionality in various ML datasets. The main challenges arising in this context are (a) how can data be partitioned both horizontally (over samples) and vertically (over features), called *hybrid-partitioning*, so that *computations can be performed locally in each block* and combined globally with a *minimal communication overhead*; (b) what *heuristics* can quickly (e.g., without the need to go through all samples) and safely (providing theoretical guarantees of correctness) eliminate irrelevant and redundant features.

Hybrid partitioning over both data samples and variables [92, 160] is an open research issue in Big ML algorithms, while safe feature selection heuristics has been proposed only for sparse Big Data [121, 135], i.e., for data where a large percentage of values are the same (typically zeros). Most existing feature selection approaches for Big Data mainly focus on high sample volume data, often assume sparsity, and do not handle high dimensionality [163]. Existing parallel feature selection methods fall into three categories: (a) methods that can deal only with large sample sizes and low dimensional data [135], (b) methods that require shared-memory systems, thus limiting the size of data they can handle [19, 95, 167], and (c) methods that can handle all of the above but are overly simplistic or restrictive [104, 121].

1.2.3 Multiple Solutions to the Feature Selection Problem

Another shortcoming of existing feature selection methods is that they arbitrarily seek to identify only one solution to the feature selection problem. In practice, it is often the case that multiple equivalent solutions exist [42, 81, 127, 137, 138]. For instance, in domains such as molecular biology there often exist multiple solutions, possibly because of the inherent redundancy present in the underlying biological system [42,

137]. We argue that, *while a single solution may be acceptable for building a predictive model, it is not sufficient when feature selection is employed for knowledge discovery*. On the contrary, it may be misleading. For example, if several sets of risk factors in a medical study are collectively equally predictive of an event, then it is misleading to return only one of them and claim that the rest are superfluous. Indeed, they are given the selected ones, but it should be noted that there are other solutions that should also be considered. Ideally, a feature selection algorithm should identify all solutions that are "equivalent" in predicting the target (for some reasonable definition of equivalence). Another advantage of outputting multiple solutions is that one could use any of them for building a predictive model. This is especially important if it is very expensive to measure certain variables, and each variable is associated with a cost. The problem where each feature is associated with a cost is called *cost-sensitive feature selection* [71]. Methods returning multiple solutions could solve the cost-sensitive feature selection problem by selecting the lowest-cost feature set among all equivalent solutions, while retaining its predictive ability [138]. Finally, it is important to note that *the problem of multiple feature selection is also directly tied to the stability of feature selection methods*. The stability of a feature selection algorithm is measured by how sensitive its solutions are to different datasets sampled from the same distribution; see [80] for a study of stability of feature selection algorithms, and [112] for methods of measuring stability. We argue that stability has to be defined in the context of multiple solutions instead of a single solution as, by definition, any optimal feature selection algorithm will arbitrarily select one out of all equivalent solutions, rendering measures of stability defined on single solutions meaningless.

There exists only little work on the problem of identifying multiple solutions, and most existing algorithms are based on heuristic approaches with little to no theoretical guarantees. Examples include methods that repeatedly apply a feature selection algorithm with some element of randomness (e.g., by resampling the data) [105, 117], or methods that perform clustering to identify groups of highly correlated features (i.e., trying to identify groups of features that give similar predictive information) [75,84,96]; a more detailed review can be found in [138]. Another class of algorithms includes causal-based methods that try to identify features that give the same information about the target in the context of other variables [89, 138, 144]. Finally, the only algorithm that is able to provably identify all equivalent solutions (under certain conditions) is TIE* [138]. Its drawbacks are that it is computationally demanding, and quite complex to understand and implement efficiently.

1.3 Outline and Main Contributions

For this work, we chose to work on and extend the forward-backward selection (FBS) algorithm, an instance of stepwise methods [88, 156]. These methods are some of the oldest, simplest and most commonly employed feature selection methods. Forward selection and variations of it has re-appeared under different names in the fields of computer science, statistics and signal processing. It has been shown that they are optimal for distributions that can be faithfully represented by causal graphs [101][2]. An attractive property is that it is a general algorithm that can be adapted to handle (a) different learning tasks (e.g., classification, regression, survival analysis), (b) mixed variable types (e.g., continuous and categorical), (c) different data types (e.g., cross-sectional and time-course data), (d) data with linear or non-linear relations, and (e) heteroscedastic data. The only requirement is to use an appropriate conditional independence test for the specific data and analysis task; many of the aforementioned tests, along with others have been implemented in the MXM R package [89].

In Chapter 2 we provide an overview of the notation and terminology used throughout the thesis, and proceed with an introduction to the necessary background knowledge required for the remainder of the thesis. Then, in Chapter 3 we describe the main principles of different approaches to the feature selection problem (stepwise methods, sparsity-based, information-theoretic and causal-based), and show that they can all be seen as stepwise methods or approximations thereof. This further encourages the choice of extending the FBS algorithm, as the techniques introduced can be translated and applied by other classes of feature selection methods. In Chapter 4 we propose the early dropping heuristic to speed-up the forward selection algorithm, in Chapter 5 we extend it to Big Data settings, and in Chapter 6 we show how a large class of greedy forward-backward type algorithms can be extended to return multiple solutions; a more detailed description of our contributions are given below. Finally, Chapter 7 contains several possible future research directions.

A list of publications produced during the course of my studies is given in Appendix E.

1.3.1 Speeding-up Forward Selection: the Forward-Backward Selection with Early Dropping Algorithm

The main drawback of forward selection is its computational cost. In order to select k variables, it performs $O(p \cdot k)$ tests for variable inclusion, where p is the total number of variables in the input data. This is acceptable for low-dimensional datasets, but

[2] It can be shown that it is optimal under weaker assumptions. The only requirement is that the distribution satisfies the local composition property with respect to T [138], which is one of the consequences of faithfulness.

quickly becomes unmanageable with increasing dimensionality. Another issue is that forward selection suffers from the multiple testing problem, resulting in p-values that are too small, and hence the selection of a large number of irrelevant variables [56].

In Chapter 4 we extend the forward selection algorithm to deal with the problems above. We propose a heuristic to reduce its computational cost without sacrificing quality, while also selecting fewer variables and reducing multiple testing issues. The idea is, in each iteration of the forward search, to filter out variables that are deemed conditionally independent of the target given the current set of selected variables. After termination the algorithm is allowed to run up to K additional times, each time initializing the set of selected variables to the ones selected in the previous run. Finally, backward selection is applied on the result of the forward phase. We call this algorithm **Forward-Backward selection with Early Dropping** (FBED^K). This heuristic is inspired by the theory of Bayesian networks and maximal ancestral graphs [126, 136], and similar ideas have been successfully applied by other feature selection methods [4]. We show that FBED^K retains the theoretical properties of FBS. Specifically, we show that members of the FBED^K algorithmic family identify the optimal solution in distributions that are faithful to Bayesian networks or maximal ancestral graphs. In the experimental evaluation we show that the proposed algorithm is 1-2 orders of magnitude faster than FBS, while selecting fewer or the same number variables and having similar predictive performance. In a comparison between different members of the FBED^K family and FBS we show that FBED^0 and FBED^1 also reduce the number of false variable selections, when the data consist of irrelevant variables only. We also investigated the behavior of FBED^K with increasing number of runs K, showing that a relatively small K is sufficient in most cases to reach optimal predictive performance. Afterwards, we compare FBED^K to FBS, feature selection with LASSO [139] and to the Max-Min Parents and Children algorithm (MMPC) [146] and show that it often has comparable predictive performance while selecting the fewest variables overall. Finally, we compare FBED^K to feature selection with LASSO [139] when both algorithms are limited to select the same number of variables, showing that both algorithms perform similarly. This, along with the generality of FBED^K makes it an interesting alternative to LASSO, especially for problems where LASSO requires specialized algorithms (like the group LASSO algorithm for logistic regression [102], LASSO for mixed-effects linear models [131], and the LASSO and group LASSO methods for functional Poisson regression [77]), which may be non-convex (as is the case for mixed-effects linear models [131] or for temporal-longitudinal data [60, 143]) and computationally demanding (taking several days to terminate on datasets with just 1000 predictors using Cox's proportional hazards model [47]).

1.3.2 Extending FBED^K for Big Data: The Parallel Forward-Backward with Pruning Algorithm

To address the challenges of Big Data described previously, in Chapter 5 we introduce the **Parallel**, **Forward-Backward with Pruning** (PFBP) algorithm, an extension of FBED^K for Big Data. PFBP does not rely on data sparsity and is generally applicable to both dense and sparse datasets; in the future, it could be extended to include optimizations specifically designed for sparse datasets. To tackle parallelization with hybrid partitioning (challenge (a) in Section 1.2.2), PFBP's decisions rely on p-values and log-likelihoods returned by the independence tests computed *locally* on each data partition; these values are then combined together using *statistical meta-analysis techniques* to produce *global* approximate p-values[3] and log-likelihoods. This technique essentially minimizes PFBP's communication cost, as only local p-values and log-likelihoods need to be communicated from workers to the master node in a cluster of machines at each iteration of the algorithm. The idea of combining local statistics to global statistics can also be applied to combining local predictive models trained on the currently selected features, to provide global predictive models. To reduce the number and workload of iterations required to compute a feature selection solution (challenge (b) in Section 1.2.2), PFBP relies on several heuristics. The Early Dropping heuristic allows the algorithm to scale-up with the number of variables. PFBP is also equipped with two new heuristics for *Early Stopping* of consideration of features within the same iteration, and *Early Returning* the current best feature for addition or removal. The three heuristics are implemented using *bootstrap-based statistical tests*. They are applied on the set of currently available local p-values and log-likelihoods to determine whether the algorithm has seen enough samples to make safely (i.e., with high probability of correctness) early decisions.

PFBP is first evaluated on a range of simulated data to assess its scalability properties. The data are simulated from randomly generated Bayesian networks in order to incorporate a complex dependency structure among the variables and include not only irrelevant features, but also redundant features. PFBP is found to scale super-linearly with the available sample size as its heuristics allow it to make early decisions before examining all available samples. It scales linearly with the number of features and available cores. PFBP is compared on a set of real datasets spanning a range of feature and sample sizes against the main forward-selection algorithms in the literature devised for Big Data architectures. PFBP is found more computationally efficient and scalable than the state-of-the-art, selecting fewer features, without sacrificing predictive performance. The algorithm is also evaluated against several variants of information-

[3]Alternatively, one can combine the test statistics that produce the p-values. This is conceptually equivalent, although there may be differences in practice.

theoretic feature selection algorithms. The latter are specialized for discrete and sparse data. PFBP is less computational efficient in this case, as it is not customized for discrete data, but exhibits a higher predictive performance. In all tasks in the evaluation, the predictive performance is assessed with models constructed with the standard logistic regression model in MLlib of the Spark distribution (hereafter SparkLR), as well as the combined predictive model constructed from the local logistic regressions ones, as proposed above (hereafter, CombLR). The experiments suggest that in several cases SparkLR fails to converge and one obtains a model that is significantly worse than random guessing (e.g., 45% accuracy of SparkLR vs. 85% accuracy for random guessing). In contrast, CombLR is never worse than the trivial model more than 0.02%. While the focus is on feature selection, these results indicate that methods for combining local statistics and models may serve a more general purpose in Big Data analytics.

1.3.3 A Strategy to Extend Forward-Backward Type Algorithms for Identifying Multiple Solutions

In Chapter 6 we deal with the problem of multiple feature selection. First, we consider the taxonomy of features proposed by John et al. [78], which assigns features into three categories: (a) strongly relevant features (provide unique information for the outcome and are necessary for optimal prediction), (b) weakly relevant features (provide information for the outcome, but are not necessary for optimal prediction), and (c) irrelevant features (do not provide any information for the outcome). We show that this taxonomy is counter-intuitive in the presence of multiple solutions, and propose an alternative which takes multiple solutions into account. Then, we state three definitions of statistical equivalence of solutions, show how they are related and how they can be tested, and make connections to the problem of model selection. Afterwards, we propose a general template for feature selection algorithms that consist of a greedy forward phase and an optional backward phase, and then introduce a simple and computationally efficient strategy to extend algorithms that fit into the above template for identifying multiple solutions. The main idea is to view the problem as a state space search problem over feature sets, and to use backtracking to explore multiple potential solutions. We prove that, under certain conditions, the proposed strategy identifies all equivalent solutions. We also propose a method to compactly represent and visualize equivalent solutions, improving the interpretability of the results, which is especially useful if the number of equivalent solutions is high. Finally, in experiments we compare the proposed strategy to TIE* [138], the only alternative method with similar theoretical guarantees, and show that both methods produce similar results in terms of predictive performance and number of solutions, with the proposed algorithm being faster.

Chapter 2
Preliminaries

We start by introducing the notation and terminology used throughout the thesis; additional notation specific to a chapter will be presented afterwards. We use upper-case letters to denote single variables (e.g., X), and bold upper-case letters to denote sets of variables (e.g., $\mathbf{Z}$). The number of elements in a set $\mathbf{Z}$ will be referred to as $|\mathbf{Z}|$. The terms variable, feature or predictor will be used interchangeably. The target variable (also called outcome) will be referred to as T. The input dataset will be denoted as $\mathcal{D}$, and the number of variables and samples in it will be denoted as p and n respectively. We use $\mathbf{F}$ to refer to the set of features in $\mathcal{D}$, excluding the target variable T. Conditional independence and dependence of sets of variables $\mathbf{X}$ and $\mathbf{Y}$ given $\mathbf{Z}$ is denoted as $\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z}$ and $\mathbf{X} \not\perp \mathbf{Y} \mid \mathbf{Z}$ respectively.

2.1 The Feature Selection Problem and a Taxonomy of Features

The **solution S** to the feature selection problem can be defined as *identifying a minimal-size subset of the variables that is optimally predictive for an outcome variable T of interest* [145], and is also called the **Markov blanket** of T [86]. More formally:

Definition 1 (Markov Blanket)**.** *A Markov blanket* $\mathbf{S}$ *of T is defined as* $\mathbf{S} = \underset{|\mathbf{S}'|}{\operatorname{argmin}}\{\mathbf{S}' \subseteq \mathbf{F} : T \perp \mathbf{F} \setminus \mathbf{S}' \mid \mathbf{S}'\}$*, that is,* $\mathbf{S}$ *is a minimal-size subset of* $\mathbf{F}$ *that renders T conditionally independent of all variables not in* $\mathbf{S}$*.*

John et al. [78] classify features into three categories: **strongly relevant** (also called indispensable), **weakly relevant** and **irrelevant** features; we will refer to this as the JKP taxonomy hereafter.

Definition 2 (Strongly Relevant Feature)**.** *A feature X is strongly relevant for T if* $T \not\perp X \mid \mathbf{F} \setminus \{X\}$*.*

Definition 3 (Weakly Relevant Feature)**.** *A feature X is weakly relevant for T if* $T \perp X \mid \mathbf{F} \setminus \{X\} \wedge \exists \mathbf{Z} \subseteq \mathbf{F} \setminus \{X\}, T \not\perp X \mid \mathbf{Z}$*.*

Definition 4 (Irrelevant Feature)**.** *A feature X is irrelevant for T if* $\forall \mathbf{Z} \subseteq \mathbf{F} \setminus \{X\}$, $T \perp X \mid \mathbf{Z}$.

Intuitively, a feature X is strongly relevant if it still provides additional predictive information for T given (conditioned on) all other features, weakly relevant if it is not strongly relevant but still provides information for T that is redundant given other features (X is informative for T given some subset $\mathbf{Z}$), and irrelevant if it does not provide any information for T (X is uninformative for T given any subset $\mathbf{Z}$). One would expect that an optimal feature selection would only return the strongly relevant features and filter out both the weakly relevant and the irrelevant features, as they do not provide additional information in the presence of all strongly relevant features. This is also what is suggested by the terminology of "strongly" and "weakly". In other words, one would expect that the strongly relevant features to correspond to the Markov Blanket features. Indeed, this is correct but only when there is a single, unique solution. However, as we describe in Section 6.1 when the solution to the problem is not unique, then there is not a single Markov Blanket and correspondence of strongly relevant features with the Markov Blanket breaks down.

2.2 Markov Blankets in Probabilistic Graphical Models

For distributions that can be represented by causal or probabilistic graphical models such as Bayesian networks and maximal ancestral graphs, the Markov blanket can be characterized based on their graphical structure. We proceed with a brief introduction to Bayesian networks and maximal ancestral graphs; for a comprehensive introduction to those models and their relation to causal modeling we refer the reader to [4, 114, 115, 126, 136].

A **directed acyclic graph** (DAG) is a graph that only contains directed edges ($\rightarrow$) and has no directed cycles. A **directed mixed graph** is a graph that, in addition to directed edges also contains bi-directed edges ($\leftrightarrow$). The graphs contain no self-loops, and vertices can be connected only by a single edge. Two vertices are called **adjacent** if they are connected by an edge. An edge between X and Y is called **into** Y if $X \rightarrow Y$ or $X \leftrightarrow Y$. A **path** in a graph is a sequence of unique vertices $\langle X_1, \ldots, X_k \rangle$ such that each consecutive pair of vertices is adjacent. The first and last vertices in a path are called **endpoints**. A path is called directed if $\forall 1 \leq i < k$, $X_i \rightarrow X_{i+1}$. If $X \rightarrow Y$ is in a graph, then X is a **parent** of Y and Y a **child** of X. A vertex W is a **spouse** of X, if both share a common child. A vertex X is an **ancestor** of Y, and Y is a **descendant** of X, if $X = Y$ or there is a directed path from X to Y. A triplet $\langle X, Y, Z \rangle$ is called a **collider** if Y is adjacent to X and Z, and both, X and Z are into Y. A triplet $\langle X, Y, Z \rangle$ is called **unshielded** if Y is adjacent to X and Z, but X and Z are not adjacent. A path p is called a **collider path** if every non-endpoint vertex is a collider on p.

2.2.1 Bayesian Networks

Bayesian networks (BNs) consist of a DAG $\mathcal{G}$ and a probability distribution $\mathcal{P}$ over a set of random variables $\mathbf{V}$. Each such variable is represented by a vertex in $\mathcal{G}$, and thus, the terms variable and vertex will be used interchangeably. The DAG represents dependency relations between variables in $\mathbf{V}$ and is linked with $\mathcal{P}$ through the **Markov condition**, which states that each variable is conditionally independent of its non-descendants given its parents. Those are not the only independencies encoded in the DAG; the Markov condition entails additional independencies, which can be read from the DAG using a graphical criterion called **d-separation** [114,152]. In order to present the d-separation criterion we first introduce the notion of blocked paths. A (not necessarily directed) path p between two vertices X and Y is called **blocked** by a set of vertices $\mathbf{Z}$ if there is a vertex X on p that is a collider and, neither X nor any of its descendants are in $\mathbf{Z}$, or if X is not a collider and it is in $\mathbf{Z}$. If all paths between X and Y are blocked by $\mathbf{Z}$, then X and Y are **d-separated** given $\mathbf{Z}$; otherwise X and Y are **d-connected** given $\mathbf{Z}$. The **faithfulness condition** states that all and only those conditional independencies in $\mathcal{P}$ are entailed by the Markov condition applied to $\mathcal{G}$. In other words, the faithfulness condition requires that two variables X and Y are d-separated given a set of variables $\mathbf{Z}$ if and only if they are conditionally independent given $\mathbf{Z}$.

2.2.2 Directed Maximal Ancestral Graphs

Bayesian networks are not closed under marginalization: a marginalized DAG, containing only a subset of the variables of the original DAG, may not be able to exactly represent the conditional independencies of the marginal distribution [126]. **Directed maximal ancestral graphs** (DMAGs) [126] are an extension of BNs, which are able to represent such marginal distributions, that is, they admit the presence of latent confounders. The graphical structure of a DMAG is a directed mixed graph with the following restrictions: (i) it contains no directed cycles, (ii) it contains no almost directed cycles, that is, if $X \leftrightarrow Y$ then neither X nor Y is an ancestor of the other, and (iii) there is no primitive inducing path between any two non-adjacent vertices, that is, there is no path p such that each non-endpoint on p is a collider and every collider is an ancestor of an endpoint vertex of p. The d-separation criterion analogue for DMAGs is called the **m-separation criterion**, and follows the same definition.

2.2.3 Markov Blankets

In BNs, the Markov blanket of T consists of its parents, children and spouses. For DMAGs it is slightly more complicated: the Markov blanket of T consists of its parents,

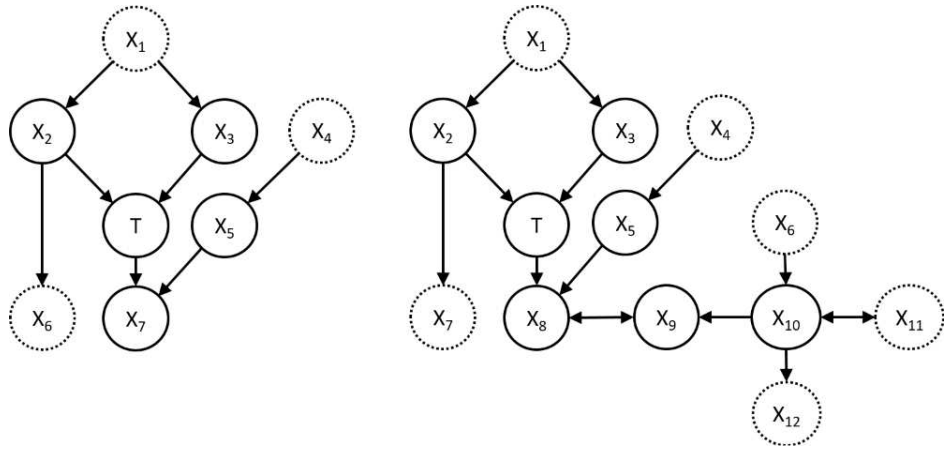


Figure 2.1: Example of Markov blankets of T in a Bayesian network (left) and a maximal ancestral graph (right). The vertices in the Markov blanket are shown with solid lines, and the remaining ones with dashed lines. In both cases, the Markov blanket contains all adjacent vertices (parents and children) and X_5 (spouse of T). In addition, in the maximal ancestral graph X_9 and X_{10} are also contained, as they are connected with T through a collider path ($T \rightarrow X_8 \leftrightarrow X_9 \leftarrow X_{10}$).

children and spouses, as well as its district (all vertices that are reachable by bi-directed edges), the districts of its children and the parents of vertices in all districts [125]. An example of the Markov blanket of T in a Bayesian network and a maximal ancestral graph are shown in Figure 2.1. An alternative definition is given next.

Definition 5 (Markov Blankets in Bayesian Networks and Directed Maximal Ancestral Graphs)**.** *The Markov blanket of T in a BN or DMAG consists of all vertices adjacent to T, as well as all vertices that are reachable from T through a collider path.*

A proof sketch follows. Recall that a collider path of length $k-1$ is of the form $X_1 * \rightarrow X_2 \ldots X_{k-1} \leftarrow * X_k$, where the path between X_2 and X_{k-1} contains only bi-directed edges. Given this, it is easy to see that Definition 5 includes vertices directly adjacent to T, its spouses (collider path of length 2), and in the case of DMAGs, vertices D in the district of T ($T \leftrightarrow \cdots \leftrightarrow D$), vertices D in the district of any children C of T ($T \rightarrow C \leftrightarrow \cdots \leftrightarrow D$), and all parents P of any vertex D in some district ($T * \rightarrow \cdots \leftrightarrow D \leftarrow P$). As the previous cases capture exactly all possibilities of vertices reachable from T through a collider path, Definition 5 does not include any additional variables that are not in the Markov blanket of T.

2.3 The Semi-Graphoid Axioms

The **semi-graphoid** axioms [115] are general axioms about conditional independence that hold in any probability distribution $\mathcal{P}$. Other axioms also exist, but only apply for

special distributions $\mathcal{P}$.

Symmetry	$\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z} \Rightarrow \mathbf{Y} \perp \mathbf{X} \mid \mathbf{Z}$
Decomposition	$\mathbf{X} \perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z} \Rightarrow \mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z} \wedge \mathbf{X} \perp \mathbf{W} \mid \mathbf{Z}$
Weak Union	$\mathbf{X} \perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z} \Rightarrow \mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z} \cup \mathbf{W}$
Contraction	$\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z} \wedge \mathbf{X} \perp \mathbf{W} \mid \mathbf{Y} \cup \mathbf{Z} \Rightarrow \mathbf{X} \perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}$
Intersection	$\mathbf{X} \perp \mathbf{Y} \mid \mathbf{W} \cup \mathbf{Z} \wedge \mathbf{X} \perp \mathbf{W} \mid \mathbf{Y} \cup \mathbf{Z} \Rightarrow \mathbf{X} \perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}$
	(If $\mathcal{P}$ is strictly positive)
Composition	$\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z} \wedge \mathbf{X} \perp \mathbf{W} \mid \mathbf{Z} \Rightarrow \mathbf{X} \perp \mathbf{Y} \cup \mathbf{W} \mid \mathbf{Z}$
	(If $\mathcal{P}$ is faithful to a BN/DMAG $\mathcal{G}$)

In case faithfulness is violated, the intersection and composition properties do not hold. The above axioms are useful tools for proving properties of algorithms, and will be used later on to prove correctness of the presented algorithms.

2.4 Stepwise Feature Selection

Stepwise methods [88,156] start with some set of selected variables and try to improve it in a greedy fashion, by either including or excluding a single variable at each step. There are various ways to combine those operations, leading to different members of the stepwise algorithmic family. Two popular members of the stepwise family are the **forward selection** and **backward selection** (also known as backward elimination) algorithms. Forward selection starts with a (usually empty) set of variables and adds variables to it, until some stopping condition is met (e.g., no variable provides additional information for T, or if a pre-specified number of variables have been selected). Similarly, backward selection starts with a (usually complete) set of variables and then excludes variables from that set, again, until some stopping criterion is met. Typically, both methods try to include or exclude the variable that offers the highest performance increase (i.e., resulting in a set of variables with higher predictive performance). We will call each step of selecting (removing) a variable a forward (backward) **iteration**. Executing forward (backward) iterations until termination will be called a forward (backward) **phase** respectively.

Algorithm 1 shows a general version of stepwise selection. To evaluate the performance of a set of variables, it uses the function Perf. Examples for Perf are the log-likelihood for logistic regression, the partial log-likelihood for Cox regression and the F-score for linear regression, or their AIC [2] or BIC [132] penalized variants. We note that Perf is not limited to any of the above; the only requirement is that it can be used to evaluate and compare sets of variables. The **selection criterion** C compares the performance of two sets of variables as computed by Perf; we will describe different selection criteria in the next section. Naturally, different stopping criteria can be em-

Algorithm 1 Stepwise Selection

Input: Dataset $\mathcal{D}$, Target T
Output: Selected Variables $\mathbf{S}$

1: $\mathbf{S} \leftarrow \emptyset$ // *Set of selected variables*
2: $\mathbf{R} \leftarrow \mathbf{F}$ // *Set of remaining candidate variables*
3: **while** $\mathbf{S}$ changes **do**
4: $\quad\mathbf{N} \leftarrow \{\mathbf{S}\}$ // *Candidate sets of selected variables for next iteration*
5: $\quad$**if** Forward Step Enabled **then**
6: $\quad\quad$// *Consider adding variables from* $\mathbf{R}$ *that improve* $\mathbf{S}$
7: $\quad\quad$**for all** $F_i \in \{F \in \mathbf{R} : \textsc{Perf}(\mathbf{S} \cup \{F\}) \underset{C}{>} \textsc{Perf}(\mathbf{S})\}$ **do**
8: $\quad\quad\quad\mathbf{N} \leftarrow \mathbf{N} \cup \{\mathbf{S} \cup \{F_i\}\}$
9: $\quad\quad$**end for**
10: $\quad$**end if**
11: $\quad$**if** Backward Step Enabled **then**
12: $\quad\quad$// *Consider removing variables from* $\mathbf{S}$ *that do not worsen* $\mathbf{S}$
13: $\quad\quad$**for all** $F_i \in \{F \in \mathbf{S} : \textsc{Perf}(\mathbf{S} \setminus \{F\}) \underset{C}{\geq} \textsc{Perf}(\mathbf{S})\}$ **do**
14: $\quad\quad\quad\mathbf{N} \leftarrow \mathbf{N} \cup \{\mathbf{S} \setminus \{F_i\}\}$
15: $\quad\quad$**end for**
16: $\quad$**end if**
17: $\quad$// *Identify best set of selected variables from* $\mathbf{N}$
18: $\quad\mathbf{S} \leftarrow \underset{\mathbf{S}' \in \mathbf{N}}{\operatorname{argmax}}\ \textsc{Perf}(\mathbf{S}')$
19: $\quad$// *Update set of remaining variables*
20: $\quad\mathbf{R} \leftarrow \textsc{UpdateRemainingVariables}(\mathbf{R}, \mathbf{S})$
21: **end while**
22: **return** $\mathbf{S}$

bedded into the selection criterion (e.g., to stop once a maximum number of variables has been selected), and thus there is no need to explicitly use a stopping criterion in the presentation of the algorithm. We will use the predicates $\underset{C}{>}$, $\underset{C}{\geq}$ and $\underset{C}{=}$ to compare two sets of variables; they are true if the left-hand-side value is greater, greater or equal, or equal than the right-hand-side value respectively, according to the criterion C.

We proceed with the description of the stepwise selection algorithm. It starts with an empty set of selected variables $\mathbf{S}$, and initially considers all variables $\mathbf{F}$ as candidates for selection. At each iteration it considers all sets of selected variables $\mathbf{N}$ that can be obtained by selecting a variable from $\mathbf{R}$, or by removing one of the already selected variables from $\mathbf{S}$. When including variables, only the ones that strictly improve $\mathbf{S}$ are considered, while variables can be removed as long as they don't decrease the performance. Afterwards, the best set $\mathbf{S} \in \mathbf{N}$ is chosen, and the set of remaining variables is updated (UpdateRemainingVariables function). How $\mathbf{R}$ is updated depends

on the specific instantiation of the algorithm. Typically, one can just set $\mathbf{R} \leftarrow \mathbf{F} \setminus \mathbf{S}$. Now, depending on when forward and/or backward steps are enabled, one can obtain different versions of the stepwise algorithmic family. For instance, forward or backward selection are trivially obtained by disabling the backward and forward steps respectively, while forward-backward selection can be obtained by disabling backward selection as long as variables can be selected, and switching to backward selection only afterwards. The implementation details of enabling or disabling forward and backward steps are omitted for brevity.

2.4.1 Criteria for Variable Selection

Next we will describe some performance functions and selection criteria that are employed in practice; for additional details see [88, 156]. The most common choices are statistical tests, information criteria and cross-validation. We will focus on statistical tests, as we use them in the remainder of the thesis. We will also describe information criteria and contrast them to statistical tests, but will not further consider cross-validation, mainly because of its high computational cost.

Statistical Tests and Conditional Independence

Since the variable sets tested at each iteration are nested (i.e., one of the sets is a subset of the other), one can employ a nested likelihood-ratio test (LRT) (or asymptotically equivalent approximations thereof such as score tests and Wald tests [46]) for nested models as a selection criterion. This is done by using an appropriate statistical model $\mathcal{M}$ for the data and outcome (e.g., logistic regression for categorical outcomes), and testing whether there is a significant difference of how well the variable sets fit the data. Let T be the outcome and $\mathbf{X}$, $\mathbf{X} \cup \mathbf{Y}$ be two sets of variables, with the latter being a superset of the former, and let $\mathcal{M}(T|\mathbf{Z})$ denote the predictive model obtained for T using statistical model $\mathcal{M}$ and variables $\mathbf{Z}$. The LRT fits two models: (i) model $\mathcal{M}(T|\mathbf{X})$ using only variables $\mathbf{X}$ (the null model), and (ii) model $\mathcal{M}(T|\mathbf{X} \cup \mathbf{Y})$ using all variables $\mathbf{X} \cup \mathbf{Y}$ (the alternative model). Let $\mathrm{LL}(T|\mathbf{X})$ and $\mathrm{LL}(T|\mathbf{X} \cup \mathbf{Y})$ denote the log-likelihood of models $\mathcal{M}(T|\mathbf{X})$ and $\mathcal{M}(T|\mathbf{X} \cup \mathbf{Y})$ respectively. The test statistic is computed as

$$\text{Statistic} \equiv -2 \cdot (\mathrm{LL}(T|\mathbf{X}) - \mathrm{LL}(T|\mathbf{X} \cup \mathbf{Y}))$$

(hence the name likelihood-ratio test). Under the null hypothesis that both models fit the data equally well, the test statistic follows asymptotically a χ^2 distribution with $\textsc{Par}(T|\mathbf{X} \cup \mathbf{Y}) - \textsc{Par}(T|\mathbf{X})$ degrees of freedom [159] [1], where $\textsc{Par}$ denotes the number of

[1] This result assumes that the larger hypothesis is correctly specified. In case of model misspecification, the statistic follows a different distribution [57]. Methods to handle model misspecification have been proposed

parameters (degrees of freedom) of a model. For continuous features in $\mathbf{Y}$, only one parameter is used so the difference in degrees of freedom increases by 1. Categorical predictors can be used by simply encoding them as $K-1$ dummy binary features, where K is the number of possible values of the original feature. In this case, the difference in degrees of freedom increases by $K-1$. In the formulation of Algorithm 1, the performance Perf of a LRT is computed using the log-likelihood of the model, and the criterion C tests the hypothesis that both variable sets are equivalent for T with respect to some prespecified significance level α.

Alternatively, one can view the LRT as testing whether the coefficients of $\mathbf{Y}$ in model $M(T|\mathbf{X} \cup \mathbf{Y})$ are zero, or equivalently that $\mathbf{Y}$ is conditionally independent of the target T given $\mathbf{X}$ relative to the statistical model used. Thus, LRT for nested models are essentially **conditional independence tests**, relative to some statistical model, and assuming that the model is correctly specified. Conditional independence of $\mathbf{Y}$ with T given $\mathbf{X}$ implies that $P(T|\mathbf{X} \cup \mathbf{Y}) = P(T|\mathbf{X})$, whenever $P(\mathbf{X} > 0)$ ($\mathbf{X}$ is allowed to be the empty set). Thus, when conditional independence holds, $\mathbf{Y}$ is not predictive of T when $\mathbf{X}$ (and only $\mathbf{X}$) is known. The null hypothesis is that features $\mathbf{Y}$ are probabilistically independent of T (i.e., redundant or irrelevant) given a set of variables $\mathbf{X}$. The test returns a p-value, which corresponds to the probability that one obtains deviations from what is expected under the null hypothesis as extreme or more extreme than the deviation actually observed with the given data, and will be denoted as $\textsc{Test}(T, \mathbf{Y}|\mathbf{X})$. In practice, decisions are made using a threshold α (significance level) on the p-values; the null hypothesis is rejected if the p-value is below α. Examples of LRT-based conditional independence tests are the G-test [1] for multinomial data, while the partial correlation test [54] is a conditional independence test for multivariate Gaussian data, which is asymptotically equivalent to a LRT using linear regression [31]. In general, LRT can be constructed for any type of data for which an algorithm for maximizing the data likelihood exists, such as binary, multinomial or ordinal logistic regression, linear regression and Cox regression to name a few, enabling them to handle different types of outcome T, as well as mixed continuous and categorical variables in $\mathbf{X}$ and $\mathbf{Y}$ (see above). Finally, we note that one is not limited to LRT conditional independence tests, but can use any appropriate conditional independence test, such as a kernel-based test [164] which also handle non-linear dependencies.

A problem when using statistical tests for feature selection is that, due to multiple testing, the test statistics do not have the claimed distribution [69] and the resulting p-values are too small [56, 67], leading to a high false discovery rate. Approaches to deal with problem include methods that dynamically adjusting significance levels [76], or methods that directly deal with the problem of sequential testing of stepwise

by [158] and [154]. A method for dealing with model misspecification in model selection with information criterion is presented in [98]. As this problem is out of this thesis's scope, we did not further consider it.

procedures [61, 140]. In order to perform tests on the model returned by stepwise selection, one can use resampling-based procedures to correct the p-values [52]. In addition to the above problems, we note that the model returned by stepwise selection is sub-optimal, as it will have inflated coefficients [56], reducing its predictive ability. If the main focus is to obtain a predictive model, methods performing regularization (like L1, L2 or elastic net) are more appropriate. In any case, procedures like cross-validation should be used to estimate out-of-sample predictive performance of the final model. We will not consider the above hereafter; we note however that the proposed algorithms are orthogonal to those methods and could be used in conjunction with them.

Information Criteria

Another way to compare two (or more) competing models is to use information criteria, such as the Akaike information criterion (AIC) [2] or the Bayesian information criterion (BIC) [132]. Information criteria are based on the fit of a model but additionally penalize the model by its complexity (that is, the number of parameters). The AIC and BIC scores of a model $\mathcal{M}(T|\mathbf{X})$ for T based on $\mathbf{X}$ are defined as follows:

$$\mathrm{AIC}(T|\mathbf{X}) \equiv -2 \cdot \mathrm{LL}(T|\mathbf{X}) + 2 \cdot \textsc{Par}(T|\mathbf{X})$$
$$\mathrm{BIC}(T|\mathbf{X}) \equiv -2 \cdot \mathrm{LL}(T|\mathbf{X}) + \log(n) \cdot \textsc{Par}(T|\mathbf{X})$$

where n is the number of samples. In the framework described above, information criteria can be applied by using the information criterion value as the performance function $\textsc{Perf}$, and a selection criterion C that simply compares the performance of two models, giving preference to the one with the lowest value. Alternatively, one could check that the difference in scores is larger than some threshold.

Neither AIC nor BIC are designed for cases where the number of predictors p is larger than the number of samples n [29], and thus also suffer from a high false discovery rate, similar to statistical tests. There have been several extensions to handle this problem, like the extended Bayesian information criterion (EBIC) [29], the generalized information criterion (GIC) [49, 83], and the corrected risk information criterion (RIC_c) [165], to name a few.

Compared to statistical tests, information criteria are somewhat limited as they can only be computed for models where the model complexity is known (like generalized linear models). An example where information criteria are not applicable are kernel-based tests [164]. Thus, statistical tests are inherently more general than information criteria. We will show next how, in case of nested models, using BIC directly corresponds to a likelihood-ratio test for some significance level α; the same reasoning

Algorithm 2 Forward-Backward Selection with Conditional Independence Tests

Input: Dataset $\mathcal{D}$, Target T, Significance Level a
Output: Selected Features $\mathbf{S}$

```
S ← ∅ //Set of selected variables
R ← F //Set of remaining candidate variables
//Forward Phase: Iterate until no more features can be selected
while S changes do
    //Identify F* with minimum p-value conditional on S
    F* ← argmin_{F∈R} TEST(T, F|S)
    //Select F* if conditionally dependent with T given S
    if TEST(T, F*|S) ≤ a then
        S ← S ∪ {F*}
        R ← R \ {F*}
    end if
end while
//Backward Phase: Iterate until no more features can be removed from S
while S changes do
    //Identify F* with maximum p-value conditional on S \ {F*}
    F* ← argmax_{F∈S} TEST(T, F|S \ {F})
    //Remove F* if conditionally independent with T given S \ {F*}
    if TEST(T, F*|S \ {F*}) > a then
        S ← S \ {F*}
    end if
end while
return S
```

can be applied to AIC and all information criteria that are computed based on the model likelihood and a penalty term. Let $\mathbf{X}$ and $\mathbf{X} \cup \mathbf{Y}$ be two candidate variables sets. $\mathbf{X} \cup \mathbf{Y}$ is selected (that is, the null hypothesis is rejected) if $\text{BIC}(T|\mathbf{X}) > \text{BIC}(T|\mathbf{X} \cup \mathbf{Y})$, or equivalently if $2 \cdot \text{LL}(T|\mathbf{X} \cup \mathbf{Y}) - 2 \cdot \text{LL}(T|\mathbf{X}) > \log(n) \cdot (\textsc{Par}(T|\mathbf{X} \cup \mathbf{Y}) - \textsc{Par}(T|\mathbf{X}))$. Note that the left-hand side term equals the statistic of a likelihood-ratio test, whereas the right-hand size corresponds to the critical value. The statistic follows a χ^2 distribution with $k = \textsc{Par}(T|\mathbf{X} \cup \mathbf{Y}) - \textsc{Par}(T|\mathbf{X})$ degrees of freedom, and thus, the significance level equals $a = 1 - F(\log(n) \cdot k; k)$, where $F(v; k)$ is the χ^2 cdf with k degrees of freedom at value v.

2.4.2 Forward-Backward Selection with Conditional Independence Tests

The Forward-Backward Selection algorithm (FBS) is an instance of the stepwise feature selection algorithm family [88, 156]. It is also one of the first and most popular algorithms for causal feature selection [100, 101, 147], and has been shown to identify

the Markov blanket in distributions that can be faithfully represented by Bayesian networks and maximal ancestral graphs [138]. It can be shown to be optimal even under weaker assumptions. As shown by Statnikov et al. [138], the only requirement is that the local composition property with respect to T holds, which is a consequence of faithfulness (see semi-graphoid axioms in Section 2.3), i.e., that the composition property holds with $\mathbf{X} = \{T\}$. Algorithm 2 shows an instantiation of FBS using conditional independence tests.

We chose to present it using conditional independence tests for several reasons. First of all, as shown in the previous section, likelihood-ratio tests, F-tests (which are Wald tests, and asymptotically equivalent to LRT) and information criteria, all of which are usually used in the statistical literature, can be viewed as conditional independence tests. Formulating it using conditional independence tests allows one to also use tests not based on LRT (e.g., kernel tests [164]). Furthermore, it has the important advantage that it *allows one to adapt and apply the algorithm to any type of outcome* for which an appropriate statistical test of conditional independence exists. This way, the same feature selection algorithm can deal with different data types[2]. Finally, in the context of feature selection, the p-values returned by statistical hypotheses tests of conditional independence can be employed not only to reject or accept hypotheses, but also *to rank the features according to the predictive information they provide for* T given $\mathbf{S}$. Intuitively, this can be justified by the fact that everything else being equal (i.e., sample size, type of test) the p-values of such tests in case of dependence have (on average) the reverse ordering with the conditional association of the variables with T given $\mathbf{S}$. So, the basic variant of the algorithm selects to add (remove) the feature with the lower (higher) p-value in each Forward (Backward) Iteration.

2.5 Combining p-values Using Meta-Analysis Techniques

A set of p-values stemming from testing the *same* null hypothesis (e.g. testing the conditional independence of X and Y given $\mathbf{Z}$) can be combined using statistical meta-analysis techniques into a single p-value. Multiple such methods exist in the literature [97]. Fisher's combined probability test [53] is one such method that has been shown to work well across many cases [97]. It assumes that the p-values are independent and combines them into a single statistic using the formula

$$\text{Statistic} \equiv -2 \cdot \sum_{i=1}^{K} \log(p_i)$$

[2]For example, the R-package MXM [89] includes asymptotic, permutation-based, and robust tests for nominal, ordinal, continuous, time-course, percentage, count, and censored time-to-event targets.

where K is the number of p-values, p_i is the i-th p-value, and log is the natural logarithm. The statistic is then distributed as a χ^2 random variable with $2 \cdot K$ degrees of freedom, from which a combined p-value is computed.

2.6 Bootstrap-based Hypothesis Testing

The bootstrap procedure [45] can be used to compute the distribution of a statistic of interest. Bootstrapping is a general-purpose non-parametric resampling-based procedure which works as follows: (a) resample with replacement from the input values a sample of equal size, (b) compute the statistic of interest on the bootstrap sample, (c) repeat steps (a) and (b) many times to get an estimate of the bootstrap distribution of the statistic. The bootstrap distribution can then be used to compute properties of the distribution such as confidence intervals, or to compute some condition of the statistic. A simple example application on the latter follows; more examples can be found in [45].

Let μ_X denote the mean of random variable X and let $\hat{\mu}_X$ denote the estimate of the mean of X given a sample of X. Assume we are given a sample of size n of random variable X and we want to compute the probability that the mean of X is larger than 10, $P(\mu_X > 10)$. That probability is a Bernoulli random variable, and the statistic in this case is a binary valued variable (i.e., taking a value of 0 or 1 with probability $P(\mu_X > 10)$). Using bootstrapping, $P(\mu_X > 10)$ can be estimated as follows: (a) sample with replacement n values of X and create the b-th bootstrap sample X^b, (b) estimate the mean of X^b, denoted as $\hat{\mu}_X^b$, and compute $I(\hat{\mu}_X^b > 10)$, where I is the indicator function returning 1 if the inequality holds and 0 otherwise, and (c) repeat (a) and (b) B times (e.g. $B = 1000$). $P(\mu_X > 10)$ is then computed as

$$P(\mu_X > 10) = \frac{I(\hat{\mu}_X > 10) + \sum_{i=1}^{B} I(\hat{\mu}_X^b > 10)}{B+1}$$

Note that, we also compute the statistic on the original sample (which is sample from the bootstrap distribution), and thus divide by $B+1$[3].

[3]Often, the original sample is not considered and thus the estimate is computed by using only the bootstrap samples and dividing by B. However, it has been noted (in the context of bootstrap-based hypothesis testing) that one should also consider the original sample statistic (see Section 4.2 in [40]), which is why we chose to do so too.

Chapter 3
Variations of Stepwise Feature Selection Methods and Their Relation to Sparsity-Based, Information Theoretic and Causal-Based Approaches

In this chapter we will briefly review and compare some common and popular classes of feature selection algorithms that have appeared in the fields of statistics, computer science and signal processing. These include members of the stepwise selection family [88, 156], sparsity-based methods [139], information-theoretic methods [22], and causal-based methods [4]. We will show that many prominent methods from different fields are simple variations or approximations of forward selection or the more general stepwise selection, and that the same algorithms have reappeared multiple times across fields. *Based on this view and connections between the algorithms, we argue that any extension to stepwise methods, such as the ones proposed in this thesis, can be translated and directly be applied with any those feature selection algorithms.*

The forward selection algorithm has often reappeared under different names. In the statistical literature, it appears as *forward stepwise regression*, *forward stepwise selection*, or simply forward selection [44,68,70,88,156]. In the signal processing community forward selection is known as *orthogonal least squares* [30]. In computer science, variations of the forward-backward selection has reappeared in the context of Markov blanket discovery and Bayesian network learning, such as the *Grow-Shrink* (GS) [100,101] and the *Incremental Association Markov Blanket* (IAMB) algorithms [147]. A high level description of forward selection is given in Figure 3.1. This view will allow an easier comparison between forward selection and other similar algorithms.

Forward Selection
1) Initialize the set of selected variables to $\mathbf{S} = \emptyset$.
2) Fit a model $\mathcal{M}(T|\mathbf{S})$ for T using all variables $\mathbf{S}$.
3) Find $F_i \in \mathbf{F} \setminus \mathbf{S}$ that, when selected, produces the best fitting model $\mathcal{M}(T|\mathbf{S} \cup \{F_i\})$, and add it to $\mathbf{S}$.
4) Repeat steps 2 and 3 until the stopping condition is met.

Orthogonal Matching Pursuit
1) Initialize the set of selected variables to $\mathbf{S} = \emptyset$.
2) Fit a model $\mathcal{M}(T|\mathbf{S})$ for T using all variables $\mathbf{S}$.
3) Find $F_i \in \mathbf{F} \setminus \mathbf{S}$ that has the largest correlation with the residuals of the current model $\mathcal{M}(T|\mathbf{S})$, and add it to $\mathbf{S}$.
4) Repeat steps 2 and 3 until the stopping condition is met.

Forward Stagewise Regression
1) Initialize a model $\mathcal{M}(T|\mathbf{F})$, where each coefficient is initially set to zero (i.e., $\mathbf{S} = \emptyset$).
2) Find $F_i \in \mathbf{F}$ that has the largest correlation c_i with the residuals of the current model $\mathcal{M}(T|\mathbf{F})$.
3) Update the model $\mathcal{M}(T|\mathbf{F})$, by updating the coefficient b_i of F_i as $b_i \leftarrow b_i + \epsilon \cdot sign[c_i]$, where ϵ is a prespecified step size hyper-parameter.
4) Repeat steps 2 and 3 until the stopping condition is met. Return the set of variables with non-zero coefficients in $\mathcal{M}(T|\mathbf{F})$.

Least Angle Regression
1) Initialize a model $\mathcal{M}(T|\mathbf{F})$, where each coefficient is initially set to zero (i.e., $\mathbf{S} = \emptyset$).
2) Find $F_i \in \mathbf{F}$ that has the largest correlation c_i with the residuals of the current model $\mathcal{M}(T|\mathbf{F})$.
3) Change the coefficient of b_i and of all non-zero coefficients in $\mathcal{M}(T|\mathbf{F})$ in the direction defined by their joint least-squares fit, until some other variable has as much correlation with the residual.
4) Repeat steps 2 and 3 until the stopping condition is met. Return the set of variables with non-zero coefficients in $\mathcal{M}(T|\mathbf{F})$.

Figure 3.1: High level description of forward selection and similar algorithms for the linear regression problem.

3.1 Variations of Forward Selection

Next, we describe several methods that can be viewed as variations of the standard forward selection algorithm. For simplicity, we will describe them for the linear regression problem, although some of them have been also extended for other problems (e.g., for generalized linear models). Furthermore, we assume that all variables have been standardized to have zero mean and unit variance, and that the outcome variable

T has been centered (i.e., it has zero mean).

3.1.1 Orthogonal Matching Pursuit

The *orthogonal matching pursuit* (OMP) algorithm [39, 113] has appeared in the signal processing community, and can be seen as an approximation to forward selection. A high-level description of OMP is shown in Figure 3.1. The main difference with forward selection is that it first identifies the next variable F_i to select based on its correlation with the residuals of the current model, and then includes F_i in the model. In contrast, forward selection selects the variable that, when included, leads to the best model. Thus, OMP can be seen as approximating the variable ranking of forward selection using the correlation of features with the residuals of the current model. This approximation makes OMP more computationally efficient than forward selection, as OMP fits a single model at each iteration, while forward selection fits a model for each non-selected variable. A comparison between OMP and forward selection (called orthogonal least squares) can be found in [11].

3.1.2 Forward Stagewise Regression and Least Angle Regression

Another class of algorithms are *Forward Stagewise Regression* (FSR) and *Least Angle Regression* (LARS) [44]. In contrast to forward selection and OMP they do not fit a full model at any stage, but gradually modify the coefficients of the variables in the current model. A description of both algorithms, based on [68], is shown in Figure 3.1. At each step, FSR updates the coefficient of the best variable F_i by a factor (step size) of ϵ in the direction of the correlation of F_i with the residuals (the sign of the correlation). Thus, it may require many steps before including another variable in the model. LARS is an extension of FSR that updates the coefficient of the best variable F_i "as much as possible", and thus will "fully include" a variable at each step. Both algorithms have the same computational advantage over forward selection as OMP, as they rank the variables based on the correlation with the residuals. A detailed description of the relationship between FSR and LARS can be found in Section 3.2 in [44], while a comparison of LARS and OMP is given in [66].

3.1.3 The Lasso

The LASSO [139] is perhaps one of the most widely used approaches to the feature selection problem. The feature selection problem is expressed as a global optimization problem using an L_1 penalty on the feature coefficients. Let $D(\mathcal{M}(T|\mathbf{S})) \equiv -2 \cdot \mathrm{LL}(\mathcal{M}(T|\mathbf{S}))$ be the deviance of a (generalized linear) model using m variables $\mathbf{S}$, and β be the vector of coefficients of $\mathbf{S}$ in $\mathcal{M}(T|\mathbf{S})$. For linear regression, the deviance

is the mean squared error. The optimization problem LASSO solves can be expressed as

$$\min_{\beta \in \mathbb{R}^m} D(\mathcal{M}(T|\mathbf{S})) + \lambda \left\|\beta\right\|_1$$

where $\left\|\beta\right\|_1$ is the L_1 norm and $\lambda \geq 0$ is a regularization parameter. The solutions LASSO returns are sparse, meaning that most coefficients are set to zero, thus implicitly performing feature selection. The regularization parameter λ controls the number of non-zero coefficients in the solution, with larger values leading to sparser solutions. This problem formulation is a convex approximation of the more general best subset selection (BSS) problem [106], defined as follows to match the LASSO optimization formulation

$$\min_{\beta \in \mathbb{R}^m} D(\mathcal{M}(T|\mathbf{S})) + \lambda \left\|\beta\right\|_0$$

where $\left\|\beta\right\|_0$ is the 0-norm (i.e., the total number of variables with non-zero coefficients). Conditions for optimal feature selection with LASSO are given in [103].

While LASSO is defined as a global optimization problem, it has been proven that it can be solved efficiently with a modified version of LARS for many tasks (e.g., for linear regression [44]). Intuitively, the only change required in LARS is that, once a non-zero coefficient becomes zero during an update, to set it to 0 and re-compute the update step. Thus, in contrast to forward selection, LARS and FSR, LASSO may also remove variables, and can be seen as a variation of the general stepwise selection instead. A detailed comparison of LASSO, LARS and FSR is given in [44].

3.2 Information Theoretic Feature Selection Algorithms

Information theoretic feature selection (ITFS) methods rely on the estimations of the mutual information (MI) and the conditional mutual information (CMI) to rank and select features, and many variations have appeared in the literature, such as the *Minimum-Redundancy Maximum-Relevance* (MRMR) [118], the *Joint Mutual Information* (JMI) [161] and the *Conditional Mutual Information Maximization* (CMIM) [55] algorithms; an overview of prominent ITFS methods and their relation is presented in [22]. ITFS methods assume discrete features and outcomes; thus, we will hereafter assume that non-discrete data have been discretized. A more detailed description follows.

The criterion J of several ITFS methods[1] for evaluating feature X_k can be expressed as

$$J(X_k) = I(T; X_k) - \beta \sum_{X_j \in \mathbf{S}} I(X_j; X_k) + \gamma \sum_{X_j \in \mathbf{S}} I(X_j; X_k | T)$$

[1]There are methods that do not fall into this framework, but we will not go into more detail; see [22] for more details.

where β and γ are parameters taking values in $[0, 1]$, and I denotes the mutual or conditional mutual information. The intuition for $J(X_k)$ above is that J increases with the information X_k directly provides for the target T (the first term), decreases with the information the other selected features already provide for X_k (second group of terms), and increases when X_k interacts with the selected features, i.e., one provides information for the other conditioned on T (third group of terms). *ITFS methods also perform a greedy type of forward selection, adding a feature at a time*, albeit with a different selection criterion. The next best feature is chosen as the one maximizing J with respect to the current set of selected variables $\mathbf{S}$.

Brown et al. [22] describe some similarities of ITFS methods and forward selection. When selecting features, both approaches try to identify the feature X maximizing the conditional information with T given the currently selected set of features $\mathbf{S}$:

$$\mathbf{S} \leftarrow \mathbf{S} \cup \operatorname*{argmax}_{X \in \mathbf{F} \setminus \mathbf{S}} I(T; X|\mathbf{S})$$

Assuming discrete variables, the quantity $I(T; X|\mathbf{S})$ requires an exponential number of samples with respect to the variables participating in it and the domain of the variables (i.e., the number of unique values they can take). For instance, if all variables are binary, then there are $2^{|\mathbf{S}|+2}$ value combinations, thus requiring an exponential sample size to accurately estimate $I(T; X|\mathbf{S})$. ITFS methods deal with this by using low-order approximations, conditioning up to one variable (see criterion J). Methods like forward selection (as well as OMP, LARS and LASSO) typically approximate $I(T; X|\mathbf{S})$ using linear statistical models, such as linear or logistic regression, which only require estimation of a linear number of parameters. Recall however that forward selection is not limited to linear models, but can also use non-linear tests, such as the G-test for multinomial data [1] or kernel-based tests for continuous data [164].

Apart from the similarity of ITFS and statistical methods described above, we want to point out that, *information theoretic methods and statistical methods are intricately theoretically connected*, and are not two completely different approaches. *Estimations of MI and CMI directly correspond to performing statistical hypotheses tests.* First, notice that

$$I(X; Y|\mathbf{Z}) = 0 \Leftrightarrow X \perp Y \mid \mathbf{Z}$$

($\mathbf{Z}$ can be empty), i.e., the (conditional) mutual information is zero if and only if the (conditional) independence holds. To estimate the MI one is forced to assume a specific probabilistic model for their data, which directly relates to an equivalent statistical test. For example, assuming a multinomial joint distribution of discrete features X, Y and

$\mathbf{Z}$, the CMI is related to the statistic of the G-test of conditional independence [1] as

$$\text{Statistic} \equiv 2 \cdot n \cdot \hat{I}(X; Y|\mathbf{Z})$$

where n is the sample size and $\hat{I}$ denotes the estimated quantity. Therefore, there is a direct correspondence between thresholding on the CMI and accepting as zero the ones below the threshold, to thresholding on a statistical p-value on a G-test above and accepting dependence.

The main advantage of ITFS over forward selection and other multivariable methods is that estimating the (low order) CMI is easier and computationally faster than fitting a model. On the other hand, ITFS methods are not as general as forward selection, which can be applied to different data types using appropriate conditional independence tests. For example, it is not clear if and how ITFS can be applied to time-to-event outcome variables or time-course data. Furthermore, ITFS variants are only applicable to discrete data, and thus require the use of discretization methods for continuous data, possibly losing information [43, 82]. This not only increases computational time but also may require extra tuning to find a good discretization of features. Last but not least, ITFS methods do not have the same theoretical properties as multivariable methods. For instance, forward-backward selection has been shown to be optimal for distributions that can be faithfully represented by causal networks [100, 101, 138, 147], while ITFS methods may select false positive variables (they won't remove variables that are conditionally independent with T given 2 or more variables), and may also miss variables (they won't select variables which are conditionally dependent with T given 2 or more variables).

3.3 Causal-Based Markov Blanket Discovery Algorithms

Apart from GS [100, 101] and IAMB [147], which are variants of forward selection inspired by causal models, another class of causal algorithms are methods like HITON [5], MMPC [146], and more recently SES [89] for multiple solutions. These algorithms are also based on conditional independence tests, and thus are general and applicable to different data types. The main difference with forward selection is that they condition on *subsets of the selected features* $\mathbf{S}$, not the full set. They do not guarantee to identify the full Markov blanket, but only a superset of the neighbors of T in causal graphs. These algorithms remove from consideration any features that become independent of T conditioned on *some* subset of the selected features $\mathbf{S}$.

Similar to forward-backward selection, HITON, MMPC and SES have a forward and a backward phase, including and removing one variable at a time; we focus on the forward phase and MMPC hereafter. For MMPC, the next variable to include is

selected as follows:

$$\mathbf{S} \leftarrow \mathbf{S} \cup \operatorname*{argmax}_{X \in \mathbf{F} \setminus \mathbf{S}} \min_{\mathbf{Z} \subseteq \mathbf{S} \wedge |\mathbf{Z}| \leq k} \textsc{Association}(T; X|\mathbf{Z})$$

where k is a parameter specifying the maximum size of $\mathbf{Z}$. In words, for each variable X, MMPC computes the minimum association (e.g., CMI or negative p-value of a conditional independence test) given any subset $\mathbf{Z}$ of $\mathbf{S}$ with maximum size k, and selects the variable having the maximum such value. The association is measured using p-values of conditional independence tests, and a threshold is used to decide when to stop selecting variables, as well as when to remove variables from consideration. MMPC and similar methods can be seen as being in between forward selection (which conditions on all selected variables) and ITFS methods (which condition on subsets of size 1 at most), trading-off the types of dependencies it can identify (due to the maximum conditioning size) with increased sample efficiency (fewer samples are required to estimate the association with small conditioning sets). Furthermore, some instances of ITFS such as CMIM [55] can be shown to specific instances of MMPC (CMIM is MMPC with $k = 1$ and significance level $\alpha = 1$).

Chapter 4
Forward-Backward Selection with Early Dropping

The standard Forward-Backward Selection (FBS) algorithm has two main issues. The first is that it is slow: at each forward iteration, all remaining variables are reconsidered to find the best next candidate. To select k variables, it performs $O(k \cdot p)$ independence tests, where p is the number of variables in the input dataset $\mathcal{D}$. Although relatively low-dimensional datasets are manageable, it can be very slow for modern datasets which often contain thousands of variables. The second problem is that it suffers from multiple testing issues, resulting in overfitting and a high false discovery rate. This happens because it reconsiders all remaining variables at each iteration; variables will often happen to seem important simply by chance, if they are given enough opportunities to be selected. As a result, it will often select a significant number of false positive variables [56]. This behavior is further magnified in high-dimensional settings and with larger significance levels α. Next, we describe a simple modification of FBS, improving its running time while reducing the problem of multiple testing.

4.1 The Early Dropping Heuristic

We propose the following modification: after each forward iteration, remove all variables that do not satisfy the criterion C for the current set of selected variables $\mathbf{S}$ from the remaining variables $\mathbf{R}$. In our case, those variables are the ones that are conditionally independent of T given $\mathbf{S}$. The idea is to quickly reduce the number of candidate variables $\mathbf{R}$, while keeping many (possibly) relevant variables in it. The forward phase terminates if no more variables can be selected, either because there is no informative variable or because $\mathbf{R}$ is empty; to distinguish between forward and backward phases, we will call a forward phase with early dropping a **run**. Extra runs can be performed to reconsider variables dropped previously. This is done by retaining the previously selected variables $\mathbf{S}$ and initializing the set of remaining variables to

Algorithm 3 Forward-Backward Selection with Early Dropping (FBED^K)

Input: Dataset $\mathcal{D}$, Target T, Significance Level α, Maximum Number of Runs K
Output: Selected Variables $\mathbf{S}$

```
S ← ∅ // Set of selected variables
K_cur ← 0 // Initializing current number of runs to 0
// Forward phase: iterate until (a) run limit reached, or (b) S does not change
while K_cur ≤ K ∧ S changes do
    S ← OneRun(D, T, S, α)
    K_cur ← K_cur + 1
end while
// Perform backward selection and return result
return BackwardSelection(D, T, S, α)
```

```
function OneRun(D, T, S, α)
    R ← F \ S // Set of remaining candidate variables
    // Forward phase: iterate until R is empty
    while |R| > 0 do
        // Identify best variable F* out of R (with minimum p-value)
        F* ← argmin_{F∈R} Test(T, F|S)
        // Select F* if it is conditionally dependent with T given S
        if Test(T, F*|S) ≤ α then
            S ← S ∪ {F*}
        end if
        // Drop all variables from R that are conditionally independent given S
        R ← {F : F ∈ R ∧ F ≠ F* ∧ Test(T, F|S) > α}
    end while
    return S
end function
```

```
function BackwardSelection(D, T, S, α)
    // Iterate until no more features can be removed from S
    while S changes do
        // Identify F* with maximum p-value conditional on S \ {F*}
        F* ← argmax_{F∈S} Test(T, F|S \ {F})
        // Remove F* if conditionally independent with T given S \ {F*}
        if Test(T, F*|S \ {F*}) > α then
            S ← S \ {F*}
        end if
    end while
    return S
end function
```

all variables which have not been selected yet, that is $\mathbf{R} = \mathbf{F} \setminus \mathbf{S}$. The backward phase employed afterwards is identical to the standard backward-selection algorithm (see function BackwardSelection in Algorithm 3). Depending on the number of additional runs K, this defines a family of algorithms, which we call **Forward Backward Selection with Early Dropping** (FBED^K), shown in Algorithm 3. The function OneRun shown in the bottom of Algorithm 3, performs one run until no variables remain in $\mathbf{R}$. Three interesting members of this family are the FBED^0, FBED^1 and FBED$^\infty$ algorithms. FBED^0 performs the first run until termination, FBED^1 performs one additional run and FBED$^\infty$ performs runs until no more variables can be selected. We will focus on those three algorithms hereafter.

The heuristic is inspired by the theory of Bayesian networks and maximal ancestral graphs [126, 136]. Similar heuristics have been applied by Markov blanket based algorithms such as the Max-Min Parents and Children (MMPC) algorithm [146] and HITON [5] successfully in practice and in extensive comparative evaluations [4]. These algorithms also remove variables from consideration, and specifically the ones that are conditionally independent given some **subset** of the selected variables. In contrast, FBED^K reconsiders variables dropped during previous runs, while existing methods do not. The connections of FBED^K to graphical models and Markov blankets are presented next.

4.2 Comparing the Theoretical Properties of FBED^K to FBS

Due to early dropping of variables, the distributions under which FBED^K and FBS perform optimally are not the same. For all versions of FBED^K, with the exception of FBED$^\infty$, it is relatively straightforward to construct examples where FBS is able to identify variables that can not be identified by FBED^K. We give an example for FBED^0. FBED^0 may remove variables that seem uninformative at first, but become relevant if considered in conjunction with other variables. For example, let $X = T + Y$, with T and Y being independent Gaussian random variables, and assume that T is the outcome for which variable selection is performed. When no variables have been selected (first iteration), X will be dependent with T, while Y will be independent of T as it does not give any information about T by itself, and thus will be dropped. However, after selecting X, Y becomes conditionally dependent again (as $T = X - Y$), but FBED^0 will not select it as it was dropped in the first iteration. Surprisingly, in practice this does not seem to significantly affect the quality of FBED^0. In contrast, FBED^0 often gives better results, while also selecting fewer variables than FBS (see Section 4.4.4).

As mentioned above, it is not clear how FBS and FBED$^\infty$ are related in the general case. What can be shown is that both identify a minimal set of variables, although the identified solutions may not necessarily be the same.

Definition 6 (Minimal Variable Set)**.** *Let* $\mathbf{F}$ *be the set of all variables and* $\mathbf{S}$ *a set of selected variables. We call a set of variables* $\mathbf{S}$ *minimal with respect to some outcome* T*, if:*

1. *No variable can be removed from* $\mathbf{S}$ *given the rest of the selected variables, that is,* $\forall F_i \in \mathbf{S}, T \not\perp F_i \mid \mathbf{S} \setminus F_i$ *holds.*

2. *Let* $\mathbf{R} = \mathbf{F} \setminus \mathbf{S}$*. No variable from* $\mathbf{R}$ *can be included in* $\mathbf{S}$*, that is,* $\forall F_i \in \mathbf{R}, T \perp F_i \mid \mathbf{S}$ *holds.*

Corollary 1. *Any set of variables* $\mathbf{S}$ *selected by FBS is minimal.*

Proof. See Appendix A.1. □

Corollary 2. *Any set of variables* $\mathbf{S}$ *selected by FBED*$^\infty$ *is minimal.*

Proof. See Appendix A.2. □

In words, a minimal set is a set such that no single variable can be included to or removed from using forward and backward iterations respectively, that is, it is a local optimum for stepwise algorithms. Note that, although no single variable is informative for T if looked at separately, there may be sets of variables that are informative if considered jointly. A simple example is if all variables are binary and $T = X \oplus Y$, where $\oplus$ is the logical XOR operator. In this case $\mathbf{S} = \emptyset$ is minimal, as neither X nor Y are dependent with T, even though the set $\{X, Y\}$ fully determines T. Thus, none of the algorithms gives a globally optimal solution in all distributions.

We next consider the special case in which distributions can be represented by Bayesian networks (BNs) or maximal ancestral graphs (DMAGs). We show that FBED^1 and FBED$^\infty$ identify the Markov blanket of a BN and DMAG respectively, assuming (a) that the distribution can be faithfully represented by the respective graph, and (b) that the algorithms have access to an **independence oracle**[1], which correctly determines whether a given conditional (in)dependence holds. This also holds for FBS but will not be shown here; proofs for similar algorithms exist [101, 147] and can be easily adapted to FBS. For FBED^0 it can be shown that it selects a superset of the variables that are adjacent to T in the graph; this can be shown using the fact that, under the Markov and faithfulness assumptions, adjacent variables are dependent with T given any subset of the remaining variables.

[1]Assuming access to an independence oracle allows one to analyze whether the strategy used by FBED^K for identifying a Markov blanket is correct; thus, in practice, any errors in the output are due to statistical errors of the tests and not due to the heuristics or strategy used by FBED^K. Furthermore, it allows one to analyze the asymptotic behavior of algorithms without parametric distributional assumptions (e.g., multivariate normality), but structural assumptions (e.g., faithfulness). Assuming a conditional independence oracle is a standard assumption for the theoretical analysis of Markov blanket and causal discovery algorithms (e.g., see [4, 136]). In practice, FBED^K will not have access to an oracle, but will perform conditional independence tests to decide (in)dependence. There exist tests that, in the sample limit, will correctly identify (in)dependence. Examples include the partial correlation test for multivariate Gaussian data, and the G-test [1] for multinomial data.

Theorem 1. *If the distribution can be faithfully represented by a Bayesian network, then FBED^1 identifies the Markov blanket of the target T.*

Proof. See Appendix A.3. □

Theorem 2. *If the distribution can be faithfully represented by a directed maximal ancestral graph, then FBED$^\infty$ identifies the Markov blanket of the target T.*

Proof. See Appendix A.4. □

4.3 Limitations and Practical Considerations

We have shown that FBED^K is able to solve the feature selection problem (that is, identify the Markov blanket of T) for distributions that are faithful to causal graphs. In practice, FBED^K may fail to identify the Markov blanket for several reasons. Naturally, in case the distribution can't be faithfully modeled with causal graphs, there is no guarantee of how close the solution will be to the optimal solution. However, previous comparisons show that forward selection performs as well as best subset selection, and is competitive with lasso [70], indicating that its solutions are reasonably good approximations to the best subset solution, which we also confirm in the experimental section. Another, more subtle issue is if the conditional independence tests used are not appropriate to capture the dependencies present in the distribution. For instance, if all relations are non-linear and linear tests are used, there is no guarantee that any of the important variables will be selected. However, this is an issue with all feature selection algorithms (and predictive algorithms in general) and is not specific to FBED^K. Finally, if sample size is too low, or if the significance level is not set appropriately, dependencies may be incorrectly labeled as independencies and vice versa. Again, this is a general problem with all algorithms and can be handled by increasing sample size (if possible) and by appropriately setting or tuning the significance level. For example, for the task of learning Bayesian networks from Gaussian data using the PC algorithm [136], [79] have shown that (under mild conditions) the significance level can be set in a way to ensure consistency asymptotically (see Theorem 1 in [79]). The problem of learning Bayesian networks and Markov blanket discovery are closely related, and such results can possibly be translated and used by algorithms such as FBED^K, but it is out of the scope of the current work.

We proceed with additional considerations regarding the sample size required to use FBED^K. FBED^K identifies the next variable to select conditional on all currently selected variables. Because of this, it can in principle take complex multivariate dependencies into consideration when selecting a variable. The complexity depends on the conditional independence test used (for example, non-linear tests can model more

complex relations than linear tests). However, there is a clear trade-off between the complexity of dependencies that can be identified, and the sample size required to do so. For instance, if all variables are binary, the G-test of conditional independence [1] can be used, which can identify any type of interaction between variables. In this case, the number of parameters increases exponentially with the number of selected variables: for k selected variables, the number of parameters is in the order of $O(2^k)$, and consequently, the number of samples required to have sufficient power also increases exponentially. Using linear models (for example, linear, logistic or Cox regression for continuous, categorical or time-to-event outcomes respectively), simpler, linear dependencies can be identified, and the number of samples required increases only linearly with the number of parameters. Rules of thumb for setting the minimum sample size for linear models are given in [67, 116, 153]. For binary logistic regression, one recommendation is to use at least $s = c/\min(p_0, p_1) \cdot k$ samples [116], where p_0 and p_1 are the proportion of negative and positive classes of T respectively, k is the number of parameters in the model and c is a user-set parameter, which is usually recommended to be between 5 and 20, with larger values leading to more accurate results. Another rule, called the STD rule, is presented in Section 5.3.1. Thus, multivariable methods like FBED^K should only be used when sufficient sample size is available; alternatively, one can use rules of thumb as stopping criteria (that is, to determine when to stop selecting variables).

Next, we make a few recommendations based on the above considerations; exact rules are hard to devise, as they depend on the specific problem at hand. In case sample size is very low (a few tens or hundreds of samples), sample-efficient methods like the max-min parents and children algorithm [146] (which condition only on small subsets of variables), information-theoretic feature selection methods [22] (which only condition on up to 1 variable), or univariate feature selection methods are more preferable than methods like FBED^K. Otherwise, we recommend using linear multivariable methods like OMP [39, 113], LASSO [139] or FBED^K with linear tests, and if sample size allows to also consider FBED^K using non-linear tests. Finally, we believe it is also worth considering robust tests [89] for FBED^K, as outliers often exist in practice and may negatively impact tests which do not take them into account.

4.4 Experimental Evaluation

In this section we evaluate FBED^K, and compare it to the standard FBS algorithm, feature selection with LASSO (called LASSO-FS hereafter) [139], the Max-Min Parents and Children algorithm (MMPC) [146], and no feature selection (NO-FS), which was used as the baseline method. We note that MMPC is designed specifically for low-sample size and high-dimensional settings, and thus may not perform optimally in the

Table 4.1: Binary classification datasets used in the experimental evaluation. n is the number of samples, p is the number of predictors and $P(T = 1)$ is the proportion of instances where $T = 1$.

Dataset	n	p	P(T = 1)	Type	Domain	Source
musk (v2)	6598	166	0.15	Real	Musk Activity Prediction	UCI ML Repository [41]
sylva	14394	216	0.94	Mixed	Forest Cover Types	WCCI 2006 Challenge [62]
madelon	2600	500	0.5	Integer	Artificial	NIPS 2003 Challenge [64]
secom	1567	590	0.93	Real	Semi-Conductor Manufacturing	UCI ML Repository M. McCann, A. Johnston
gina	3568	970	0.51	Real	Handwritten Digit Recognition	WCCI 2006 Challenge [62]
hiva	4229	1617	0.96	Binary	Drug discovery	WCCI 2006 Challenge [62]
gisette	7000	5000	0.5	Integer	Handwritten Digit Recognition	NIPS 2003 Challenge [64]
p53 Mutants	16772	5408	0.01	Real	Protein Transcriptional Activity	UCI ML Repository [37]
arcene	200	10000	0.56	Binary	Mass Spectrometry	NIPS 2003 Challenge [64]
nova	1929	16969	0.72	Binary	Text classification	WCCI 2006 Challenge [62]
dexter	600	20000	0.5	Integer	Text classification	NIPS 2003 Challenge [64]
dorothea	1150	100000	0.9	Binary	Drug discovery	NIPS 2003 Challenge [64]

datasets considered here. The reason we compare against it is because, it belongs in the same category of algorithms as FBED^K (that is, is also inspired by causal graphs).

We implemented all algorithms in Matlab except for LASSO, for which we used the glmnet implementation [120]. We used 12 binary classification datasets, with sample sizes ranging from 200 to 16772 and number of variables between 166 and 100000. The datasets were selected from various competitions [62, 64] and the UCI repository [41], and were selected to cover a wide range of variable and sample sizes. A summary of the datasets is shown in Table 4.1. All experiments were performed in Matlab, running on a desktop computer with an Intel i7-7700K processor and 32GB of RAM.

The remainder of this section is organized as follows. First, we describe in detail the experimental setup, that is, all algorithms used, their hyper-parameters, and how we performed model selection and performance estimation. We proceed by evaluating

how the running time, number of selected variables and predictive performance of FBED^K is affected by the number of runs K. Afterwards, we compare FBED^K to FBS, to show the effects of the early dropping heuristic. Next, we evaluate FBED^K in a realistic scenario with other feature selection methods, where hyper-parameters of the feature selection and classification algorithms are optimized. Then, we compare FBED^K and LASSO in terms of predictive ability when the number of variables to select is fixed so that all algorithms produce solutions of equal size. This is done for two reasons: (a) because LASSO tends to select many variables otherwise, giving it an advantage over FBED^K in terms of predictive performance, and (b) because this allows us to evaluate how well FBED^K orders the variables in comparison to LASSO. Finally, we compare FBED^0, FBED^1, FBED$^\infty$ and FBS on simulated data containing only irrelevant variables, investigating how is each algorithm is affected by multiple testing in terms of the falsely selected variables.

4.4.1 Experimental Setup

We present an overview of the experimental setup next. Additional details for each specific experiment are described in the respective section.

Feature selection algorithms

As selection criteria for FBED^K, FBS and MMPC we used a nested likelihood-ratio independence test based on logistic regression. For FBED^K and MMPC, the significance level α of the conditional independence test was set to $\{0.001, 0.005, 0.01, 0.05, 0.1\}$, covering a range of commonly used values, while for FBS we explored a total of 100 values, uniformly spaced in $[0.001, 0.01]$. For the K value of FBED^K we used $\{0, 1, \ldots, \infty\}$, while the maximum conditioning size $maxK$ of MMPC was set to $\{1, 2, 3, 4\}$. For LASSO-FS we set all parameters to their default values and set the maximum number of λ values, λ_{max}, to 100. Thus, we used 5 hyper-parameter combinations for each value K of FBED^K, 100 for FBS and LASSO-FS, and 20 for MMPC.

Unfortunately, for MMPC there were 2 datasets where not all hyper-parameter combinations were executed, as they were taking too long to terminate (see results about running time in Appendix D.1); we stopped execution if a time limit of 2 days was exceeded. Specifically, for the gisette and nova datasets MMPC was only executed with $maxK \leq 2$

We would like to point out that for a given value K for FBED^K, all solutions with fewer runs (smaller K) can be computed with minimal computational overhead, as the forward phases have already been computed and only the backward phases need to be performed separately. As the number of variables selected is relatively small, the computational cost of the backward phases is usually negligible. Thus, FBED^K

required a single execution with $K = \infty$ for a given a. Unfortunately, something similar can not be done with MMPC, and thus it has to be executed for each hyper-parameter value separately [2].

Predictive models

We used both, linear and non-linear predictive models. As linear models we used elastic net regularized logistic regression [169], using $\lambda_{max} = 100$ and the mixture parameter a set to $\{0, 0.25, 0.5, 0.75, 1\}$ ($a = 0$ corresponds to L2 regularization and $a = 1$ to L1 regularization), leading to a total of 500 hyper-parameter combinations. We remind the reader that regularization is important, especially after feature selection has been performed, in order to improve predictive performance due to inflated coefficients [56] (see also Section 2.4.1). As non-linear models we used Gaussian support vector machines (SVM) [34] and random forests (RF) [20]. For SVMs we used the LIBSVM [26] implementation, while for RFs we used the TreeBagger implementation in Matlab. The cost hyper-parameter C of SVMs was set to $\{2^{-10}, 2^{-9}, \ldots, 2^{9}\}$ (a total of 20 values), while the remaining hyper-parameters were set to their default values. For RFs the number of trees was set to 500, the minimum leaf node size was set to $\{1, 5, 9\}$ and the number of variables to split at each node was set to $\{0.5, 1, 1.5\} \cdot \sqrt{p}$ (9 combinations in total). In a recent large-scale evaluation between many classifiers (179 from 17 different families) on the whole UCI data base (121 datasets at the time), it has been shown that RFs and SVMs with Gaussian kernels are the top two performers [50], and thus we did not consider other non-linear methods.

Linear vs non-linear models

Throughout the section, we will report results obtained by using only linear models or a combination of linear and non-linear models. The former is done to evaluate the ability of the feature selection methods of identifying features that are linearly (or possibly monotonically) related to the outcome. The reason for that is that all evaluated methods can only identify such types of dependencies; we note that all algorithms (except for LASSO) can be trivially adapted to also handle non-linear dependencies by using an appropriate conditional independence test. Non-linear models were also considered to better simulate a realistic scenario, as such methods would be used in a typical analysis. Furthermore, it is interesting to see whether there are any significant differences between linear and non-linear modeling for any of the considered feature selection algorithms.

[2] This is only partially true, as for FBED^K, FBS and MMPC we implemented a caching mechanism to avoid fitting the same logistic regression model more than once. This mechanism was not used however for experiments measuring the running time of the algorithms.

Model selection and performance estimation protocols

Ideally, we would like to evaluate each feature selection algorithm using an optimal predictive model, in order to measure how informative the selected features are. As an optimal model is not available in practice, we approximate this by using a variety of predictive algorithms as well as multiple hyper-parameter value combinations for each (see above), and perform hyper-parameter optimization (also called tuning or model selection) to find a good approximate model; interested readers may refer to [51,150] for more details.

We proceed with a description of the model selection and performance estimation protocols we used. As the performance metric we optimize and report the area under the ROC curve (AUC). For model selection and performance estimation we used a 60/20/20 stratified split of the data, using 60% as a training set, 20% as a validation set and the remaining 20% as a test set. A hyper-parameter configuration is defined as a combination of a feature selection algorithm and its hyper-parameters, as well as a modeling algorithm and its hyper-parameters. Given a set of configurations, the best one is chosen by training models for all of them on the training set and selecting the configuration of the model with the highest performance on the validation set. Finally, the predictive performance of that configuration is obtained by training a final model on the pooled training and validation sets, and evaluating it on the test set. To account for the variation due to the data splitting, we repeated this procedure multiple times for different splits and report averages over repetitions. For datasets with more than 1000 samples the number of repetitions was set to 10, and to 50 for the rest.

4.4.2 Effect of the Number of Runs K

We performed an experiment to measure the effect of K on the running time, number of selected variables and predictive performance of FBED^K. For the running time and number of selected variables we executed FBED^K once for each hyper-parameter value on the complete datasets, while for the predictive performance we used the model selection and performance estimation protocols described previously and report averages over multiple repetitions of the experiment.

Figure 4.1 shows how the number of runs K affects the running time and the number of selected variables. Vertical lines show the value of K for which the algorithm has converged (that is, after that point more runs do not select any more variables). We can see that running time increases almost linearly with an increasing number of runs K, meaning that any additional run has a roughly linear computational cost with respect to the size of the dataset. Furthermore, convergence is typically achieved in less than 10 runs, although for a few cases up to 16 runs are required. As expected, the number of selected variables increases with K, as well as with the threshold α. In the majority of cases however, most progress is made in the first few runs, and further

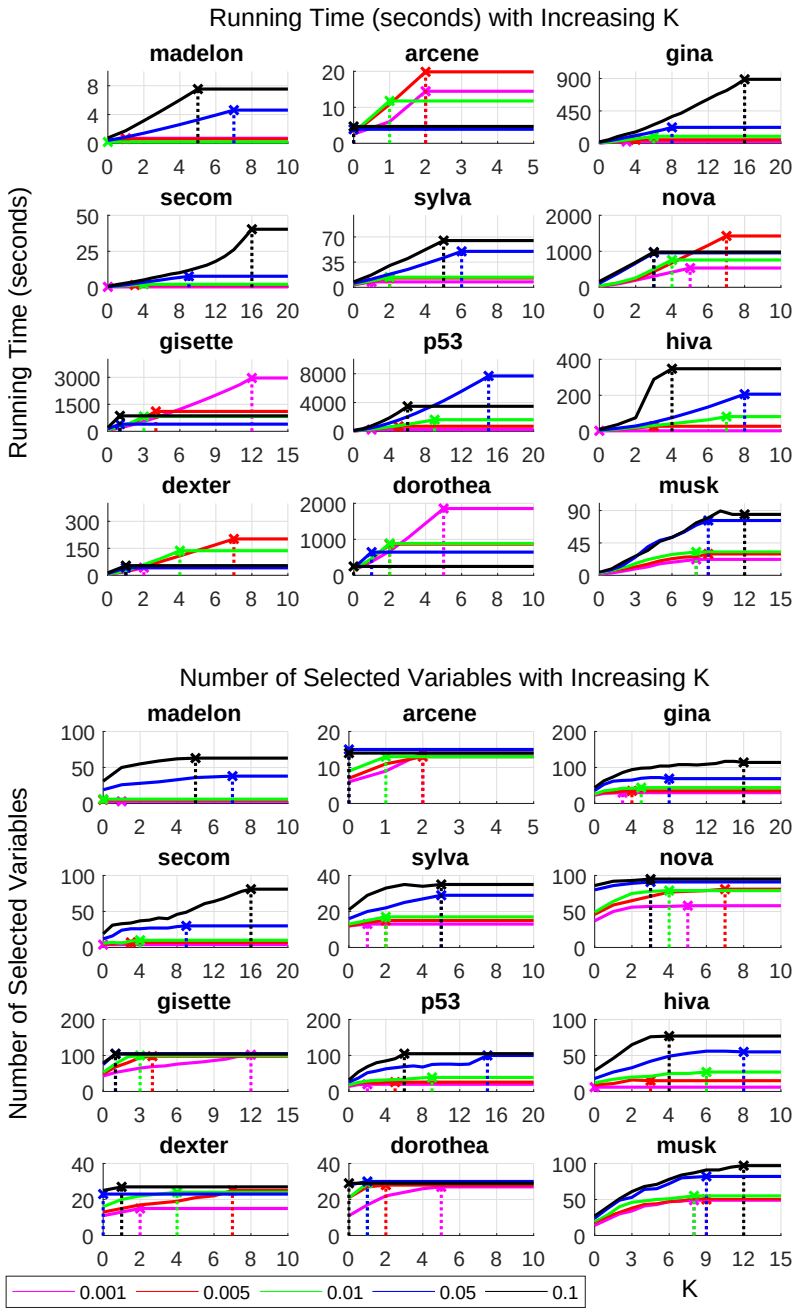


Figure 4.1: The figure shows how the running time (top) and the number of selected variables (bottom) vary with an increasing number of runs K for different values of the threshold parameter α. The vertical lines indicate the value of K for which FBED^K has converged. Running time increases almost linearly with K. Most progress is made in the first few runs, and additional runs increase running time while only selecting a few more variables.

runs increase the number of selected variables only marginally. *Based on those results, we recommend considering relatively small values of K, up to $K < 10$.*

Figure 4.2 shows how the area under the ROC curve (AUC) varies with an increasing number of runs K, for 5 different values of the threshold parameter a of FBED^K. We observe that, although AUC often tends to increase with K, this is not always the case. For instance, for the nova and dexter datasets, AUC actually decreases with K for some values of a. The maximum AUC is typically achieved with relatively small values of K, further suggesting that considering higher values for K is not necessary. Also, there are no clear relationships between AUC, the value of a and the type of predictive models used. Depending on the dataset different values of a or predictive models may be optimal. *Thus, in practice we recommend considering several combinations of a and K.* Optimizing over both a and K will be considered in Section 4.4.4.

4.4.3 FBED^K vs FBS

In this section we compare FBED^K to the standard FBS algorithm in terms of predictive performance, number of selected variables and running time. The algorithms were compared on the same hyper-parameters (for example, FBS vs FBED^0 with $a = 0.01$); results when also optimizing over hyper-parameters are shown in Section 4.4.4. Model selection and performance estimation was performed for each feature selection algorithm and each hyper-parameter value separately, following the procedure described in the experimental setup. *The main goal of this comparison is to show that FBED^K and FBS perform similarly for the same hyper-parameters, with the former being faster.* A summary of the results is presented next.

Figure 4.3 shows how the algorithms compare in terms of predictive performance, number of selected variables and running time. Each column shows the distribution of the respective metric across all thresholds and datasets, as well as the mean and median values. The difference in AUC is computed is AUC(FBED^K) - AUC(FBS), the relative number of selected variables is computed as the ratio of variables selected by FBED^K compared to FBS, and the speed-up is computed as Time(FBS) / Time(FBED^K). The y-axis corresponds to different values of K used by FBED^K. Only the first few values ($K \leq 9$), as well as the last one ($K = \infty$) are shown, as the left-out ones were almost identical to $K = \infty$.

Regarding predictive performance, FBED^K performs as good as FBS on average, irrespective of the type of predictive models used. For FBED^K with $K < 3$, the average difference in AUC is less than 1% while the median difference is close to 0, and for all other K the performance is almost identical to FBS. We note that all those lower-performing cases are also the ones where FBED^K selected much fewer variables than FBS. In terms of the number of selected variables, FBED^K produces smaller solutions

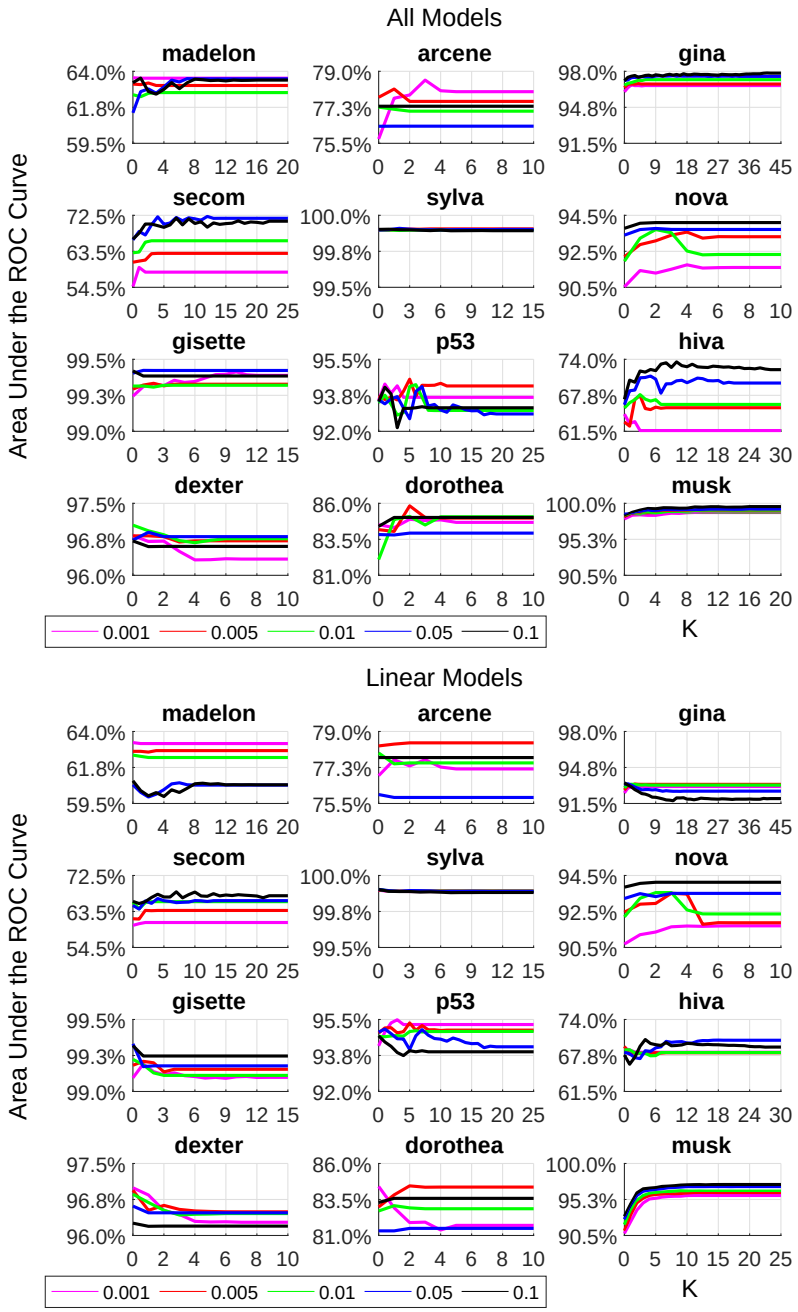


Figure 4.2: The figure shows how the AUC varies with an increasing number of runs K for different values of the threshold parameter α, using non-linear and linear models (top) or linear models only (bottom). There is no clear pattern for which thresholds or values of K to prefer, but the optimal values depend on the specific dataset, as well as on the predictive models used. In most cases only a few runs are required to achieve maximal AUC.

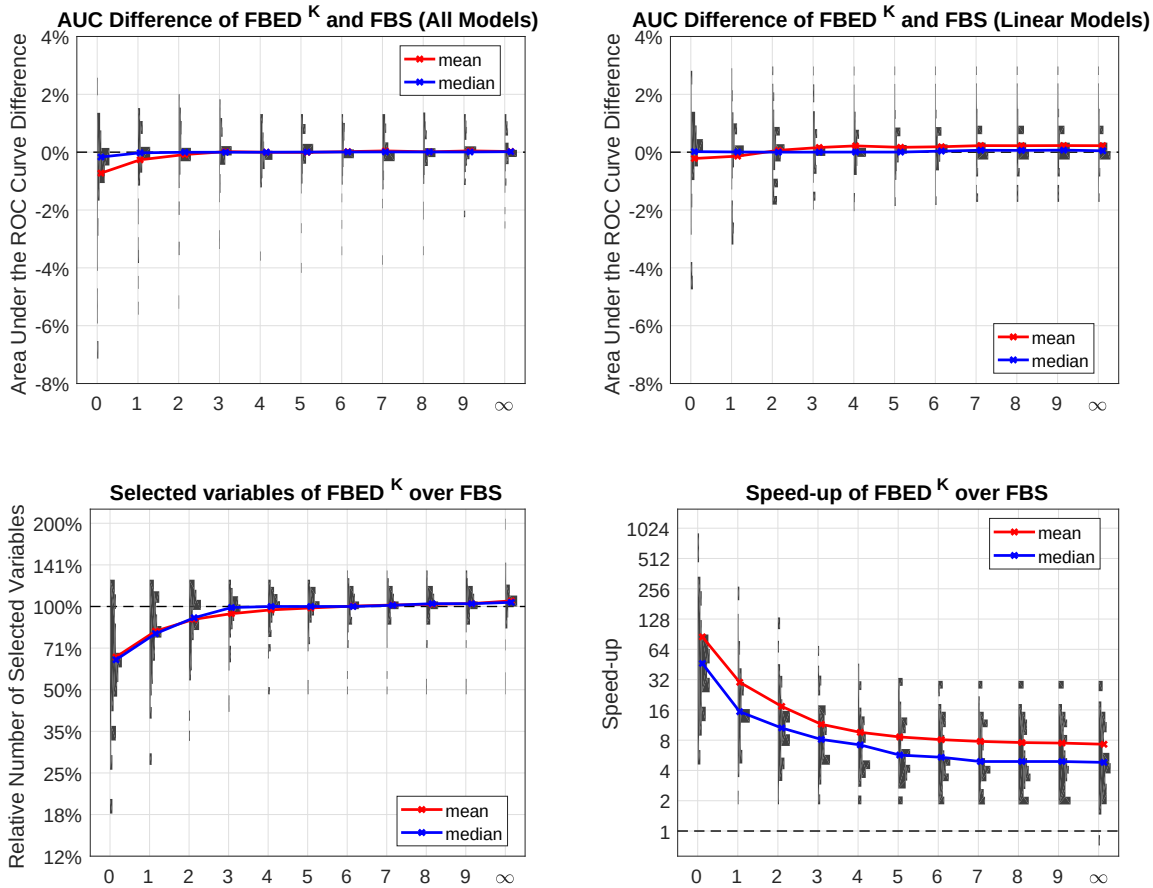


Figure 4.3: The x-axis of the figures on the top row shows the difference in AUC between FBED^K and FBS, with positive values indicating that FBED^K performs better than FBS. The AUC of the top left figure is computed by optimizing over all models, while for the one of the top right figure only linear models were considered. The relative number of selected variables (bottom left) shows the number of variables selected by FBED^K compared to the ones selected by FBS. The speed-up (bottom right) is computed as the one obtained by FBED^K over FBS. For all cases, the distribution over all thresholds and datasets is shown, as well as the mean and median values. The y-axis on all figures is the value of K used by FBED^K. Overall, FBED^K has a virtually identical performance with FBS, while being on average between 1 and 2 orders of magnitude faster.

for $K < 3$, and tends to select the same number as FBS with increasing K. Finally, in terms of running time, FBED^K is significantly faster than FBS, being about 1-2 orders of magnitude faster on average in all cases.

An interesting case is for FBED^3, where the number of selected variables and AUC between both algorithms is almost identical, while being around 10 times faster than FBS. If speed and small solutions are important, FBED^0 and FBED^1 are good choices,

as they are ~30-100 times faster than FBS, selecting only ~70%-80% of the variables with a minimal drop in AUC. Therefore, if the number of variables is high, FBED^0 and FBED^1 are preferable due to their low computational lost. Furthermore, in low sample settings the smaller solutions of FBED^0 and FBED^1 are important, as selecting many variables leads to loss of power and overfitting. If on the other hand the sample size is large and the number of variables is relatively small, both FBS and FBED^K with higher values of K are reasonable choices, with the latter being more attractive, as it is around 1 order of magnitude faster and thus can scale to higher variable sizes.

4.4.4 Comparison of FBED^K with other Feature Selection Methods

We performed an experiment where we also optimize over the hyper-parameter values of feature selection algorithms. *The main objective of this comparison is to compare FBED^K to other feature selection algorithms in a realistic scenario, where hyper-parameter values are optimized.* For this comparison we focus on the predictive performance and number of selected variables; additional results showing the running time of each algorithm can be found in Appendix D.1.

Setup

For FBED^K optimization is performed over the threshold α and the number of runs K. We examine four versions of FBED^K: FBED^0, FBED$^{\leq 1}$, FBED$^{\leq 3}$ and FBED$^{\leq \infty}$. FBED$^{\leq K}$ means that optimization was performed for all results up to K runs. Thus, the number of hyper-parameter configurations used were 5, 10, 20 and around 50 (for most cases) for FBED^0, FBED$^{\leq 1}$, FBED$^{\leq 3}$ and FBED$^{\leq \infty}$ respectively. The hyper-parameter values for FBS, MMPC and LASSO-FS are the ones described in Section 4.4.1 (a total of 100, 20 and 100 respectively). We also included results when no feature selection was performed (NO-FS). Finally, we remind the reader that we used two sets of classification algorithms and hyper-parameters, one containing only linear algorithms (elastic net regularized logistic regression) and one also containing non-linear ones in addition to the linear ones (Gaussian support vector machines and random forests). For brevity, we will refer to linear models as LM and to the combination of linear and non-linear models as NLM hereafter.

Results

A summary of the results averaged over repetitions, measuring the AUC and number of selected variables is shown in Tables 4.2 and 4.3. For each algorithm, we computed a score which is the average rank of that algorithm over all datasets. The final rank of an algorithm is then computed based on that score. We used a bootstrap-based procedure

Table 4.2: Area under the ROC curve and number of selected variables for all feature selection algorithms using linear and non-linear models. The results are obtained after optimizing the hyper-parameters of the feature selection and modeling algorithms. Bold and italic entries denote that the method is significantly better or worse than all other feature selection methods (excluding NO-FS) respectively. The score is the average rank of each method over all datasets and the final rank is computed using those scores. Methods that select more variables tend to also perform better (Spearman correlation between AUC and variable rankings is -0.976).

	Algorithm	musk	sylva	madelon	secom	gina	hiva	gisette	p53	arcene	nova	dexter	dorothea	Score	Rank
AUC (all models)	$FBED^0$	*98.5*	99.9	63.4	63.7	*97.0*	67.4	99.4	93.3	77.5	93.6	96.7	83.8	6.33	8
	$FBED^{\leq 1}$	98.7	99.9	63.3	63.0	97.3	70.9	99.4	93.5	76.9	94.0	96.8	83.5	6.17	7
	$FBED^{\leq 3}$	99.2	99.9	63.0	68.0	97.3	68.8	99.4	93.2	77.3	93.6	96.8	84.3	5.46	6
	$FBED^{\leq \infty}$	99.5	99.9	63.6	67.6	97.6	72.3	99.4	93.5	77.3	93.6	96.8	84.9	4.71	4
	FBS	99.5	99.9	64.8	69.5	97.5	69.1	99.4	94.1	77.2	92.4	*95.9*	84.3	5.00	5
	MMPC	98.9	99.9	65.1	63.4	98.1	68.3	99.6	93.1	79.6	**96.7**	97.3	**91.7**	4.00	3
	LASSO-FS	**99.9**	99.9	**82.3**	69.8	98.2	74.2	**99.7**	94.2	**82.4**	96.1	97.2	89.0	2.58	2
	NO FS	99.9	99.9	82.8	71.9	98.3	74.3	99.6	94.7	90.0	96.6	98.2	94.6	1.75	1
Selected Variables	$FBED^0$	**22.1**	13.4	8.4	15.0	**32.9**	**18.8**	72.9	24.2	8.7	66.5	17.4	21.6	1.33	1
	$FBED^{\leq 1}$	35.6	15.2	8.2	18.6	47.7	34.1	80.2	23.5	9.6	71.9	18.0	23.2	2.79	2
	$FBED^{\leq 3}$	47.1	21.1	14.0	24.4	56.5	43.5	80.2	26.4	9.3	72.1	18.5	22.3	3.63	4
	$FBED^{\leq \infty}$	77.6	25.5	14.9	41.1	105.8	75.1	79.4	33.7	9.3	72.3	18.7	22.4	4.79	5
	FBS	76.1	21.1	19.6	28.4	78.7	33.8	**65.0**	32.6	11.0	71.6	16.9	22.0	3.46	3
	MMPC	42.5	20.7	17.0	16.2	155.4	51.7	*388.2*	68.8	*43.9*	*879.7*	*187.2*	*445.1*	5.42	6
	LASSO-FS	*138.2*	*43.0*	*495.4*	*133.3*	*406.4*	*306.6*	187.5	*161.9*	29.6	349.8	97.9	70.1	6.58	7
	NO FS	166.0	213.0	500.0	468.0	970.0	1617.0	4948.0	5408.0	9955.0	11853.0	9988.0	88215.0	8.00	8

to compute the probability of an algorithm being significantly better or worse than all competitors, and used a threshold of 95%. The procedure is described in more detail in Appendix B.1. In the tables, algorithms that are statistically significantly better than all others are shown in bold, whereas algorithms that are worse than the rest are shown in italic.

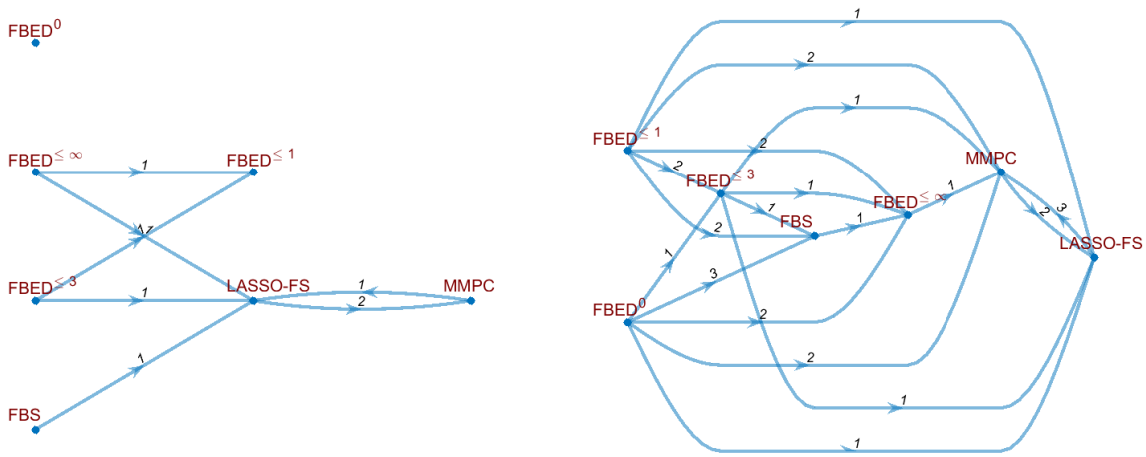
Overall, performing no feature selection has the highest AUC. Out of all feature selection methods, LASSO-FS offers the best predictive performance, statistically significantly outperforming the rest in 4 datasets. MMPC outperforms the rest in 2 and 3 datasets using NLM and LM respectively. In two cases $FBED^0$ is significantly worse than the rest (musk and gina), while FBS is the worst in 1 dataset. However, in terms of the number of selected variables, LASSO-FS selects statistically significantly more in 7 and 5 cases for NLM and LM, while MMPC selects the most variables in 5 datasets for both NLM and LM. $FBED^K$ and FBS on the other hand tend to select fewer variables. Thus, *there is a clear trade-off between model interpretability (number of selected variables) and predictive performance (AUC)*. Specifically, there is a -0.976 and -0.595 Spearman correlation between the AUC and selected variables ranks for NLM and LM respectively. For that reason, we performed an additional experiment, comparing the AUC between $FBED^K$, FBS and LASSO-FS by constraining the total number of variables to select, presented in Section 4.4.5.

Table 4.3: Area under the ROC curve and number of selected variables for all feature selection algorithms using linear models. The results are obtained after optimizing the hyper-parameters of the feature selection and modeling algorithms. Bold and italic entries denote that the method is significantly better or worse than all other feature selection methods (excluding NO-FS) respectively. The score is the average rank of each method over all datasets and the final rank is computed using those scores. Methods that select more variables tend to also perform better (Spearman correlation between AUC and variable rankings is -0.595), but the effect is not as strong as the one of the previous results (Table 4.2).

	Algorithm	musk	sylva	madelon	secom	gina	hiva	gisette	p53	arcene	nova	dexter	dorothea	Score	Rank
AUC (linear models)	FBED^{0}	*93.0*	99.9	62.3	65.1	93.3	69.4	99.3	94.0	78.0	93.3	96.6	81.7	4.58	4
	FBED$^{\leq 1}$	94.6	99.9	62.2	62.6	93.3	68.3	99.3	94.4	78.0	93.7	96.6	83.5	4.83	5
	FBED$^{\leq 3}$	96.5	99.9	62.0	64.0	93.2	68.3	99.2	95.0	77.7	93.4	96.5	83.5	5.38	6
	FBED$^{\leq \infty}$	97.1	99.9	62.0	63.8	92.1	69.0	99.2	95.1	77.7	93.4	96.5	83.3	5.71	7
	FBS	97.1	99.9	62.2	67.0	92.5	67.3	99.2	94.6	75.5	92.5	*95.6*	83.3	5.92	8
	MMPC	93.8	**99.9**	62.3	64.3	93.4	68.3	99.3	95.3	75.6	**96.7**	96.6	**89.3**	3.50	3
	LASSO-FS	**97.5**	99.9	62.1	64.8	92.1	70.7	**99.6**	95.4	**80.6**	95.8	**97.3**	86.9	3.17	2
	NO FS	97.6	99.9	59.5	66.6	91.2	71.8	99.6	95.8	87.3	96.5	98.3	91.7	2.92	1
Selected Variables	FBED^{0}	**22.7**	**14.1**	8.0	15.6	**34.2**	**20.4**	71.5	23.2	**8.6**	**66.5**	17.5	21.0	1.21	1
	FBED$^{\leq 1}$	35.2	16.8	10.6	18.0	46.9	31.2	79.3	25.5	9.4	71.4	17.9	22.0	2.71	2
	FBED$^{\leq 3}$	51.4	23.3	11.6	26.4	51.9	40.3	83.9	32.8	9.7	71.6	18.3	21.2	3.88	4
	FBED$^{\leq \infty}$	86.7	25.3	11.6	47.0	114.1	70.2	84.0	33.8	9.7	71.6	18.3	21.3	4.71	5
	FBS	79.5	22.8	9.0	34.7	80.1	29.3	**65.0**	34.6	11.0	72.4	16.7	22.0	3.63	3
	MMPC	43.3	34.2	8.0	17.7	137.8	36.1	*389.0*	92.2	*43.0*	*879.7*	*193.4*	*363.2*	5.29	6
	LASSO-FS	*144.5*	*57.8*	47.2	93.5	*369.3*	*285.5*	206.5	*208.0*	25.7	307.8	87.0	74.6	6.58	7
	NO FS	166.0	213.0	500.0	468.0	970.0	1617.0	4948.0	5408.0	9955.0	11853.0	9988.0	88215.0	8.00	8

A strong outlier in the NLM case is the difference in performance of LASSO-FS compared to the other methods on the madelon dataset, where LASSO-FS reaches an AUC that is 17.3 – 19.5% higher, which is also close to the AUC of NO-FS. The main reason why all methods fail is because the madelon dataset has been constructed in a way that makes it hard for linear methods (FBED^{K}, FBS and MMPC using the logistic regression test). Specifically, the outcome variable has been artificially constructed to be a XOR-type problem of 5 variables [65]. Further evidence for the hardness of this problem is the fact that using LM and NO-FS achieves an AUC of only 59.5 (even lower than all feature selection methods). LASSO-FS, although also linear, is able to pick up the signal as it basically performs no feature selection, selecting 495.4 out of 500 variables on average. This happens because LASSO-FS explores up to 100 values for λ, some of which correspond to very dense solutions. In contrast, due to the experimental setup, none of the remaining methods selects that many variables in this case.

An interesting observation is the fact that FBED^{0} and FBED$^{\leq 1}$, the forward selection methods selecting the fewest variables, are ranked higher in terms of AUC than FBED$^{\leq 3}$, FBED$^{\leq \infty}$ and FBS when using LM. Thus, they are better suited to pick out linear trends in the data, producing solutions that are smaller and more predictive



(a) Dominating Relationships (All Models) (b) Dominating Relationships (Linear Models)

Figure 4.4: The figures show how often a feature selection method dominates another (that is, has a higher AUC while selecting fewer variables), using non-linear models (left) and linear models (right). An edge from method A to B with weight w indicates that A dominates B in w datasets. Except for FBED$^{\leq\infty}$ for linear models, which gets dominated by FBS in 1 dataset, methods in the FBED^K family are never dominated by FBS, MMPC or LASSO-FS, while typically dominating them in 1-3 datasets.

compared to algorithms of the same type. Using NLM gives an even bigger advantage to methods selecting more variables, as this increases the chance to also capture some non-linear signals in the data.

Finally, we performed a statistical test between all pairs of methods to identify cases where a method outperforms others, both in terms of AUC and number of selected variables. Again, we used a bootstrap-based test, computing the joint probability that method A has a higher AUC and selecting fewer variables than method B, using the same procedure as described in Appendix B.1. A method is considered to dominate another, if that probability is higher than 95%. The results are summarized in Figure 4.4. Each node corresponds to a feature selection method, and a directed edge from method A to B with weight w denotes that A dominates B in w datasets. We observe that, except for a single case where FBED$^{\leq\infty}$ gets dominated by FBS in 1 dataset for LM, FBED^K is never dominated by any other method, neither for the NLM nor for the LM case. In both NLM and LM cases, FBED^K dominates the competitors in 1-3 cases, while LASSO-FS and MMPC only dominate each other in 1-3 cases. Especially interesting is the fact that for NLM, FBED$^{\leq 3}$, FBED$^{\leq\infty}$ and FBS (that is, the forward-selection algorithms typically selecting the most variables) are the only algorithms that dominate others, while also not getting dominated. On the other hand, using LM

only FBED^0 and FBED$^{\leq 1}$ both dominate others and are not getting dominated. This agrees with the observation made before, that FBED^0 and FBED$^{\leq 1}$ are particularly well suited to identify compact and linearly predictive solutions.

Overall, *there is no clear winner, and the choice depends solely on the goal*. If the goal is predictive performance, LASSO-FS or MMPC are clearly preferable. If on the other hand one is interested in interpretability, then methods from the FBED^K family with small values of K are preferable. Regarding FBS, there is no scenario where it is preferable over one of the other algorithms. We must note that those results are somewhat artificial, as the performance of FBED^K and FBS highly depends on the hyper-parameter values chosen for the experiment, while LASSO-FS is not as sensitive to those choices. Furthermore, the fact that hyper-parameters are optimized based on performance naturally tends to favor methods that select more variables, putting LASSO-FS at a disadvantage in terms of interpretability.

4.4.5 Fixing the Number of Selected Variables

As confirmed by the previous experiment, there are two main trade-offs for feature selection algorithms: (a) the number of selected variables, with fewer variables leading to more interpretable results, and (b) the predictive ability of the selected variables, with more variables typically leading to better results. In general, algorithms that select more variables also tend to perform better in terms of predictive performance. Because of that, *we performed a comparison where algorithms are forced the select the same number of variables. That way, the predictive performance of algorithms can be compared on equal footing*.

We will compare FBED^K and LASSO-FS, when both are limited to select the same number of features. FBS is not included in the comparison, as we have already shown in Section 4.4.3 that FBS and FBED^K exhibit similar predictive performance with a similar number of selected features, with FBED^K being orders of magnitude faster. MMPC was not included, because neither MMPC nor FBED^K allow to set the number of features to select. In order to perform the comparison, we executed FBED^K for multiple hyper-parameter values (the ones given in the experimental setup) and then executed LASSO-FS with the constraint to select the same number of variables as FBED^K did. As it was not always possible to select the exact same number of variables, we identified the solution of LASSO-FS with at least as many variables as FBED^K. Except for a few cases where LASSO-FS selected 1 more variable than FBED^K, both methods selected the same number of variables. As a final comment, we note that the above experiment does not favor FBED^K over LASSO-FS, as no optimization over its hyper-parameter values is performed. The reason we did not use a fixed number of variables M to select is because there is no easy way to select exactly M variables

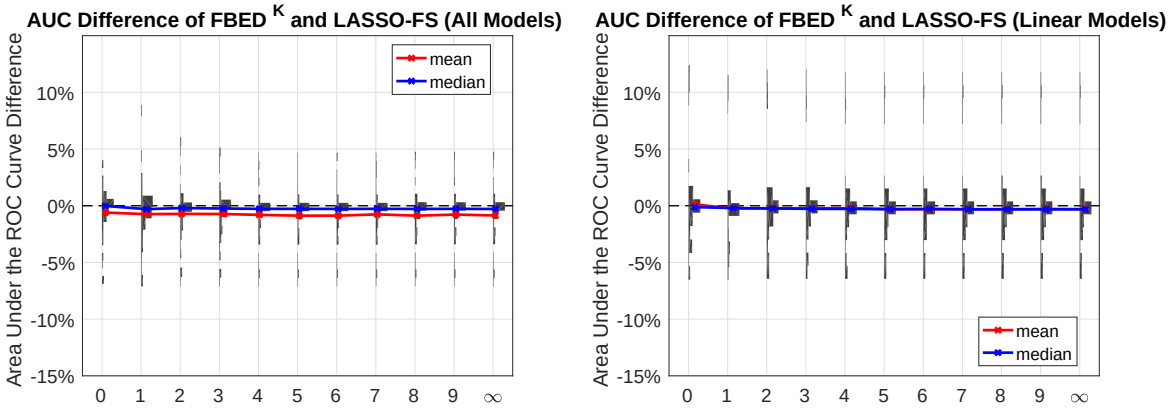


Figure 4.5: **LASSO-FS with limit on selected variables:** The x-axis shows the distribution of the difference in AUC of FBED^K and LASSO-FS, with positive values indicating that FBED^K performs better. The y-axis corresponds to value of K used by FBED^K. The mean and median values are shown in red and blue respectively. The average difference using non-linear models is 0.78%, and 0.23% when using only linear models. The difference can be explained by the arcene dataset, where LASSO-FS outperforms FBED^K even when selecting the same number of variables. A more detailed explanation is given in the main text.

using FBED^K.

The comparison was performed similarly to the comparison between FBED^K and FBS in Section 4.4.3. Specifically, for a given K and threshold a, we computed the difference in AUC obtained by FBED^K and LASSO-FS. Thus, the only hyper-parameter optimization performed was over predictive models and not over the hyper-parameters of FBED^K. The results are shown in Figure 4.5. The x-axis corresponds to the difference in AUC between FBED^K and LASSO-FS, while the y-axis indicates the value K of FBED^K. We can see that, overall, both methods perform very similarly, regardless of whether linear models or non-linear models were used. For non-linear models, LASSO-FS outperforms FBED^K on average (over all K and a) by 0.78%, while using linear models the difference drops to 0.23%. In terms of median difference in AUC, a metric which is more stable with the respect to the datasets and hyper-parameter values used, both algorithms perform almost identically. Those results also agree with previous comparisons between forward selection and LASSO-FS [70].

We investigated the difference in performance and found that it can be attributed to the arcene dataset. By removing this dataset, LASSO-FS outperforms FBED^K by 0.32% on average using non-linear models, while for linear models FBED^K performs better by 0.22%. Note that, arcene is the dataset which contains the fewest number of samples, while also containing a large number of variables. It contains 200 samples and 10000 variables, and only 120/160 are used for training and validation respectively.

Theoretical results by [111] show that LASSO performs well in settings with low sample size and many irrelevant variables, as is the case for the arcene dataset. One possible explanation for the lower performance of FBED^K on arcene is that forward selection based procedures use more effective degrees of freedom (see Figure 1 in [70]), thus requiring more sample size to have sufficient statistical power to pick up weak signals. It would be interesting to study this effect in more depth, but it is out of the scope of this work. In summary, *LASSO-FS and FBED^K perform similarly when the number of variables to select is the same*.

4.4.6 Simulation Study on the Multiple Testing Problem

The idea of early dropping of variables used by FBED^K does not only reduce the running time, but also reduces the problem of multiple testing, in some sense. Specifically, it reduces the number of variables falsely selected due to type I errors. In general, the number of false selections is related to the total number of variables considered in all forward iterations. Thus, the effect highly depends on the value of K used by FBED^K, with higher values of K leading to more false selections. We show this for FBED^0 by considering a simple scenario, where none of the candidate variables are predictive for the outcome. Then, in the worst case, FBED^0 will select about $\alpha \cdot p$ of the variables on average (where α is the significance level), since all other variables will be dropped in the first iteration. This stems from the fact that, under the null hypothesis of conditional independence, the p-values are uniformly distributed. In practice, the number of selected variables will be even lower, as FBED^0 will keep dropping variables after each variable inclusion. On the other hand, FBS may select a much larger number of variables, since each variable is given the chance to be included in the output at each iteration and will often do so, simply by chance.

We performed a small simulation to investigate the behavior of FBED^0, FBED^1, FBED$^\infty$ and how they compare to FBS. We generated 500 normally distributed datasets with 1000 samples each, a uniformly distributed random binary outcome, and considered different variable sizes $p \in \{100, 200, 300, 400, 500\}$ and 5 significance levels α uniformly spaced in $[0.01, 0.1]$. All variables are generated randomly, and there is no dependency between any of them. Thus, a false positive rate of about α is expected, if no adjustment is done to control the false discovery rate. For each setting, we computed the ratio of false positives with respect to the expected number of false positives.

Figure 4.6 (top) shows how the ratio varies for sample sizes 100 and 500 with increasing α, and Figure 4.6 (bottom) shows how the ratio varies for α 0.01 and 0.1 with increasing number of variables. In all cases, FBED^0 and FBED^1 select fewer false positives than expected, and their behavior improves both with increasing α and

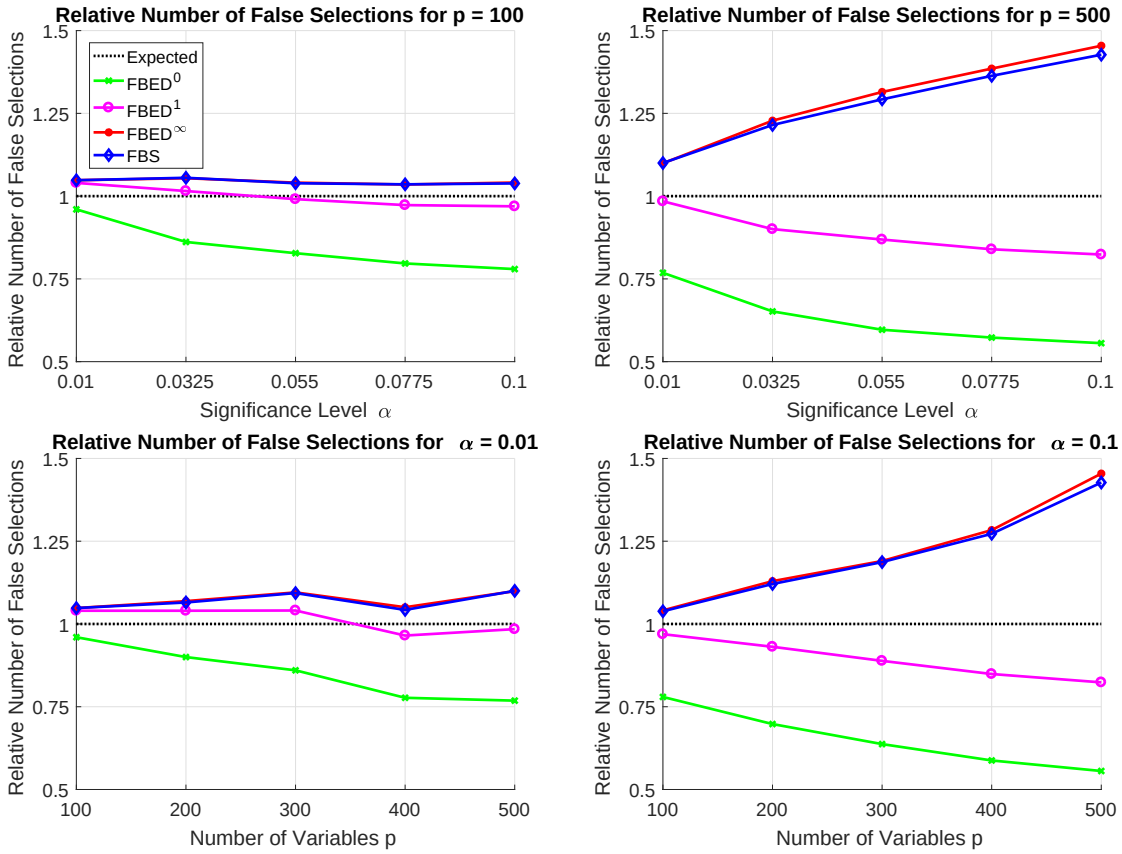


Figure 4.6: The figures show the relative number of false selections by each algorithm on randomly generated data. The expected number of false selections is $a \cdot p$, where a is the significance level and p the number of variables. The numbers are computed as the ratio between the average number of selected variables to the expected false positives. FBED^0 and FBED^1 typically select fewer variables than expected, and their behavior improves with increasing a and p. FBED$^\infty$ and FBS on the other hand select more false positive variables, getting worse with larger values of a or on datasets with more variables.

number of variables. FBED$^\infty$ and FBS perform almost identically, and tend to select more variables. We also observe that the number of false positives increases both with a and with the number of variables. Thus, *in case one is interested to limit the number of false selection, we recommend running FBED^K with a small value of K.*

Chapter 5
Extending FBED for Big Data of High Dimensionality

We introduced the Forward-Backward selection with Early Dropping algorithm, to speed-up the standard Forward-Backward Selection. Next, we will further extend it for Big Data with high sample volume and high dimensionality, using heuristics that take advantage of large sample sizes, by avoiding examining all samples unless necessary. Furthermore, in order to efficiently parallelize it, we will use statistical meta-analysis techniques that allow it to compute statistics and p-values with minimal communication overhead. All of the above, combined with the early dropping heuristic, allow the algorithm to scale super-linearly with sample size, and linearly with feature size and number of available cores, enabling it to deal will massive datasets.

5.1 Massively Parallel Forward-Backward Algorithm

We provide an overview of our algorithm, called Parallel, Forward-Backward with Pruning (**PFBP**), an extension of the Forward-Backward Selection with Early Dropping (FBED^K) algorithm. PFBP is presented in "evolutionary" steps where successive enhancements are introduced in order to make computations local or reduce computations and communication costs; the complete algorithm is presented in Section 5.1.4. To evaluate predictive performance of candidate features we use the p-values of conditional independence tests, as described in Section 2.4.1. We assume the data are provided in the form of a 2-dimensional matrix $\mathcal{D}$ where rows correspond to training instances (samples) and columns to features (variables), and one of the variables is the target variable T. Physically, the data matrix is partitioned in sub-matrices $D_{i,j}$ and stored in a distributed fashion in *workers* in a cluster running Spark [162] or similar platforms. Workers perform in parallel local computations on each $D_{i,j}$ and a *master* node performs the centralized, global computations.

5.1.1 Data Partitions in Blocks and Groups and Parallelization Strategy

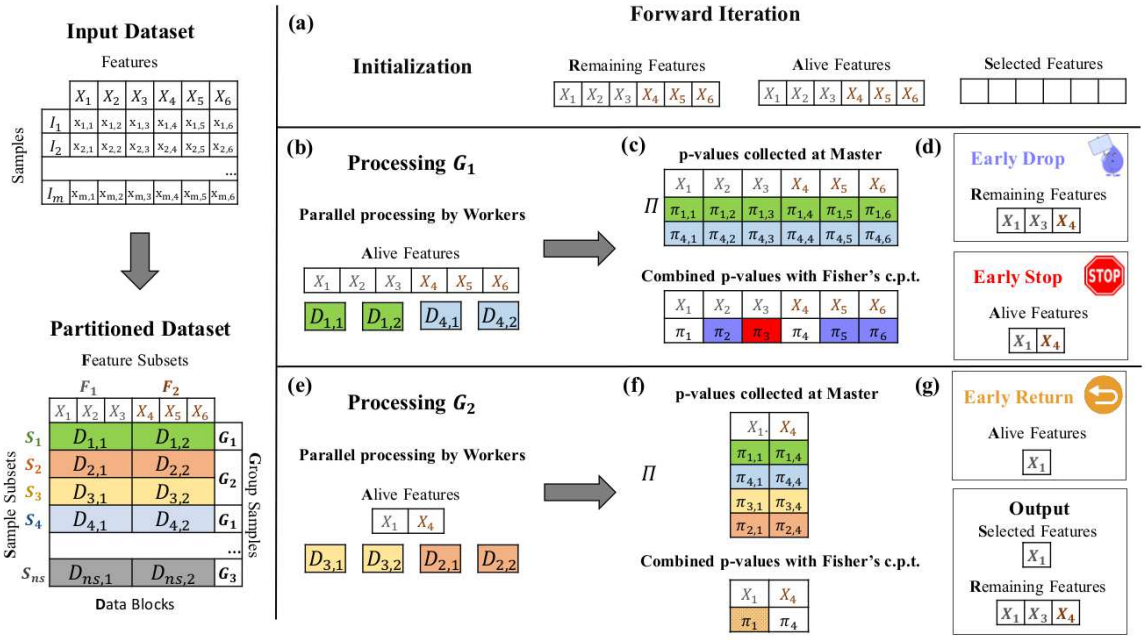


Figure 5.1: **Left**: Data partitioning of the algorithm. In the top the initial data matrix $\mathcal{D}$ is shown with 6 features and instances $I_1, \ldots, I_m$. In the bottom, the 6 features are partitioned to Feature Subsets $F_1 = \{1, 2, 3\}$ and $F_2 = \{4, 5, 6\}$. The rows are randomly partitioned to Sample Subsets $S_1, \ldots, S_{ns}$, and the Sample Subsets are assigned to Group Samples. Each Block $D_{i,j}$ is physically stored as a unit. **Right**: Example of trace of a Forward Iteration of PFBP. (a) The **R**emaining features, **A**live features, and **S**elected features are initialized. (b) All Data Blocks $D_{1,1}, D_{1,2}, D_{4,1}, D_{4,2}$ in the first Group are processed in parallel (by workers). (c) The resulting local p-values are collected (reduced) in a master node for each Alive feature and Sample Set in the first Group (as well as the likelihoods, not shown in the Figure). (d) Bootstrap-based tests determine which features to Early Drop or Stop based on Π, or whether to Early Return (based on Λ, not shown in the Figure). The sets **R** and **A** are updated accordingly. In this example, X_2, X_5 and X_6 are Dropped, X_3 is stopped, and only X_1 and X_4 remain Alive. Notice that always $\mathbf{A} \subseteq \mathbf{R}$. (e) The second Group is processed in parallel (by workers) containing Blocks $D_{3,1}, D_{3,2}, D_{2,1}, D_{2,2}$. (f) New local p-values for all features still Alive are appended to Π. If G_2 was the last Group, global p-values for the Alive features would be computed and the one with the minimum value (in this example X_1) would be selected for inclusion in **S**. (g) In case, X_1 and X_4 are deemed almost equally predictive (based on their log-likelihoods) the current best is Early Returned.

We now describe the way $\mathcal{D}$ is partitioned in sub-matrices to enable parallel computations. First, the set of available features (columns) **F** is partitioned to about equal-sized **Feature Subsets** $\{F_1, \ldots, F_{nf}\}$. Similarly, the samples (rows) are randomly

partitioned to about equal-sized **Sample Subsets** $\{S_1, \ldots, S_{ns}\}$. The row and column partitioning defines sub-matrices called **Data Blocks** $D_{i,j}$ with rows S_i and features F_j. Sample Subsets are assigned to Q **Group Samples** $\{G_q\}_1^C$ of size C each, where each group sample G_q is a set $\{S_{q_1}, \ldots, S_{q_n}\}$ (i.e., the set of Sample Subsets is partitioned). The Data Blocks $D_{i,j}$ with samples within a group sample $S_i \in G_q$ belong in the same **Group**. This second, higher level of grouping is required by the bootstrap tests explained in Section 5.2. Data Blocks in the same Group are processed in parallel in different workers (provided enough are available). However, Groups are processed sequentially, i.e., computation in all Blocks within a Group has to complete to begin computations in the Blocks of the next Group. Obviously, if workers are more than the Data Blocks, there is no need for defining Groups. The data partitioning scheme is shown in Figure 5.1:Left. Details of how the number of Sample Sets ns, the number of Feature Subsets nf, and the number C of Group Samples are determined are provided in Section 5.3.

5.1.2 Approximating Global p-values by Combining Local p-values Using Meta-Analysis

Recall that FBED^K uses p-values stemming from conditional independence tests to rank the variables and to select the best one for inclusion (forward Phase) or exclusion (backward Phase). Extending the conditional independence tests to be computed over multiple Data Blocks is not straightforward, and may be computationally inefficient. For conditional independence tests based on regression models (e.g. logistic or Cox regression), a maximum-likelihood estimation over all samples has to be performed, which typically does not have a closed-form solution and thus requires the use of an iterative procedure (e.g. Newton descent). Due to its iterative nature, it results in a high communication cost rendering it computationally inefficient, especially for feature selection purposes on Big Data where numerous models have to be fit at each Iteration.

Instead of fitting full (global) regression models, we propose to perform the conditional independence tests locally on each data block, and to combine the resulting p-values using statistical meta-analysis techniques. Specifically, the algorithm computes *local p-values denoted by* $\pi_{i,k}$ for candidate feature X_k from only the rows in S_i of a data block $D_{i,j}$, where F_j contains the feature X_k. This enables massive parallelization of the algorithm, as each data block can be processed independently and in parallel by a different worker (Figure 5.1(b)). The local p-values $\pi_{i,k}$ are then *communicated* to the master node of the cluster, and are stored in a matrix Π (Figure 5.1(c)); we will use $\pi_{i,k}$ to refer to the elements of matrix Π, corresponding to the local p-value of X_k computed on a data block containing samples in sample set S_i. Using the p-values in matrix Π, the master node combines the p-values to *global p-values* for each feature X_k

using Fisher's combined probability test [53] (Figure 5.1(c)) [1]. Finally, we note that this approach is not limited to regression-based tests, but can be used with any type of conditional independence test, and is most appropriate for tests which are hard to parallelize, or computationally expensive (e.g. kernel-based tests [164]).

Using Fisher's combined probability test to combine local p-values does not necessarily lead to the same p-value as the one computed over all samples. There are no guarantees how close those p-values will be in case the null hypothesis of conditional independence holds, except that they are uniformly distributed between 0 and 1. In case the null hypothesis does not hold however (the dependency holds), one expects to reject the null hypothesis using either method in the sample limit. What is important for PFBP is to make the same decision at each Iteration, that is, that the top ranked variable given by either p-value computation method is the same. However, even if the top ranked variable is not the same one, PFBP may still perform well, as long as some other informative variable is ranked first. In Appendix D.2 we investigate in experiments on synthetic data how both approaches compare when the task is to select the best variable at a given Iteration. We show that, if the sample size per data block is sufficiently large, combined p-values and p-values obtained from tests on all samples lead to the same choice with high probability.

For the computation of the local p-values on $D_{i,j}$, samples S_i of the selected features $\mathbf{S}$ are required, and thus the data need to be broadcast to every worker processing $D_{i,j}$ whenever $\mathbf{S}$ is augmented, i.e., in the end of each Forward Iteration. In total, the communication cost of the algorithm is due to the assembly of all local p-values $\pi_{i,k}$ to determine the next feature to include (exclude), as well as the broadcast of the data for the newly added feature in $\mathbf{S}$ at the end of each forward Iteration. *We would like to emphasize that the bulk of computation of the algorithm is the calculation of local p-values that require expensive statistical tests and it takes place in the workers in parallel. The central computations in the master are minimal.*

5.1.3 Speeding-up PFBP using Pruning Heuristics

In this section, we present 3 pruning heuristics used by PFBP to speed-up computation. Implementation details of the heuristics using locally computed p-values are presented in Section 5.2.

Early Dropping of Features from Subsequent Iterations

PFBP, as does FBED^K, uses the **Early Dropping** (ED) heuristic, presented in Section 4.1. For the sake of completeness, we briefly describe it next.

[1] Naturally, any method for combining p-values can be used instead of Fisher's method, but we did not further investigate this in this work.

Let **R** denote the set of remaining features, that is, the set of features still under consideration for selection. Initially, $\mathbf{R} = \mathbf{F} \setminus \mathbf{S}$, where **F** is the set of all available features and **S** is the set of selected features, which is initially empty. At each forward Iteration, ED removes from **R** all features that are conditionally independent of the target T given the set of currently selected features **S**. Typically, just after the first few Iterations of PFBP, only a very small proportion of the features will still remain in **R**, leading to orders of magnitude of efficiency improvements even in the non-parallel version of the algorithm. When the set of variables **R** becomes empty, we say that PFBP finished one **Run**. Unfortunately, the Early Dropping heuristic without further adjustments may miss important features which seem uninformative at first, but provide information for T when considered with features selected in subsequent Iterations. Features should be given additional opportunities to be selected by performing more Runs. Each additional Run calls the forward phase again but starts with the previously selected variables **S** and re-initializes the remaining variables to $\mathbf{R} = \mathbf{F} \setminus \mathbf{S}$. By default, PFBP uses 2 Runs, although a different number of Runs may be used. Typically a value of 1 or 2 is sufficient in practice, with larger values requiring more computational time while also giving stronger theoretical guarantees.

Early Stopping of Features within the Same Iteration

The next addition to the algorithm regards **Early Stopping** (ES) of consideration of features *within the same* Iteration, i.e., in order to select the next best feature to select in a forward Iteration or to remove in a backward Iteration. To implement ES we introduce the set **A** of features still *Alive* (i.e., under consideration) in the current Iteration, initialized to $\mathbf{A} = \mathbf{R}$ at the beginning of each Iteration (see Figure 5.1(a)). As the master node gathers local p-values for a feature X_k from several Data Blocks, it may be able to determine that no more local p-values need to be computed for X_k. This is the case if these p-values are enough to safely decide that with high probability X_k is not going to be selected for inclusion (Forward Phase) or exclusion (Backward Phase) in this Iteration (see Section 5.2 for a bootstrap-based procedure that performs this test). In this case, X_k is removed from the set of alive features **A**, and is not further considered in the current Iteration (see Figure 5.1(d)). *This allows PFBP to quickly filter out variables which will not be selected at the current Iteration. Thus, ES leads to a super-linear speed-up of the feature selection algorithm with respect to the sample size: even if the sample size is doubled, the same features will be Early Stopped; p-values will not be computed for these features on the extra samples.*

Early Return of the Winning Feature

The final heuristic of the algorithm is called **Early Return** (ER). Recall that Early Dropping will remove features conditionally independent of T given $\mathbf{S}$ from this and subsequent Iterations while Early Stopping will remove non-winners from the current Iteration. However, even using both heuristics, the algorithm will keep producing local p-values for features X_j and X_k that are candidates for selection and at the same time are informationally indistinguishable (equally predictive given $\mathbf{S}$) with regards to T (this is the case when the residuals of X_j and X_k given $\mathbf{S}$ are almost collinear). When two or more features are both candidates for selection and almost indistinguishable, it does not make sense to go through the remaining data: all choices are almost equally good. Hence, Early Return terminates the *computation* in the current Iteration and returns the current best feature X_j, if with high probability it is not going to be much worse than the best feature at the end of the Iteration (see Figure 5.1(g)). Again, the result is that computation in the current Iteration may not process all Groups. The motivation behind Early Return is similar to Early Stopping, in that it tries to quickly determine the next feature to select. The difference is that, Early Return tries to quickly determine whether a variable is "good enough" to be selected, in contrast to Early Stopping which discards unpromising variables.

A technical detail is that judging whether two features X_i and X_j are "equally predictive" is implemented using the log-likelihoods λ_i and λ_j of the models with predictors $\mathbf{S} \cup \{X_i\}$ and $\mathbf{S} \cup \{X_j\}$ instead of the corresponding p-values. The likelihoods are part of the computation of the p-values, thus incur no additional computational overhead.

5.1.4 The Parallel Forward-Backward with Pruning Algorithm

We present the proposed Parallel Forward-Backward with Pruning (PFBP) algorithm, shown in Algorithm 4. To improve readability, several arguments are omitted from function calls. PFBP takes as input a dataset $\mathcal{D}$ and the target variable of interest T. Initially the number of Sample Sets *ns* and number of Feature Sets *nf* are determined as described in Section 5.3. Then, (a) the samples are randomly assigned to Sample Sets $S_1, \ldots, S_{ns}$, to avoid any systematic biases (see also Section B.2.1), (b) the Sample Sets $S_1, \ldots, S_{ns}$ are assigned to Q approximately equal-sized Groups, $G_1, \ldots, G_Q$, (c) the features are assigned to feature sets $F_1, \ldots, F_{nf}$, in order of occurrence in the dataset, and (d) the dataset $\mathcal{D}$ is partitioned into data blocks $D_{i,j}$, with each such block containing samples and features corresponding to sample set S_i and feature set F_j respectively. The selected variables $\mathbf{S}$ are initialized to the empty set. The main loop of the algorithm performs up to *maxRuns* Runs, as long as the selected variables $\mathbf{S}$ change. Each such Run executes a forward and a backward Phase.

Algorithm 4 Parallel Forward-Backward With Pruning (PFBP)

Input: Dataset $\mathcal{D}$, Target T, Maximum Number of Runs maxRuns
Output: Selected Variables **S**

```
1: //Data Partitioning
2: Randomly assign samples to sample sets S_1, ..., S_ns
3: Assign sample sets S_1, ..., S_ns to equally-sized Groups G_1, ..., G_K
4: Assign features to feature sets F_1, ..., F_nf
5: Partition D to data blocks D_{i,j} containing samples from S_i and F_j, ∀i, j
6:
7: S ← ∅ //No selected variables
8: run ← 1 //First run
9:
10: //Iterate until (a) maximum number of runs reached, or (b) selected features S did not
    change
11: while run ≤ maxRuns ∧ S changes do
12:     S ← OneRun(D, T, S)
13:     run ← run + 1
14: end while
15: return S
16: function OneRun(Data Blocks D, Target T, Selected Variables S, Maximum Number
    of Variables To Select maxVars)
17: R ← F \ S //All variables remaining
18: //Forward phase: iterate until (a) maximum number of variables selected or (b) no new
    variable has been selected
19: while |S| < maxVars ∧ S changes do
20:     ⟨S, R⟩ ← ForwardIteration(D, T, S, R)
21: end while
22:
23: //Backward phase: iterate until no variable can be removed
24: while S changes do
25:     S ← BackwardIteration(D, T, S)
26: end while
27: return S
```

The OneRun function takes as input a set of data blocks D, the target variable T, a set of selected variables **S**, and a limit on the number of variables to select *maxVars*. It initializes the set of remaining variables **R** to all non-selected variables $\mathbf{F} \setminus \mathbf{S}$. Then, it executes the forward and backward Phases. The forward (backward) Phase executes forward (backward) Iterations until some stopping criteria are met. Specifically, the forward Phase terminates if the maximum number of variables *maxVars* has been selected, or until no more variable can be selected, while the backward Phase terminates if no more variables can be removed from **S**.

Algorithm 5 FORWARDITERATION

Input: Data Blocks D, Target T, Selected Variables $\mathbf{S}$, Remaining Variables $\mathbf{R}$
Output: Selected Variables $\mathbf{S}$, Remaining Variables $\mathbf{R}$

1: $\mathbf{A} \leftarrow \mathbf{R}$ //*Initialize Alive Variables*
2: Π //*Array of log-pvalues, initially empty*
3: Λ //*Array of log-likelihoods, initially empty*
4: $q \leftarrow 1$ //*Initialize current Group counter*
5: $Q \leftarrow$ #*Groups* //*Set Q to the total number of Groups*
6:
7: **while** $q \leq Q$ **do**
8: //*Process the alive features* $\mathbf{A}$ *for all data blocks containing sample sets in* G_q *(denoted as* D_q*) in parallel in workers for the given* T*,* $\mathbf{S}$ *and* $\mathbf{A}$*, compute sub-matrices* Π_q *and* Λ_q *from each block, and append results to* Π *and* Λ
9: $\langle \Pi_q, \Lambda_q \rangle \leftarrow$ TESTPARALLEL$(D_q, T, \mathbf{S}, \mathbf{A})$
10: $\langle \mathbf{R}, \mathbf{A} \rangle \leftarrow$ EARLYDROPPING$(\Pi, \mathbf{R}, \mathbf{A})$
11: $\mathbf{A} \leftarrow$ EARLYSTOPPING$(\Pi, \mathbf{A})$
12: $\mathbf{A} \leftarrow$ EARLYRETURN$(\Lambda, \mathbf{A})$
13: Update Π and Λ (Retain only columns of alive variables)
14: //*Stop if single variable alive*
15: **if** $|\mathbf{A}| \leq 1$ **then**
16: **break**
17: **end if**
18: $q \leftarrow q + 1$
19: **end while**
20:
21: **if** $|\mathbf{A}| > 0$ **then**
22: $\pi \leftarrow$ COMBINE(Π) //*Compute final combined* p*-value for all alive variables*
23: $X_{best} \leftarrow \underset{X_i \in \mathbf{A}}{\operatorname{argmin}}\ \pi_i$ //*Identify the best variable* X_{best}
24: //*Select* X_{best} *if dependent with* T *given* $\mathbf{S}$
25: **if** $\pi_{best} \leq \alpha$ **then**
26: $\mathbf{S} \leftarrow \mathbf{S} \cup \{X_{best}\}$
27: $\mathbf{R} \leftarrow \mathbf{R} \setminus \{X_{best}\}$
28: **end if**
29: **end if**
30: **return** $\langle \mathbf{S}, \mathbf{R} \rangle$

The forward and backward Iteration procedures are shown in Algorithms 5 and 6. FORWARDITERATION takes as input the data blocks D, the target variable T as well as the current sets of remaining and selected variables, performs a forward Iteration and outputs the updated sets of selected and remaining variables. It uses the variable set $\mathbf{A}$ to keep track of all alive variables, i.e. variables that are candidates for selection. The arrays Π and Λ contain the local log p-values and log-likelihoods, containing ns rows

Algorithm 6 BACKWARDITERATION

Input: Data Blocks D, Target T, Selected Variables $\mathbf{S}$
Output: Selected Variables $\mathbf{S}$, Remaining Variables $\mathbf{R}$

1: $\mathbf{A} \leftarrow \mathbf{S}$ //*Initialize Alive Variables*
2: Π //*Array of log-pvalues, initially empty*
3: $q \leftarrow 1$ //*Initialize current Group counter*
4: $Q \leftarrow$ #*Groups* //*Set Q to the total number of Groups*
5:
6: **while** $q \leq Q$ **do**
7: //*Process the alive features* $\mathbf{A}$ *for all data blocks containing sample sets in* G_q *(denoted as* D_q*) in parallel in workers for the given* T*,* $\mathbf{S}$ *and* $\mathbf{A}$*, compute sub-matrix* Π_q *from each block, and append it to* Π
8: $\Pi_q \leftarrow$ TESTPARALLEL$(D_q, T, \mathbf{S}, \mathbf{A})$
9: $\mathbf{A} \leftarrow$ EARLYSTOPPINGBACKWARD$(\Pi, \mathbf{A})$
10: Update Π (Retain only columns of alive variables)
11: //*Stop if single variable alive*
12: **if** $|\mathbf{A}| \leq 1$ **then**
13: **break**
14: **end if**
15: $q \leftarrow q + 1$
16: **end while**
17:
18: **if** $|\mathbf{A}| > 0$ **then**
19: $\pi \leftarrow$ COMBINE(Π_q) //*Compute final combined* p*-value for all alive variables*
20: $X_{worst} \leftarrow \underset{X_i \in \mathbf{A}}{\operatorname{argmax}}\, \pi_i$ //*Identify the worst variable* X_{worst}
21: //*Remove* X_{worst} *if independent with* T *given* $\mathbf{S} \setminus X_{worst}$
22: **if** $\pi_{worst} > \alpha$ **then**
23: $\mathbf{S} \leftarrow \mathbf{S} \setminus \{X_{worst}\}$
24: **end if**
25: **end if**
26: **return S**

(one for each sample set) and $|\mathbf{A}|$ columns (one for each alive variable). The values of Π and Λ are initially empty, and are filled gradually after preprocessing each Group. We use D_q to denote all data blocks which corresponds to Sample Sets contained in Group G_q. Similarly, accessing the values of Π and Λ corresponding to Group q and variables $\mathbf{X}$ is denoted as Π_q and Λ_q.

In the main loop, the algorithm iteratively processes Groups in a synchronous fashion, until all Groups have been processed or no alive variable remains. The TESTPARALLEL function takes as input the data blocks D_q corresponding to the current Group G_q, and performs all necessary independence tests in parallel in workers. The

results, denoted as Π_q and Λ_q are then appended to the Π and Λ matrices respectively. After processing a Group, the tests for Early Dropping, Early Stopping and Early Return are performed, using all local p-values computed up to Group q; details about the implementation of the EarlyDropping, EarlyStopping and EarlyReturn algorithms when data have only been partially processed are given in Section 5.2. The values of non-alive features are then removed from Π and Λ (see also Figure 5.1(f) for an example). If only a single alive variable remains, processing stops. Note that, this is not checked in the while loop condition, in order to ensure that at least one Group has been processed if the input set of remaining variables contains a single variable. Finally, the best alive variable (if such a variable exists) is selected if it is conditionally dependent with T given the selected variables $\mathbf{S}$. Conditional dependence is determined by using the p-value resulting from combining all local p-values available in Π. BackwardIteration is similar to ForwardIteration with the exception that (a) the remaining variables are not needed, and thus no dropping is performed, (b) no early return is performed, and (c) the tests are reversed, i.e. the worst variable is removed.

Implementation details regarding the accurate computation of the logarithm of p-values, the logistic-regression based likelihood-ratio test, as well as a more efficient method for the univariate tests based on score tests are given in Appendix B.2.

5.1.5 Massively-Parallel Predictive Modeling

The technique of combining locally computed p-values to global ones to massively parallelize computations, can be applied not only for feature selection, but also for predictive modeling. We exploit this opportunity in the context of independence tests implemented by logistic regression. During the computation of local p-values $\pi_{i,k}$ a (logistic) model for T using all selected features $\mathbf{S}$ is produced from the samples in S_i. Such a model computes a set of coefficients β_i that weighs each feature in the model to produce the probability that $T = 1$. We used the weighted univariate least squares (WLS) approach [72], with equal weights for each model; equal weights were used as the sample size of each partition is (approximately) the same. The WLS method with equal weights combines the N local models to a global one $\hat{\beta}$ by just taking the average of the coefficient vectors of the model , i.e., $\hat{\beta} = \frac{1}{N}\sum_{i=1}^{N}\beta_i$. Thus, the only change to the algorithm is to cache each β_i and average them in the master node. This way, at the end of the feature selection process one could obtain an approximate predictive model with no additional overhead! By default, PFBP uses the WLS method to construct a predictive model at each forward Iteration. Other multivariate methods for combining multiple models, which also consider the co-variance of the estimated coefficients are described in [9]. Such methods could also be applied in our case

without any significant computational overhead, but were not further considered in this work.

Using the previous technique, one could obtain a model at the end of each Iteration without extra computations and assess its predictive performance (e.g., accuracy) on a hold-out validation set. Constructing for instance the graph of the number of selected features versus achieved predictive performance on the hold-out set could visually assist data analysts [87] in determining how many features to include in the final selections; an example application on SNP data is given in the experimental section in Figure 5.4. An automated criterion for selecting the best trade-off between the number of selected features and the achieved predictive performance could also be devised, although this is out of the scope of this work, as multiple testing has to be taken into consideration.

5.2 Implementation of the Early Dropping, Stopping and Return Heuristics using Bootstrap Tests on Local p-values

Recall that the algorithm processes Group Samples sequentially. After processing each Group and collecting the results, PFBP applies the Early Dropping, Early Stopping and Early Return heuristics, computed on the master node, to filter out variables and reduce subsequent computation. Thus, all three heuristics involve making early probabilistic decisions based on a subset of the samples examined so far. Naturally, if all samples have been processed, Early Dropping can be applied on the combined p-values without making probabilistic decisions.

Before proceeding with the details, we provide the notation used hereafter. Let Π and Λ be 2-dimensional arrays containing K local log p-values and log-likelihoods for all alive variables in $\mathbf{A}$ and for all Groups already processed. The matrices reside on the master node, and are updated each time a Group is processed. Let $\pi_{i,j}$ and $\lambda_{i,j}$ denote the i-th value of the j-th alive variable, denoted as X_j. Recall that those values have been computed locally on the Data Block containing samples from Sample Set S_i. For the sake of simplicity, we will use π_j and λ_j (l_j) to denote the combined p-value and sum of log-likelihoods (likelihood) respectively of variable X_j. The vectors π and λ will be used to refer to the combined p-values and sum of log-likelihoods for all alive variables respectively. Also, let X_{best} be the variable that would have been selected if no more data blocks were evaluated, that is, the one with the currently lowest combined p-value, denoted as π_{best}.

5.2.1 Bootstrap Tests for Early Probabilistic Decisions

In order to make early probabilistic decisions, we test: (a) $P(\pi_j \geq a) > P_{drop}$ for Early Dropping of X_j (i.e., the probability of the j-th feature deemed independent at significance level a at the end of the Iteration is larger than a threshold), (b) $P(\pi_{best} < \pi_j) > P_{stop}$ for Early Stopping of X_j (i.e., the probability of the current best feature having a smaller "better" p-value than feature j is larger than a threshold), and (c) $\forall X_j, (P(l_{best}/l_j \geq t) > P_{return})$ for Early Return of X_{best} (the likelihood ratio l_{best}/l_j indicate how close is the model with the currently best feature and the mode with feature l_j; if all ratios with all alive features are above a certain threshold with high probability, then the current best choice is close to optimal). The t is a tolerance parameter that determines how close the compared models should be. It takes values between 0 and 1; the closer it is to 1, the closer it is guaranteed that the current best model will be to all other ones in consideration in terms of likelihood. By taking the logarithm, (c) can be rewritten as $\forall X_j, P(\lambda_{best} - \lambda_j \geq lt)$, where $lt = \log(t)$.

We employed bootstrapping to test the above. A bootstrap-sample b of Π (Λ), denoted as Π^b (Λ^b), is created by sampling with replacement K **rows** from Π (Λ). Then, for each such sample, the Fisher's combined p-values (sum of log-likelihoods) are computed, by summing over all respective values for each alive variable; we refer to the vector of combined p-values (log-likelihoods) on bootstrap sample b as π^b (λ^b), and the i-th element is referred to as π_i^b (λ_i^b). By performing the above B times, probabilities (a), (b) and (c) can be estimated as:

$$P(\pi_j \geq a) = \frac{\mathrm{I}(\pi_j \geq a) + \sum_{b=1}^{B} \mathrm{I}(\pi_j^b \geq a)}{B+1} \qquad \text{(Early Dropping)}$$

$$P(\pi_j > \pi_{best}) = \frac{\mathrm{I}(\pi_j > \pi_{best}) + \sum_{b=1}^{B} \mathrm{I}(\pi_j^b > \pi_{best}^b)}{B+1} \qquad \text{(Early Stopping)}$$

$$P(\lambda_{best} - \lambda_j \geq lt) = \frac{\mathrm{I}(\lambda_{best} - \lambda_j \geq lt) + \sum_{b=1}^{B} \mathrm{I}(\lambda_{best}^b - \lambda_j^b \geq lt)}{B+1} \qquad \text{(Early Return)}$$

where I is the indicator function, which evaluates to 1 if the inequality holds and to 0 otherwise. For all of the above, the condition is also computed on the original sample, and the result is divided by the number of bootstrap iterations B plus 1. Note that, for Early Return the above value is computed for all features X_j.

Algorithms 7,8 and 9 show the procedures in more detail. For all heuristics, a vector named *cnts* is used to keep track of how often the inequality is satisfied for each variable. To avoid cluttering, the indicator function I performs the check for multiple variables and returns a vector of values in each case, containing one

Algorithm 7 EARLYDROPPING

Input: Log p-values Π, Remaining Variables $\mathbf{R}$, Alive Variables $\mathbf{A}$, Number of Bootstrap Samples B, Significance Level Threshold α, ED Threshold P_{drop}
Output: Remaining variables $\mathbf{R}$, Alive Variables $\mathbf{A}$

1: $\pi \leftarrow$ COMBINE(Π) //*Combine log p-values Π using Fisher's c.p.t.*
2: cnts $\leftarrow 0^{|\mathbf{A}|}$ //*Count vector of size equal to the number of alive variables*
3: cnts $\leftarrow$ cnts + I($\pi \geq \alpha$)
4: **for** b = 1 to B **do**
5: $\quad \Pi^b \leftarrow$ BOOTSTRAPSAMPLE(Π)
6: $\quad \pi^b \leftarrow$ COMBINE(Π^b) //*Combine log p-values Π^b using Fisher's c.p.t.*
7: $\quad$ cnts $\leftarrow$ cnts + I($\pi^b \geq \alpha$)
8: **end for**
9: //*Drop variables if p-value larger than α with probability at least P_{drop}*
10: $\mathbf{R} \leftarrow \mathbf{R} \setminus \{X_i \in \mathbf{A} : cnts_i/(B+1) \geq P_{drop}\}$
11: $\mathbf{A} \leftarrow \mathbf{A} \setminus \{X_i \in \mathbf{A} : cnts_i/(B+1) \geq P_{drop}\}$
12: **return** $\langle \mathbf{R}, \mathbf{A} \rangle$

Algorithm 8 EARLYSTOPPING

Input: Log p-values Π, Alive Variables $\mathbf{A}$, Number of Bootstrap Samples B, ES Threshold P_{stop}
Output: Alive Variables $\mathbf{A}$

1: $\pi \leftarrow$ COMBINE(Π) //*Combine log p-values Π using Fisher's c.p.t.*
2: $X_{best} \leftarrow \underset{X_i \in \mathbf{A}}{\operatorname{argmin}}\ \pi_i$ //*Identify variable with minimum Fisher's combined p-value*
3: cnts $\leftarrow 0^{|\mathbf{A}|}$ //*Count vector of size equal to the number of alive variables*
4: cnts $\leftarrow$ cnts + I($\pi_{best} < \pi$)
5: **for** b = 1 to B **do**
6: $\quad \Pi^b \leftarrow$ BOOTSTRAPSAMPLE(Π)
7: $\quad \pi^b \leftarrow$ COMBINE(Π^b) //*Combine log p-values Π^b using Fisher's c.p.t.*
8: $\quad$ cnts $\leftarrow$ cnts + I($\pi^b_{best} < \pi^b$)
9: **end for**
10: //*Exclude variables from $\mathbf{A}$ that are worse than V_{best} with probability at least P_{stop}*
11: $\mathbf{A} \leftarrow \mathbf{A} \setminus \{X_i \in \mathbf{A} : cnts_i/(B+1) \geq P_{stop}\}$
12: **return** $\mathbf{A}$

value for each variable. The function BOOTSTRAPSAMPLE creates a bootstrap sample as described above, function COMBINE uses Fisher's combined probability test to compute a combined p-value, and SUMROWS sums over all rows of the log-likelihoods contained in Λ, returning a single value for each alive variable.

Algorithm 9 EARLYRETURN

Input: Log-likelihoods Λ, Alive Variables $\mathbf{A}$, Number of Bootstrap Samples B, ER Threshold P_{return}, ER Tolerance lt
Output: Alive Variables $\mathbf{A}$

1: $\pi \leftarrow$ COMBINE(Π) //*Combine log p-values Π using Fisher's c.p.t.*
2: $\lambda \leftarrow$ SUMROWS(Λ) //*Sum rows of log-likelihoods Λ*
3: $X_{best} \leftarrow \underset{X_i \in \mathbf{A}}{\text{argmin}}\ \pi_i$ //*Identify variable with minimum Fisher's combined p-value*
4: cnts $\leftarrow 0^{|\mathbf{A}|}$ //*Count vector of size equal to the number of alive variables*
5: cnt $\leftarrow$ cnts + I($\lambda_{best} - \lambda > lt$)
6: **for** b = 1 to B **do**
7: $\quad \Lambda^b \leftarrow$ BOOTSTRAPSAMPLE(Λ)
8: $\quad \lambda^b \leftarrow$ SUMROWS(Λ) //*Sum rows of log-likelihoods Λ^b*
9: $\quad$ cnts $\leftarrow$ cnts + I($\lambda^b_{best} - \lambda^b > lt$)
10: **end for**
11: //*Select X_{best} early if better than all other variables with probability at least P_{return}*
12: **if** $\forall i, cnts_i/(B+1) \geq P_{return}$ **then**
13: $\quad \mathbf{A} \leftarrow \{X_{best}\}$
14: **end if**
15: **return A**

5.2.2 Implementation Details of Bootstrap Testing

We recommend using the same sequence of bootstrap indices for each variable, and for each bootstrap test. The main reasons are to (a) simplify implementation, (b) avoid mistakes and (c) ensure results do not change across different executions of the algorithms. This can be done by initializing the random number generator with the same seed. Next, note that ED, ES and ER do not necessarily have to be performed separately, but can be performed simultaneously (i.e,. using the same bootstrap samplings). This allows the re-usage of the sampled indices for all tests and variables, saving some computational time. Another important observation for ED and ES is that the actual combined p-values are not required. It suffices to compare statistics instead, which are inversely related to p-values: larger statistics correspond to lower p-values. For the ED test, the statistic has to be compared to the statistic corresponding to the significance level α, which can be computed using the inverse χ^2 cumulative distribution. This is crucial to speed-up the procedure, as computing log p-values is computationally expensive. Finally, note that it is not always necessary to perform all bootstrap iterations to decide if the probability is below the threshold. This can be done by keeping track of an upper bound of the estimated probabilities, and to stop the bootstrap procedure if that bound is below the threshold, further reducing the computational cost. For example, let $P_{drop} = 0.99$ and $B = 999$. Then, in order to drop

a variable X_i, the number of times $cnts_i$ where the p-value of X_i exceeds a has to be at least 990. If after K iterations $(B-K)+cnts_i$ is less than 990, one can determine that X_i will not be dropped; even if in all remaining bootstrap iterations its p-value is larger than a, $cnts_i + B - K$ will always be less than 990, and thus the probability $P(\pi_i \geq a)$ will be less than the threshold $P_{drop} = 0.99$.

Finally, we note that, in order to minimize the probability of wrong decisions, large values for the ED, ES and ER thresholds should be used. We found that values of 0.99 for P_{drop} and P_{stop}, and values of $P_{return} = 0.95$ and $tol = 0.9$ work well in practice. Furthermore, the number of bootstraps B should be as large as possible, with a minimum recommended value of 500. By default, PFBP uses the above values.

5.3 Tuning the Data Partitioning Parameters of the Algorithm

The main parameters for the data partitioning to determine are (a) the sample size s of each Data Block, (b) the number of features f in each Data Block, and (c) the number of Sample Subsets C in each Group; the latter determines how many new p-values per feature are computed in each Group. Notice that *s determines the horizontal partitioning of the data matrix and f the vertical partitioning of data matrix*. In this section, we provide detailed guidelines to determine those parameters, and show how those values were set for the special case of PFBP using conditional independence tests based on binary logistic regression. *Selecting the data partitioning parameters needs to consider both statistical phenomena, as well as the hardware architecture*. A trade-off exists between accuracy of statistical estimation of p-values and the bootstrap tests and the induced parallelism.

5.3.1 Determining the Required Sample Size s for Conditional Independence Tests

For optimal computational performance, the number of Sample Sets should be as large as possible to increase parallelism, and each Sample Set should contain as few samples as possible to reduce the computational cost for performing the local conditional independence tests. On the other hand, there should be enough samples per Sample Set so that the local tests have enough statistical power.

Various rules of thumb have appeared in the literature to choose a sufficient number of samples for linear, logistic and Cox regression [67,116,153]. We focus on the case of binary logistic regression hereafter. For binary logistic regression, it is recommended to use at least $s = c/\min(p_0, p_1) \cdot df$ samples, where p_0 and p_1 are the proportion of negative and positive classes in T respectively, df is the number of degrees of freedom in the model (that is, the total number of parameters, including the intercept term) and c is usually recommended to be between 5 and 20, with larger values leading to

more accurate results. This rule is based on the events per variable (EPV) [116], and will referred to as the EPV rule hereafter.

Rules like the above can be used to determine the number of samples s in each Sample Set, by setting the minimum number of samples in each Data Block in a way that the locally computed p-values are valid for the type of test employed *in the worst case*. The worst case scenario occurs if the maximum number of features *maxVars* have been selected. If all features are continuous, then the maximum number of parameters of a model is $df = maxVars + 1$. This can easily be adapted for the case of categorical features, by considering the *maxVars* variables with the most categories, and setting df appropriately. By considering the worst case scenario, the required number of samples can be computed by plugging the values of df, c, p_0 and p_1 into the EPV rule. We found out that, although the EPV rule works reasonably well, it tends to overestimate the number of samples required for skewed class distributions. As a result, it may unnecessarily slow down PFBP in such cases. Ideally for a given value of c the results should be equally accurate irrespective of the class distribution and the number of model parameters.

To overcome the drawbacks of the EPV rule, we propose another rule, called the STD rule, which is computed as $s = df \cdot c / \sqrt{p_0 \cdot p_1}$. For balanced class distributions the result is identical to the EPV rule, while for skewed distributions the value is always smaller. We found that a value of $c = 10$ works sufficiently well, and recommend to always set c to a minimum of 10; higher values could lead to more accurate results, but will also increase computation time. Again, the number of samples per Sample Set is determined as described above. A comparison of both rules is given in Appendix D.2. We show that the STD rule behaves better across different values of df and class distributions of the outcome than the EPV rule.

5.3.2 Setting the Number of Sample Sets C per Group

We now discuss the determination of the C value, the number of Sample Sets in each Group. The value of C determines how many Sample Sets are processed in parallel and thus, how many additional local p-values for each feature are added to matrix Π at the end of processing each Group. In other words, before invoking the next round of bootstrap tests that decide on Early Dropping, Stopping or Return, C additional p-values will be available to these tests. We recommend a minimal value for C to be at least 15, otherwise the first round of bootstrap tests becomes unreliable. The value of C determines how often the workers stop and await the master to perform the bootstrap tests, which should not be too often. In our experiments we have set $C = 30$, without extensive tuning. In addition, whenever there is no progress made by any of the heuristics (when the "easy-to-determine" features have already been stopped or

dropped), the value of C is doubled dynamically for that Iteration. This trick avoids stopping too often without any progress made. C is then reset in the next iteration.

5.3.3 Determining the Number of Features per Data Block

At this point, we assume we have chosen the sample size s of each data block $D_{i,j}$. We also assume we have decided upon the value of C, i.e., the number of Sample Sets in each Group. In other words, we have selected the horizontal partitioning of the data at two levels: first, the partitioning of samples to Sample Sets and then to samples that belong to the same Group. Next, we need to decide the vertical partitioning to nf equal-size Feature Sets. The number of blocks per group will then become $C \times nf$. In a system with M available workers that can process the blocks in parallel, it makes sense to determine nf so that $M \approx C \times nf$. Specifically, we set $nf = \lfloor M/C \rfloor$. In the extreme case where a data block does not fit in the main memory of a machine, nf has be to increased and the data to be physically partitioned to different machines.

5.4 Related Work

In this section we provide an overview of related parallel feature selection methods, focusing on methods for MapReduce-based systems (such as Spark), and compare them to PFBP. An overview of common classes of feature selection algorithms can be found in Chapter 3, as well as in [63] and [94].

5.4.1 Parallel Univariate Feature Selection and Parallel Forward-Backward Selection

UFS and FBS rank features according to p-values of independence tests. UFS computes the p-values of the unconditional test between a feature and the target, while FBS performs conditional tests given the already selected features. UFS statically ranks features, while FBS updates the ranking with every newly selected feature. The algorithms stop when the maximum number of features has been selected, or the p-values are below some significance threshold. *Both UFS and FBS can be parallelized at the level of the underlying statistical test employed*. Specifically, the Spark machine learning library MLlib [104] offers parallel implementations of Pearson and Spearman correlation coefficients for continuous data, and the chi-square test of independence for discrete data. For conditional independence, tests can be constructed using the likelihood ratio technique by fitting statistical models. MLlib offers parallelized binomial, multinomial, and linear regression models to this end. We employed these parallelized statistical tests to implement UFS and FBS in the experimental section.

The main advantages of PFBP over UFS and FBS are that (a) PFBP does not

require specialized distributed implementations of independence tests, as it only relies on local computations and thus can use existing implementations. Local fitting and combining is also much faster than fitting full models over all samples, and (b) PFBP employs the Early Dropping, Early Stopping and Early Return heuristics, allowing it to scale both with number of features and samples. Perhaps, most importantly (c) UFS will not necessarily identify the Markov Blanket of T even in faithful distributions; the solution by UFS will have false positives (e.g., redundant features) as well as false negatives (missed Markov Blanket features).

5.4.2 Single Feature Optimization

The Single Feature Optimization algorithm (SFO) [135] is a Map-Reduce-based extension of the standard forward selection algorithm using binary logistic regression. In essence, SFO employs an efficiency trick to approximate the parallel computation of the criterion to select the next best feature, when the binary logistic regression test is employed. Thus, the algorithm is specific to classification tasks and cannot be tivially generalized to other types of classifiers in place of the logistic regression.

In more detail, SFO (a) employs a heuristic that ranks the features at each step without the need to fit a full logistic regression model (that is, one over all samples) for all variables, and (b) uses a parallelization scheme to perform parallel computation over samples and features.

We proceed by describing the ranking heuristic used by SFO. Let $\mathbf{S}$ be the selected features up to some point and $\mathbf{R} = \mathbf{F} \setminus \mathbf{S}$ be all candidate variables for selection, and assume that a full logistic regression model M for T using $\mathbf{S}$ is available. SFO creates an approximate model for each variable $R_i \in \mathbf{R}$ by fixing the coefficients of $\mathbf{S}$ using their coefficients in M, and only optimizing the coefficient of R_i. This problem is much simpler than fitting full models for each remaining variable, significantly reducing running time. Then, the best variable R^* is chosen based on those approximate models (using some performance measure such as the log-likelihood), and a full logistic regression model M^* with $\mathbf{S} \cup R^*$ is created. Thus, at each iteration only a single, full logistic regression model needs to be created. By default, SFO uses a maximum number of variables to select as a stopping criterion. Alternatively, to decide whether R^* should be selected a likelihood-ratio test could be used, in which case the test is performed on M and M^*, and R^* is selected if the p-value is below a threshold α; we used this in our implementation of SFO in the experiments. The parallelization over samples is performed in the map phase, in which one value p_j is computed for each sample j, which equals

$$p_j = \frac{e^{\beta \cdot \mathbf{S}_j}}{1 + e^{\beta \cdot \mathbf{S}_j}}$$

where β are the coefficients of $\mathbf{S}$ in M and $\mathbf{S}_j$ are the values of $\mathbf{S}$ in the j-th sample. The values of p_j, the values of the outcome T and all of candidate variables $\mathbf{R}$ are then sent to workers to be processed in the reduce phase. Note that, *this incurs a high communication cost, as essentially the whole dataset has to be exchanged among workers over the network*. Finally, in the reduce phase, all workers fit in parallel over all variables $\mathbf{R}$ the approximate logistic regression models.

Although SFO significantly improves over the standard forward selection algorithm in terms of running time, it has three main drawbacks compared to PFBP: (a) it has a high communication cost, in contrast to PFBP which only requires minimal information to be communicated, (b) to select a variable all non-selected variables have to be considered, while PFBP employs the Early Dropping heuristic that significantly reduces the number of remaining variables, and (c) SFO always uses all samples, while PFBP uses Early Stopping and Early Return allowing it to scale sub-linearly with number of samples. Finally, (d) SFO does not provide any theoretical guarantees of correctness.

5.4.3 Information Theoretic Feature Selection for Big Data

Information theoretic feature selection (ITFS) [22] methods have been extended to Big Data settings [121] and implemented for Spark[2]. They are applicable only for discrete features and outcomes, and use mutual information and conditional mutual information estimates to rank and select variables; a more detailed description and their relation with forward selection can be found in Chapter 3 and in [22].

The main advantage of the specific implementation of ITFS over PFBP's is that estimating the (conditional) mutual informations for discrete data under the multinomial assumption, does not require fitting any model; it only requires counting and constructing the contingency tables of the joint. This can be done in one pass of the data and can thus trivially exploit the sparsity of the data. In Big Data settings it can be easily parallelized. However, ITFS methods do not have the same theoretical properties of PFBP, which can be shown to be optimal for distributions that can be faithfully represented by Bayesian networks and maximal ancestral graphs. This stems from the fact that PFBP solves an inherently harder problem, as it conditions on all selected features (creates a model with all selected features) at each iteration in order to select the next feature, while ITFS only conditions on one feature at a time. Furthermore, ITFS methods are not as general as PFBP, which can be applied to various data types as long as an appropriate conditional independence test is available. Appropriate statistical tests for different cases (e.g., survival analysis, time-course data) are available and put in use within statistical-based methods similar to PFBP [89]. Last

[2]https://spark-packages.org/package/sramirez/spark-infotheoretic-feature-selection

but not least, as currently implemented, ITFS variants are only applicable to discrete data. Thus, continuous variables have to be discretized before feature selection, which may lead to information loss [43, 82], increases computational time and may require tuning to find a good discretization.

5.4.4 Parallel Lasso

The Lasso has been parallelized for single machines and shared-memory clusters [19, 95,167]. These approaches only parallelize over features and not samples (i.e. consider vertical partitioning). Naturally, ideas and techniques presented in those works could be adapted or extended for Spark or related systems. An implementation of Lasso linear regression is provided in the Spark MLlib library [104]. A disadvantage of that implementation is that it requires extensive tuning of its hyper-parameters (like the regularization parameter λ and several parameters of the optimization procedure), rendering it impractical as typically many different hyper-parameter combinations have to be tried to obtain the optimal settings. Unfortunately, we were not able to find any Spark implementation of Lasso for logistic regression, or any work dealing with the problem of efficient parallelization of Lasso on Spark.

Comparing Lasso with algorithms related to PFBP, we note that in extensive simulations it has been shown that causally-inspired feature selection methods are competitive in terms of predictive performance with Lasso on classification and survival analysis tasks on many real datasets [4, 89–91]. Furthermore, as shown in Section 4.4, FBED^K performs as well as Lasso when restricted to select the same number of variables. Finally, we note that in contrast to forward selection using conditional independence tests, Lasso is not easily extensible for different tasks, and requires specialized algorithms for different data types [77, 102, 131], whose objective function may be non-convex [131] or computationally demanding [47]. For example, for time-course data, the Lasso problem is not convex and does not scale up, while causal-based FS methods do [143].

5.4.5 Other Approaches

In addition to forward-selection based, information theoretic and Lasso based methods, there also exist other parallel feature selection methods which can't directly be categorized into one of the above but are worth mentioning. However, none of those methods has actually been applied to Big Data, which may contain millions of samples and/or features. Furthermore, all of the methods perform virtual partitioning of the data, and thus only parallelize over features and not over samples, in contrast to PFPB which parallelizes over both.

Bolón-Canedo et al. [15] introduced a method which uses vertical partitioning to distribute computations across workers. The method focuses mainly on DNA mi-

croarray data, whose number of features is much larger than the number of samples, although it is not limited to those cases. It can be applied using any feature selection method that ranks features, like the ITFS methods described previously, or even forward selection type algorithms using as a ranking the order of selection of variables. The main idea is to rank variables on each worker independently and to combine the partial rankings into a complete one. In order to be able to combine the rankings, each variable may be distributed to multiple workers in order to have some kind of overlap between the partial rankings, allowing them to be merged into one. Although the method is quite general, as it can be used in combination with any method that produces a ranking of the features, its theoretical properties are not explored in the paper, making it hard to compare theoretically against other methods.

Zhou et al. [168] introduced a general, parallel feature selection method for classification tasks that is based on group testing theory. The idea is to select a collection of tests (i.e., a subset of the input variables), with each variable being present with some probability p. This implicitly creates multiple datasets, one for each test. Those datasets can be processed independently in parallel, producing a score using some scoring function for each dataset corresponding to how predictive the respective variables are of the outcome of interest. Then, variables are ranked based on the scores attained on the datasets they participated in. The authors show that, for specific values of p, if the number of tests is large enough and if the scoring function satisfies some properties (i.e., it is what the authors call C-separable), the proposed algorithm is able to select the best features with high probability. However, it is not clear how those results are related to the theoretical properties of other feature selection algorithm (such as PFBP or Lasso), nor how they can be used in practice, especially given that the authors do not propose any way to tune the hyper-parameters of their method.

Wang et al. [155] proposed a method that also uses vertical partitioning for parallelization. It is method that can be used with any penalized regression method, both for feature selection and for model fitting. Given a dataset, it is partitioned vertically into multiple datasets, and a decorrelation step is performed on each new dataset that tries to minimize the dependencies between all datasets. Then, each dataset can be analyzed independently across multiple workers, and the results are combined into a final one on the master machine. It is shown that the convergence rate of the proposed method is nearly minimax optimal under weakly sparse assumptions on the model parameters. Furthermore, it is shown that the algorithms retains those properties irrespective of the number of partitions. In experiments, the method is shown to perform similarly to lasso while exhibiting lower running time. In contrast to PFBP, the method is specialized only to specific cases (penalized regression methods with continuous predictors), and thus is not easily extensible to other cases.

5.5 Experimental Evaluation

We performed three sets of experiments to evaluate PFBP.

1. We investigate the scalability of PFBP in terms of variable size, sample size and number of workers on synthetic datasets, simulated from Bayesian networks.
2. We compare PFBP to three competing forward-selection based feature selection algorithms.
3. We compare against three algorithms from the family of information theoretic feature selection methods [22], implemented for Big Data [121].
4. We performed a proof-of-concept experiment of PFBP on dense synthetic Single Nucleotide Polymorphism (SNP) data. These are important types of very high-dimensional data that arise in biology and for which feature selection algorithms that can scale up are needed.

We made every reasonable effort to include all candidate competitors. These alternatives constitute algorithms specifically designed for MapReduce architectures (i.e., SFO), standard FS algorithms using parallel implementations of the conditional independence tests (i.e., UFS and FBS) and ITFS. The only Lasso implementation for Spark available in the Spark MLlib library [104] (a) is for continuous targets, and thus is not suitable for binary classification tasks, and (b) required tuning of 5 hyper-parameters; as no procedure has been suggested by the authors for their tuning, it was excluded from the comparative evaluation. Additionally, for the comparative evaluation, final predictive models were build using the selected features by each algorithm using:

1. The logistic regression implementation in the Spark machine learning library MLlib, hereafter denoted as *SparkLR*
2. The logistic regression model stemming from combining local logistic models using our own implementation (see Section 5.1.5), hereafter denoted as *CombLR*

The reason for including both types of modeling is because we noticed that SparkLR often fails to converge and produces models that are worse than the trivial classifier classifying to the most common class. This last comparison provides evidence that not only local p-values can be combined using meta-analysis techniques, but also model coefficients and other estimated quantities.

5.5.1 Experimental Setup

For the scalability experiments of PFBP (Section 5.5.2) and the proof-of-concept application on SNP data (Section 5.5.4) we used a cluster with 5 machines, 1 acting as a

master and 4 as workers, connected to a 1 Gigabit network, with each machine having 2 Intel Xeon E5-2630 v3 CPUs with 8 physical cores each and 256 GB of RAM (a total of 64 cores and 1 TB of RAM). For the comparative evaluation of PFBP with other feature selection algorithms we used a cluster with 5 workers, connected to a 14 Gigabit network, with each machine having 4 Intel Xeon E5-4650 v2 CPUs with a total of 80 cores and 400 GB of RAM (a total of 320 cores and 1.6 TB of RAM). In all cases, 2 cores per worker were left out for other tasks (e.g. the operating system). The first cluster is running Spark 2.1.0 while the second is running Spark 2.0.2, and both are using the HDFS file system. All algorithms were implemented in Java 1.7 and Scala 2.11.

The significance level α was set to 0.01 for all algorithms [3], and PFBP was executed with 2 Runs. For the bootstrap tests used by PFBP, we used the default parameter values as described in Section 5.2.2. For each feature selection method we produced two predictive models: (a) using the approach described in Section 5.1.5 that combines multiple locally learned models into a single one, and (b) using the logistic regression implementation in the Spark machine learning library MLlib [104]. Parameter values related to the number of Group Samples, Sample Sets and Feature Sets were determined using the STD rule, and by setting the maximum number of variables to select to *maxVars* (the exact value is given for each specific experiment later); see Section 5.3 for more details. We note that, none of the experiments required a physical partitioning to Feature Sets, and thus each Block contains all features (i.e., the number of Features Sets nf is 1).

5.5.2 Scalability of PFBP with Sample Size, Feature Size and Number of Workers

We investigated the scalability of PFBP on dense synthetic datasets in terms of sample size, variable size and number of workers used. The data were sampled from randomly generated Bayesian networks, which are probabilistic models that can encode complicated dependency structures among features. Such *simulated data contain not only features necessary for optimal prediction (strongly relevant in the terminology of [78]) and irrelevant, but also redundant features (weakly relevant [78])*. This is a novelty in the Big Data FS literature as far as we know, making the synthetic tasks more realistic. A detailed description of the data and network generating procedures is given in

[3] Typical choices for the significance level α are 0.01 and 0.05 [4]. Using lower values of α leads to fewer type I errors (falsely selected variables) but more type II errors (falsely rejected variables). This also depends on the sample size, with more samples typically leading to fewer type II errors, with type I errors not affected by sample size. Therefore, in large sample settings, as the ones considered in our experiments, one should use lower values for α to minimize type I errors. For that reason, we chose $\alpha = 0.01$, although it would make sense to consider even lower values.

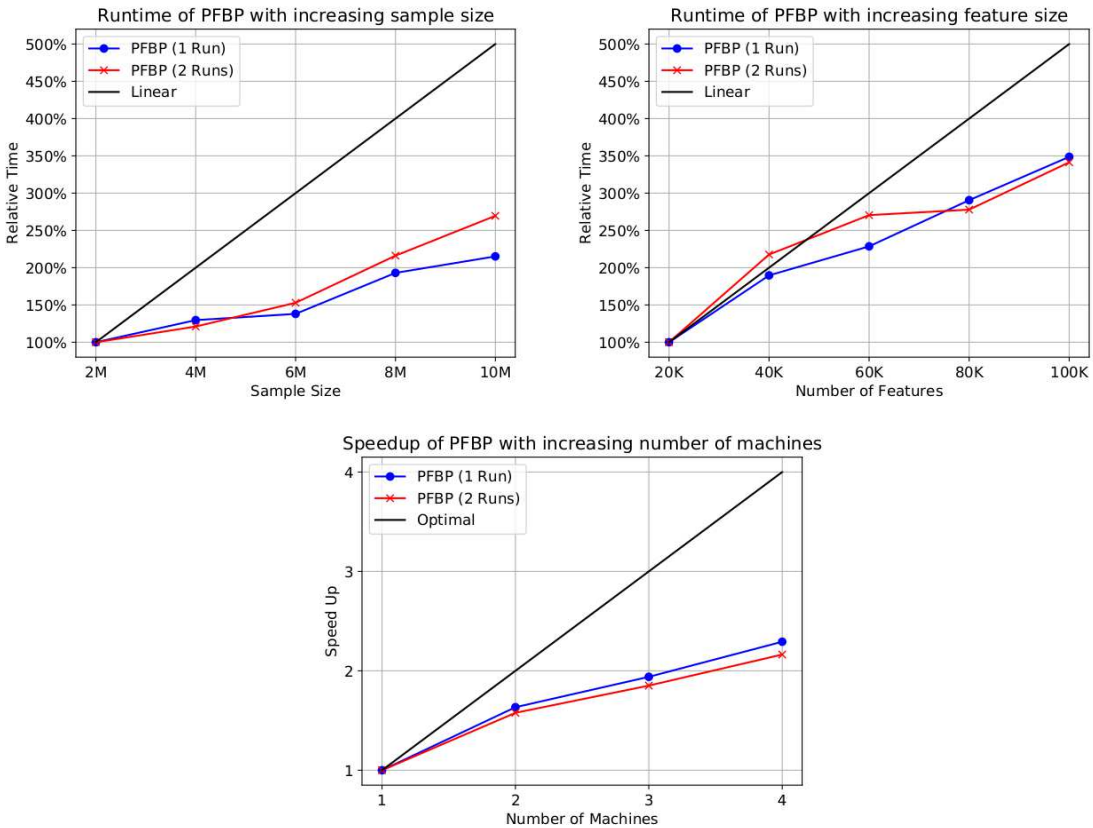


Figure 5.2: Scalability of PFBP with increasing sample size (top left), feature size (top right) and number of machines (bottom). Time and speed-up were computed relatively to the first point on the x-axis, for the same number of Runs. PFBP improves super-linearly with sample size, linearly with feature size and running time is reduced linearly with increasing number of machines. The results are similar for PFBP with 1 run and 2 runs.

Appendix C.1.

For each experimental setting, we generated 5 Bayesian networks, and sampled one dataset from each. The connectivity parameter C was set to 10 (i.e., the average degree of each node), the class distribution of T was 50/50, and the variance of the error term was set to 1. To investigate scalability in terms of sample size, we fixed the number of features to 1000 and varied the sample size from 2M to 10M. Scalability in terms of feature size was evaluated on datasets with 100K samples and varying the feature size from 20K to 100K, all of which also included the optimal feature set (i.e. the Markov blanket of T). Finally, scalability in terms of number of workers was investigated on datasets with 10K variables and 1M samples. The maximum number of variables *maxVars* to select was set to 50.

The results are summarized in Figure 5.2. On the top row we show the relative

Table 5.1: Binary classification datasets used in the comparative evaluation

Name	#Samples	#Variables	Non-zeros per Row
SUSY	5000000	18	17.79
HIGGS	11000000	28	25.79
covtype.binary	581012	54	11.88
epsilon	500000	2000	2000.00
rcv1.binary	697641	47236	73.15
avazu-app	14596137	1000000	15.00
avazu-site	25832830	1000000	15.00
criteo	45840617	1000000	39.00
news20.binary	19996	1355191	454.99
url	2396130	3231961	115.63
webspam	350000	16609143	3727.71
kdd2010a	8407752	20216830	36.35
kdd2010b	19264097	29890095	29.40

runtime of PFBP with varying sample size (left) and number of variables (right), respectively. The bottom figure shows the speed-up achieved with varying the number of workers. Relative time and speed up are computed with respect to the lowest point on the x-axis. We can clearly see that: (Top Left) PFBP improves super-linearly with sample size; in other words, feeding twice the number of rows to the algorithm requires less than double the time. This characteristic can be attributed to the Early Stopping and Early Return heuristics. (Top Right) PFBP scales linearly with number of features due to the Early Dropping heuristic and (Bottom) PFBP is able to utilize the allocated machines, although the speed-up factor does not reach the theoretical optimum. The reason for this is that the Early Stopping heuristic quickly prunes many features from consideration after processing the first Group sample, reducing parallelization in subsequent Groups as only few features remain Alive.

5.5.3 Comparative Evaluation of PFBP on Real Datasets

We evaluated the PFBP algorithm on 13 binary classification datasets, collected from the LIBSVM dataset repository[4], with the constraint that each dataset contains at least 500K samples or variables. A summary of the datasets, shown in order of increasing variable size, is shown in Table 6.1. The first two columns show the total number of samples and variables of each dataset, while the last column shows the average number of non-zero elements of each sample. The maximum number of non-zero elements equals the total number of variables. Except for the first four datasets, all

[4]http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/

other datasets are extremely sparse.

All algorithms were compared in terms of classification accuracy and running time. To estimate the classification accuracy, 10% of the training instances were randomly selected and kept out. The remaining 90% were used by each algorithm to select a set of features and to train a logistic regression model using those features. The maximum number of features to select was set to 50. We note that, for PFBP, the backward phases and the second phase were only executed if the algorithm terminated (i.e., the remaining variables were empty) before the variable limit was reached. This was done because PFBP would not have terminated otherwise (i.e., the first phase would still have variables to consider), and thus would not have executed the extra phases. A timeout limit of 12 hours was used for each algorithm. In case an algorithm did not terminate within the time limit, the number of features selected up to that point are reported. If no feature was selected, the accuracy was set to N/A (not available). For PFBP, we used the data partitioning strategy described in Section 5.3. For the remaining methods, the number of partitions was set to 4 times the total number of Spark tasks. We ran 6 Spark tasks, each one using 13 cores, on each of the 5 workers. Thus, the total number of partitions was set to 120.

Comparison of PBFP with Forward Selection Based Methods

We compared PFBP to 3 forward selection based algorithms: (i) Single Feature Optimization (SFO) [135], (ii) Forward-Backward Selection (FBS), and (iii) Univariate Feature Selection (UFS). UFS and FBS were implemented using a parallelized implementation of standard binary logistic regression for Big Data provided in the Spark MLLib [104].

Table 5.2 shows the running times of the algorithms (rounded up to the closest minute), and Table 5.3 show the classification accuracy and the number of selected variables. We included the results of the trivial classifier, which assigns each sample to the most frequent class, and thus attains an accuracy equal to the frequency of the most common class.

It can be seen that PFBP outperforms all competing methods in terms of running time. SFO, UFS, and FBS only terminate selecting at least 1 feature in the smallest, first 4 datasets. UFS and FBS reach the timeout limit and do not select a single feature even for the moderately sized rcv1 dataset, which contains only 47K variables and 698K samples, while SFO is able to select only a single feature in 12 hours [5]. PFBP is able to terminate for all datasets within 12 hours, taking a maximum of 10.5 hours for the kdd2010b dataset which contains 19M samples and 30M variables.

[5]We tried running SFO, UFS and FBS on some of the large datasets using a timeout limit of 2 days, the maximum possible on the cluster we used; however, none of the algorithms were able to select even a single variable

Table 5.2: The table shows the total running time for each algorithm and dataset. The fastest algorithms are shown in bold, while algorithms that timed out are indicated with an asterisk. PFBP significantly outperforms all competitors in terms of running time, and is the only algorithm that is able to terminate for all datasets within the given time limit of 12 hours. Furthermore, except for 2 cases (FBS on the epsilon dataset and SFO on the rcv1 dataset; see Table 5.3), none of the competing algorithms were able to select a single variable within 12 hours.

	Running Time (HH:MM)			
Dataset	PFBP	SFO	UFS	FBS
SUSY	**00:01**	00:09	00:02	00:40
HIGGS	**00:02**	00:16	00:03	01:59
covtype	00:09	01:05	**00:04**	05:17
epsilon	**00:02**	02:43	00:49	12:00*
rcv1	**00:15**	12:00*	12:00*	12:00*
avazu-app	**04:23**	12:00*	12:00*	12:00*
avazu-site	**05:43**	12:00*	12:00*	12:00*
criteo	**03:15**	12:00*	12:00*	12:00*
news20	**00:44**	12:00*	12:00*	12:00*
url	**01:48**	12:00*	12:00*	12:00*
webspam	**06:13**	12:00*	12:00*	12:00*
kdd2010a	**06:37**	12:00*	12:00*	12:00*
kdd2010b	**10:34**	12:00*	12:00*	12:00*

In terms of predictive performance, *PFBP always produces the best or an equally predictive model*. When a competitor produces a better model the difference is in order of 0.01% of accuracy. Some larger differences are observed only when the final model is fit with the SparkLR. *In terms of the number of features*, *PFBP always selects the lowest number of features* to achieve the same or better performance. An exception is when FBS timed-out for the epsilon dataset selecting 10 features vs. 50 for PFBP; however, the lower number of features comes with a significant drop in performance of about 10% of accuracy.

Comparison of PBFP with Information Theoretic Feature Selection Methods

Next, we compare PFBP to three algorithms of the family of information theoretic feature selection (ITFS) methods [22]: the Minimum-Redundancy Maximum-Relevance (MRMR) algorithm [118], the Joint Mutual Information (JMI) algorithm [161] and the Conditional Mutual Information Maximization algorithm [55]. Those methods were chosen as they have been shown to be perform well compared to several other members of the ITFS family [22]. For all of the above algorithms, we used existing Spark-based implementations[6] [121].

As information-theoretic feature selection methods require discrete data, we per-

[6]https://github.com/sramirez/spark-infotheoretic-feature-selection

Table 5.3: The table shows the number of selected variables and the classification accuracy of forward-selection based algorithms on all datasets. Classification accuracy is obtained by combining models CombLR (see Section 5.1.5) as well as using the default MLlib logistic regression implementation, SparkLR. Bold numbers show the best performing method for a given classifier, while numbers highlighted in red indicate that there is a significant difference (> 1%) between the predictive performance obtained using the classifiers, or that the classifier performs worse than the trivial classifier. Overall, all feature selection methods perform similarly, with PFBP and SFO typically having the best predictive performance. PFBP achieves the better or on par performance by selecting fewer variables than its competitors. Furthermore, in most cases, combining models CombLR works as well or better than the logistic regression of MLlib, SparkLR.

	Classification Accuracy (%)									#Selected Variables			
	CombLR				SparkLR								
Dataset	PFBP	SFO	UFS	FBS	PFBP	SFO	UFS	FBS	Trivial	PFBP	SFO	UFS	FBS
SUSY	78.91	78.91	78.90	**78.91**	78.59	78.59	**78.61**	78.59	54.22	12	14	18	13
HIGGS	64.26	**64.27**	64.27	**64.27**	64.12	64.15	**64.16**	64.15	53.06	16	18	28	18
covtype	**75.16**	71.02	57.77	57.59	75.80	75.74	**76.00**	73.57	50.78	34	44	50	48
epsilon	**86.05**	85.84	80.08	76.39	**86.07**	85.82	80.02	76.33	51.05	50	50	50	10
rcv1	**91.46**	64.86	N/A	N/A	**91.28**	64.86	N/A	N/A	52.71	50	1	0	0
avazu-app	88.16	N/A	N/A	N/A	87.80	N/A	N/A	N/A	88.12	50	0	0	0
avazu-site	80.48	N/A	N/A	N/A	80.07	N/A	N/A	N/A	80.14	50	0	0	0
criteo	76.43	N/A	N/A	N/A	76.37	N/A	N/A	N/A	74.41	50	0	0	0
news20	85.15	N/A	N/A	N/A	83.19	N/A	N/A	N/A	51.47	50	0	0	0
url	96.93	N/A	N/A	N/A	97.13	N/A	N/A	N/A	67.11	50	0	0	0
webspam	98.08	N/A	N/A	N/A	98.03	N/A	N/A	N/A	60.42	50	0	0	0
kdd2010a	86.12	N/A	N/A	N/A	57.11	N/A	N/A	N/A	85.33	50	0	0	0
kdd2010b	86.16	N/A	N/A	N/A	44.18	N/A	N/A	N/A	86.09	50	0	0	0

formed a simple discretization method on all sparse datasets (i.e., except for SUSY, HIGGS, covtype and epsilon), by setting the value to 0 if the original value was 0, and to 1 otherwise. Thus, a 0 or 1 indicates the absence or presence of a value respectively. Although this discretization method may be sub-optimal, it still allows for a fair comparison between all methods, as they are all executed on the same data. Furthermore, discretization to more than 2 values would put PFBP at a disadvantage, as the independence tests based on logistic regression models would need to fit models for more parameters. Specifically, if a discrete variable takes K values, logistic regression would need to fit $K-1$ coefficients. As we will see below, this discretization method does not significantly affect the results in terms of classification accuracy, justifying its use in this case. On the contrary, in some cases the produced models have a higher accuracy than the ones obtained from the original data.

The results are summarized in Tables 5.4 and 5.5. They show the running time in hours and minutes (rounded up to the closest minute) and the classification accuracy for each algorithm. The number of selected variables is not shown, as all algorithms

Table 5.4: The table shows the total running time of each algorithm on all discretized datasets. The fastest algorithm for each dataset is shown in bold. ITFS methods significantly outperform PFBP in terms of running time, being almost 23 times faster than PFBP (for the avazu-app dataset).

	Running Time (HH:MM)			
Dataset	PFBP	MRMR	JMI	CMIM
rcv1	00:13	**00:06**	00:06	00:07
avazu-app	06:00	00:16	**00:16**	00:16
avazu-site	06:02	00:29	**00:25**	00:32
criteo	03:56	**01:15**	01:36	01:23
news20	01:02	00:03	**00:03**	00:04
url	02:20	00:15	**00:14**	00:16
webspam	02:00	**00:53**	01:04	00:58
kdd2010a	07:03	00:25	**00:22**	00:23
kdd2010b	08:49	00:43	**00:40**	00:48

Table 5.5: The table shows the classification accuracy % for each algorithm and dataset. Classification accuracy is obtained by combining models (see Section 5.1.5) as well as using the default MLlib logistic regression implementation. Bold numbers show the best performing method for a given classifier, while numbers highlighted in red indicate that the classifier performs worse than the trivial classifier. In most cases, PFBP produces better methods, often significantly so, having a higher accuracy of up to 5-9% on the rcv1, news20 and webspam datasets. As before, in most cases, combining models works as good or better than the implementation in MLlib.

	Classification Accuracy (%)								
	CombLR				SparkLR				
Dataset	PFBP	MRMR	JMI	CMIM	PFBP	MRMR	JMI	CMIM	Trivial
rcv1	**90.64**	86.14	85.64	85.60	**90.87**	86.35	86.20	85.63	52.71
avazu-app	**88.19**	88.14	87.89	88.15	87.45	87.54	88.11	88.08	88.12
avazu-site	80.48	**80.50**	79.74	79.79	79.77	80.33	**80.42**	**80.42**	80.14
criteo	**76.39**	76.19	75.98	75.91	**76.32**	75.92	75.98	75.91	74.41
news20	**86.03**	81.22	79.79	79.27	**83.16**	77.43	77.12	78.30	51.47
url	**96.80**	94.87	95.79	95.07	**96.98**	95.77	95.36	95.70	67.11
webspam	**98.22**	94.30	94.62	93.92	**98.22**	89.52	91.23	90.88	60.42
kdd2010a	86.17	**86.27**	86.15	86.10	59.15	38.89	30.20	29.58	85.33
kdd2010b	**86.10**	86.08	86.07	86.07	43.17	37.98	38.22	37.49	86.09

terminated within the time-limit of 12 hours and selected 50 variables.

As expected, *regarding running time, ITFS methods clearly outperform PFBP, being about 2-23 times faster than PFBP*. As explained in the discussion in Section 5.4.3, this is because PFBP treats data as dense and is not specific or optimized for discrete data. In addition, PFBP's current implementation is based on fitting logistic regression models that require iterative techniques, while ITFS methods only require the counts in the contingency tables of the joint distributions.

Table 5.6: Difference in classification accuracy between models obtained using CombLR and SparkLR across all experiments. Positive values indicate that CombLR performs better. In the continuous data from the comparison between PFBP, SFO, UFS and FBS, N/A values correspond to cases where the algorithm did not terminate. For the discretized data, N/A values correspond to cases where the experiment was not performed. In all cases, PFBP using CombLR produces models with similar or better performance than SparkLR.

	Continuous Data				Discretized Data			
Dataset	PFBP	SFO	UFS	FBS	PFBP	MRMR	JMI	CMIM
SUSY	0.32	0.32	0.29	0.32	N/A	N/A	N/A	N/A
HIGGS	0.14	0.12	0.11	0.12	N/A	N/A	N/A	N/A
covtype	-0.64	-4.72	-18.23	-15.98	N/A	N/A	N/A	N/A
epsilon	-0.02	0.02	0.06	0.06	N/A	N/A	N/A	N/A
rcv1	0.18	0.00	N/A	N/A	-0.21	-0.21	-0.76	-0.03
avazu-app	0.36	N/A	N/A	N/A	0.74	0.70	-0.22	0.07
avazu-site	0.41	N/A	N/A	N/A	0.71	0.17	-0.68	-0.63
criteo	0.06	N/A	N/A	N/A	0.07	0.27	0.00	0.00
news20	1.96	N/A	N/A	N/A	2.87	3.79	2.67	0.97
url	-0.20	N/A	N/A	N/A	-0.18	-0.90	0.43	-0.63
webspam	0.05	N/A	N/A	N/A	0.00	4.78	3.39	3.04
kdd2010a	29.01	N/A	N/A	N/A	27.02	47.38	55.95	56.52
kdd2010b	41.98	N/A	N/A	N/A	42.93	48.10	47.85	48.58

In terms of classification accuracy, PFBP outperforms ITFS methods in most cases. In some cases (rcv1, news20, webspam) the accuracy difference is more than 4%; when PFBP does not produce the best model, the accuracy difference is less than 0.1%. Thus, overall, if the goal is predictive accuracy, PFBP should be preferred over ITFS methods.

Comparison of CombLR and SparkLR

We proceed by comparing the performance obtained using different logistic regression classifiers, namely SparkLR and CombLR. We remind the reader that CombLR comes at no additional computational overhead by combining the coefficients of local models already produced by PFBP for feature selection purposes. SparkLR in contrast, fits a global LR model, with a corresponding computational overhead. Table 5.6 shows the difference in accuracy obtained by CombLR and SparkLR over all experiments, with positive values corresponding to cases where CombLR outperforms SparkLR.

For the comparison between PFBP, SFO, UFS and FBS on the original datasets, CombLR slightly outperforms SparkLR on most datasets. There are however a few cases where large differences between both classifiers can be seen. For covtype, CombLR results in a large performance drop for SFO, UFS and FBS. However, the performance of PFBP is similar, regardless of the method used for producing the classifier. For PFBP, for kdd2010a and kdd2010b, SparkLR completely fails, achieving an

accuracy lower than the one obtained by the trivial classifier (see Table 5.3). Regarding the comparison of PFBP with information-theoretic methods on the discretized data, we observe again a qualitatively similar behavior as in the previous experiments. The problematic datasets seem to be news20, webspam, kdd2010a and kdd2010b. As before, there are cases where SparkLR fails to produce a model better than the trivial classifier, whereas CombLR is more robust overall. The difference may exceed 50% of accuracy! When CombLR fails to beat the trivial classifier the accuracy difference is less than 0.5% of accuracy; this case never happen for the PFBP algorithm, arguably due to a better selection of features with less collinearities (deterministic relations). In any case, *it is encouraging that in all cases, PFBP using CombLR produces models on par or better than SparkLR.*

Unfortunately, we were not able to determine the cases where SparkLR fails. Specifically, we tried to (a) vary the number of partitions used, and (b) relax the stopping conditions used by the model fitting procedures (increasing number of iterations and reducing tolerance of the termination criterion), but neither of those made any difference.

5.5.4 Proof-of-Concept Application on genetic SNP Data

Single Nucleotide Polymorphisms (SNPs)[7], the most common type of genetic variation, are variations of a single nucleotide at specific loci in the genome of a species. The Single Nucleotide Polymorphism Database (dbSNP) (build 150) [133] now lists 324 million different variants found in sequenced human genomes[8]. In several human studies, SNPs have been associated with genetic diseases or predisposition to disease or other phenotypic traits. As of 2018-05-12, the GWAS Catalog[9] contains 3379 publications and 61620 unique SNP-trait associations. Large scale studies under way (e.g., Precision Medicine Initiative [32]) intend to collect SNP data in large population cohorts, as well as other medical, clinical and lifestyle data. The resulting matrices may end up with millions of rows, one for each individual, and variables (SNP or some other measured quantity). Thus, we believe that investigating the behavior of newly proposed Big Data FS algorithms on such data is worthwhile. A proof-of-concept application of PFBP is presented next.

[7]https://ghr.nlm.nih.gov/primer/genomicresearch/snp

[8]https://ncbiinsights.ncbi.nlm.nih.gov/2017/05/08/dbsnps-human-build-150-has-doubled-the-amount-of-refsnp-records/

[9]https://www.ebi.ac.uk/gwas/

SNP Data Generation and Setup

We simulated genotype data containing 500000 individuals (samples) and 592555 SNP genotypes (variables), following the procedure described in [25]. As SNP data are dense, they require approximately 2.16 TB of memory, and thus are more challenging to analyze than sparse data, such as the ones used in the previous set of experiments. The data were simulated with the HAPGEN 2 software [27] from the Hapmap 2 (release 22) CEU population [33]. A more detailed description of the data generation procedure is given in Appendix C.2.

We used $M = 100$ randomly selected SNPs to generate a binary phenotype (outcome), as described in [25] (see also Appendix C.2). The optimal accuracy using all 100 SNPs is 81.42%. Ideally and given enough samples, any feature selection method should select those 100 SNPs and achieve an accuracy around 81.42%. Due to linkage disequilibrium however, many neighboring SNPs are highly correlated (collinear) and as a consequence offer similar predictive information about the outcome and are informationally equivalent. Therefore, a high accuracy can be achieved even with SNPs other than the 100 used to simulated the outcome.

We used 95% of the samples as a training set, and 5% as a test set for performance estimation. We set a timeout limit of 15 hours, and used the same setup as used in previous experiments, with the exception that the maximum number of variables to select was set to 100.

Repartitioning to Reduce Memory Requirements

For big, dense data such as the SNP data considered in this experiment which require over 2 TB of memory, a direct application of PFBP as used in other experiments is possible, but may be unnecessarily slow. We found that for such problems, it makes sense to repartition the data at some point, if enough variables have been removed by the Early Dropping heuristic. Repartitioning and discarding dropped variables reduces storage requirements, and may offer a significant speed boost. It is an expensive operation however, and should only be used in special situations. For the SNP data, after the first iteration only about a third of the variables remained, reducing the memory requirements to less than 1 TB, and thus most (if not all) of the data blocks were able to fit in memory. In this case repartitioning makes sense, as its benefits far outweigh the computational overhead.

Results on the SNP Data

PFBP was able to select 84 features in 15 hours, using a total of 960 core hours. It achieved an accuracy of 77.62%, which is over 95% of the theoretical optimal accuracy.

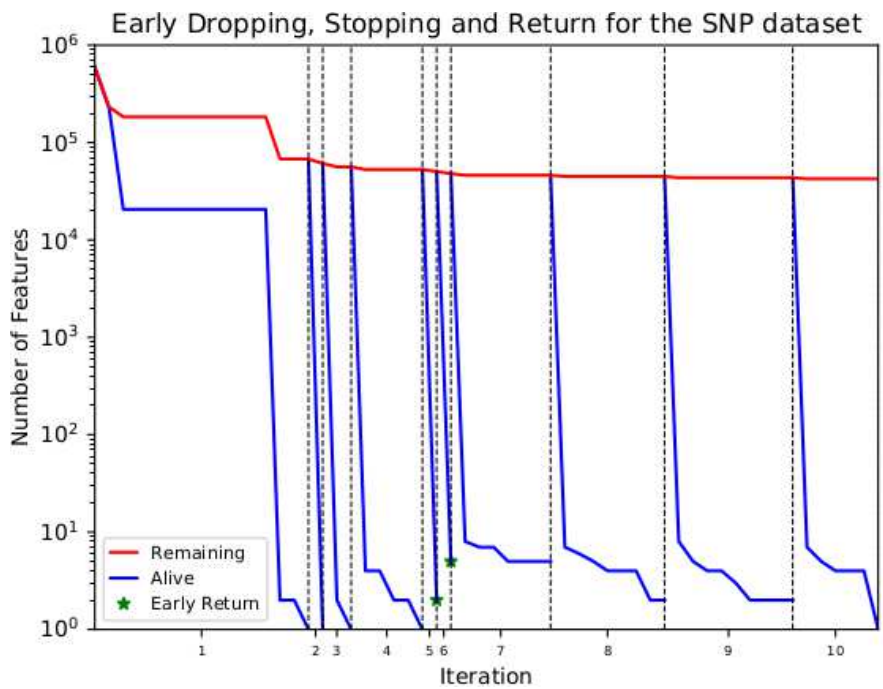


Figure 5.3: The effects of the early pruning heuristics is shown for the first 10 forward iterations on the SNP data. The y-axis shows the number of variables on a logarithmic scale. The width of each iteration is proportional to the number of groups processed. The early dropping heuristic is able to quickly discard many features, reducing them by about an order of magnitude. Early stopping filters out most variables after processing the first group, and early return is applied two times.

The results are very encouraging; in comparison the DISSECT software [25] took 4 hours on 8400 cores (that is, 33600 core hours) and using 16 TB of memory to fit a mixed linear model on similar data, and to achieve an accuracy around 86% of the theoretical maximum. The two experiments are not directly comparable because (a) the outcome in our case is binary instead of continuous requiring logistic regression instead of linear regression models favoring DISSECT in terms of computational time, (b) the scenarios simulated in [25] had larger Markov blankets (1000 and 10000 instead of 100) favoring PFBP (although, their results are invariant to the size of the Markov blanket). Nevertheless, the reported results are still indicative of the efficiency of PFBP on SNP Big Data.

Figure 5.3 shows the effects of the heuristics used by PFBP for the first 10 iterations. The y-axis shows the number of Remaining and Alive features on a logarithmic scale. The x-axis shows the current iteration, and the width is proportional to the total number of Groups processed in that iteration. We observe that (a) Early Dropping discards many features in the first iteration, reducing the number of Remaining features

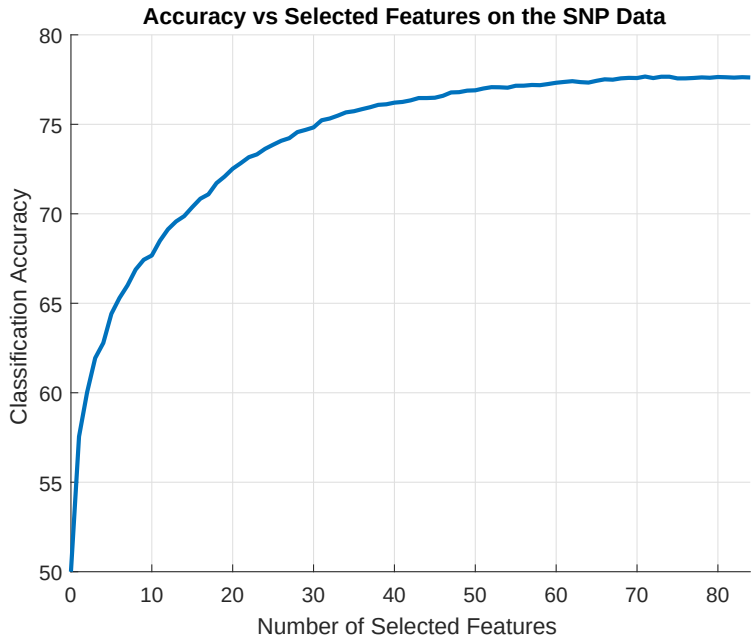


Figure 5.4: The figure shows how the accuracy of PFBP on the SNP data increases as it selects more features. The models are produced by PFBP at each iteration with minimal computational overhead. In the first few iteration, accuracy increases sharply, while in the later iterations a plateau is reached, reaching a value of 77.59% with 70 features, with the maximum being 77.62% with 84 features. This could be used as a criterion to stop feature selection early.

by about an order of magnitude, (b) in most iterations, Early Stopping is able to reduce the number of Alive features to around 10 after processing the first Group, (c) Early Return is applied 2 times, ending the Iteration and selecting the top feature after processing a single Group.

Finally, by combining the intermediate logistic regression models at each Iteration using CombLR, we computed the accuracy at each iteration of PFBP with no additional overhead. This provides an insight regarding its predictive performance behavior with increasing number of selected features. As before, the accuracy is computed on the 5% of the data that were kept out as a test set. Such information could be used to decide early whether a sufficient number of features has been selected, and to stop computation if the accuracy reaches a plateau. This is often the case, as most important features are typically selected during the first few iterations. This task can be performed using PFBP with minimal computational overhead, as the local models required to approximate a full global model (see Section 5.1.5) are already available. The results are shown in Figure 5.4. As expected, the largest increase in accuracy is obtained after selecting the first few features, reaching an accuracy of 75%

even after selecting only 30 features. In addition, after selecting about 70 features, the accuracy increases only marginally afterwards, increasing from 77.59% with 70 features to 77.62% with 84. Thus, computation could be stopped after 70 features have been selected, and still attain almost the same accuracy.

5.5.5 Summary and Discussion of Experimental Results

Overall, the experiments indicate that PFBP scales superlinearly with the available sample size, and linearly with the number of features and available workers. Compared with other algorithms in its class, namely forward selection-based algorithms with map-reduce implementations, under the same conditions (type of test and predictive model), PFBP dominates the alternatives (UFS, FBS, SFO) in terms of execution time, number of selected features, and predictive performance. Against information-theoretic variants specialized for discrete and sparse data with available map-reduce implementations, PFBP performs worse in terms of running time, however, it is still applicable and practical to apply to large datasets. However, PFBP dominates the information-theoretic variants in terms of predicting performance. Furthermore, as a side product of the experiments, we compared two logistic regression algorithms, namely SparkLR that is available in MLlib and fits in a parallelized fashion a global logistic regression model, and CombLR that combines the coefficients of local logistic regression models. CombLR always converges, providing on average more predictive models than SparkLR, and it is considerably more efficient than SparkLR even when computed from scratch and not during PFBD. Finally, the proof-of-concept application to SNP data demonstrates that the emergence of Big genetic Data can become amenable to analysis using algorithms such as PFBP. A detailed trace of the computational experiment shows the effectiveness of the Early Stop, Drop and Return heuristics of PFBP: (a) after the first few iterations the Remaining features are reduced by 1b•“2 orders of magnitude. (b) The number of Alive features drops exponentially as more groups are processed. The trace visualizes PFBPb••s scalability properties.

Chapter 6
Extending Greedy Feature Selection Algorithms to Multiple Solutions

Up to this point we considered the single feature selection problem, and proposed several extensions of the Forward-Backward Selection algorithm to improve its computational efficiency, by introducing several heuristics and proposing a method for parallelizing it. Next, we will consider theory and algorithms for the **multiple feature selection problem**, and show how algorithms like forward-backward selection can be extended to identify all solutions.

Definition 7 (Multiple Feature Selection Problem)**.** *Let* $\mathbf{S}$ *be the solution to the single feature selection problem (called the* ***reference*** *solution). The solution* $\mathbf{M}$ *to the multiple feature selection problem consists of all minimal-size sets* $\mathbf{S_i} \subseteq \mathbf{F}$ *that are statistically equivalent to* $\mathbf{S}$.

There are several ways to define and test statistical equivalence between feature sets, each of which can lead to a different solution set $\mathbf{M}$ to the above problem, which will be addressed in Section 6.2.

6.1 A Taxonomy of Features in the Presence of Multiple Solutions

In the presence of multiple solutions, the JKP taxonomy of features [78] is counter-intuitive and misleading. Intuitively, we'd expect that keeping only the strongly relevant features should be enough for optimal prediction using an optimal classifier, as weakly relevant features are superfluous. Consider however the case where, feature X is in a reference solution and X' is a copy of X (or a one-to-one deterministic transformation). Now, both features are weakly relevant (one makes the other redundant); one expects that both should be filtered out (not selected). However, at least one of them should be selected for optimal prediction. In fact, if all members of a reference solution have a copy in the dataset, then no feature is strongly relevant for the given

problem, which is counter-intuitive. The problem stems from the fact that the JKP taxonomy does not distinguish between weakly relevant features that carry superfluous information, and features that carry information necessary for optimal prediction but this information component is shared among many features or feature subsets. The above considerations lead us to define the following taxonomy of features:

Definition 8 (Irrelevant Feature)**.** *A feature X is irrelevant if it provides no information for T in any context $\mathbf{Z}$, i.e., if $\forall \mathbf{Z} \subseteq \mathbf{F} \setminus \{X\}, T \perp X \mid \mathbf{Z}$.*

Definition 9 (Indispensable Feature)**.** *A feature X is indispensable if it belongs in all solutions for T, i.e., $\forall \mathbf{S}_i \in \mathbf{M}, X \in \mathbf{S}_i$.*

Definition 10 (Replaceable Feature)**.** *A feature X is replaceable if it is not indispensable and belongs in some solution for T, i.e., $\exists \mathbf{S}_i, \mathbf{S}_j \in \mathbf{M}, \mathbf{S}_i \neq \mathbf{S}_j \wedge X \in \mathbf{S}_i \wedge X \notin \mathbf{S}_j$.*

Definition 11 (Redundant Feature)**.** *A feature X is redundant if it provides information for T in some context but does not belong in any solution, i.e., $\exists \mathbf{Z} \subseteq \mathbf{F} \setminus \{X\}, T \not\perp X \mid \mathbf{Z} \wedge \forall \mathbf{S}_i \in \mathbf{M}, X \notin \mathbf{S}_i$.*

The proposed taxonomy is related to the JKP taxonomy as follows: (a) irrelevant features are defined the same in both taxonomies, (b) strongly relevant features coincide with indispensable features, (c) a weakly relevant feature is called replaceable if it is present in some solution, and is called redundant otherwise. Thus, when there is a single solution, then irrelevant, strongly relevant, and weakly relevant features coincide with irrelevant, indispensable, and redundant features respectively.

6.2 Statistically Equivalent Feature Sets

In this section, we will provide several definitions and tests for statistical equivalence of feature sets. Such tests are used by the main algorithm proposed in Section 6.4 for identifying multiple solutions. We review methods from related statistical literature on model selection, and show how they can be used to test different types of statistical equivalence of feature sets. Finally, we provide some advice for performing such tests in practice.

We proceed with some definitions used hereafter. Let $\mathcal{H}$ be a predictive algorithm (also called learner). Examples are linear and logistic regression, as well as support vector machines and random forests, where hyper-parameter values (such as the kernel and cost for support vector machines) are fixed. For given target T and set of random features $\mathbf{X}$, we denote with $\mathcal{H}^*(T|\mathbf{X})$ the optimal model attainable in the sample limit using algorithm $\mathcal{H}$ and features $\mathbf{X}$ for predicting T. Given a dataset $\mathcal{D}$ sampled from the joint distribution of T and $\mathbf{F}$ (the set of all features), we denote with $\hat{\mathcal{H}}(T|\mathbf{X})$ the

model obtained by $\mathcal{H}$ on the dataset $\mathcal{D}$ for T using only features $\mathbf{X}$. Let $\mathcal{L}$ be a loss (or performance) function, such as the squared error for regression tasks and the deviance for probabilistic classification (such as logistic regression), as well as penalized versions of the above such as the Akaike [2] and Bayesian information criteria [132]. We use $E[\mathcal{L}(\mathcal{H}^*(T|\mathbf{X}))]$ to denote the expected loss of $\mathcal{H}^*(T|\mathbf{X})$ with respect to the true joint distribution of T and $\mathbf{X}$.

6.2.1 Definitions of Statistical Equivalence of Feature Sets

We proceed by listing different definitions for assessing equivalence of feature sets. Tests for them are presented in the subsequent section.

Definition 12 (Performance Equivalence (**PEQ**) (see also Figure 11 in [138])**.** *Feature sets* $\mathbf{X}$ *and* $\mathbf{Y}$ *are performance equivalent relative to model* $\mathcal{H}$ *and loss function* $\mathcal{L}$ *if* $E[\mathcal{L}(\mathcal{H}^*(T|\mathbf{X}))] = E[\mathcal{L}(\mathcal{H}^*(T|\mathbf{Y}))]$.

PEQ requires that the predictive models obtained by two sets features have the same expected loss, and depends both on $\mathcal{H}$ and $\mathcal{L}$. A drawback of performance equivalent feature sets is that it is not guaranteed that they are Markov blankets (i.e., optimal solutions). PEQ can hold even if the feature sets predict separate parts of the outcome's distribution equally well. For example, let $X \backsim \mathcal{N}(0,1)$, $Y \backsim \mathcal{N}(0,1)$ and $T = X + Y$. It can be seen that the feature sets $\{X\}$ and $\{Y\}$ are PEQ for T relative to any performance measure and linear models. However, none of them is a Markov blanket; the Markov blanket of T is the union of both feature sets. Another problematic case is if the loss function used is not a proper scoring function, i.e., if the minimum loss is not achieved for optimal feature sets. This can be the case for performance functions such as the classification accuracy. For example, let T and X be binary features, and let $P(T = 1|X = 1) = 0.9$, $P(T = 1|X = 0) = 0.6$ and $P(X = 1) = 0.5$. Any optimal model optimizing accuracy would always predict $T = 1$, irrespective of the value of X, attaining an accuracy of 75%. Thus, both feature sets $\{\emptyset\}$ and $\{X\}$ are performance equivalent using classification accuracy. However, only $\{X\}$ is a Markov blanket, as X clearly gives information about T (X is dependent with T). Therefore, if the main goal is knowledge discovery, the PEQ definition for identifying equivalences may not be preferable.

Definition 13 (Model Equivalence (**MEQ**))**.** *Feature sets* $\mathbf{X}$ *and* $\mathbf{Y}$ *are model equivalent relative to model* $\mathcal{H}$ *if* $\mathcal{H}^*(T|\mathbf{X}) = \mathcal{H}^*(T|\mathbf{Y})$.

MEQ requires that T can be modeled using $\mathcal{H}$ equally well with either set of features. Intuitively, MEQ implies that the predictions of models obtained using either feature set are identical. Thus, MEQ directly implies PEQ, irrespective of the loss function $\mathcal{L}$. Naturally, the opposite does not necessarily hold.

Definition 14 (Information Equivalence (**IEQ**)). [1] *Feature sets* $\mathbf{X}$ *and* $\mathbf{Y}$ *are information equivalent if both* $T \perp \mathbf{X} \mid \mathbf{Y}$ *and* $T \perp \mathbf{Y} \mid \mathbf{X}$ *hold.*

IEQ [93, 138] is the strictest definition of equivalence, and is independent both of $\mathcal{H}$ and $\mathcal{L}$. It requires that feature sets are **interchangeable**: they should contain the same information about the outcome. For example, let $X \backsim \mathcal{N}(0,1)$, $Y = \beta X$ and $T = X + Y$. In this case we say that $\{X\}$ and $\{Y\}$ are information equivalent, as Y depends deterministically on X. Intuitively, choosing any of them gives the same information about T. In the taxonomy proposed previously, X and Y are called replaceable.

We note that in practice the definition of IEQ may also implicitly depend on some predictive algorithm. That is because the tests of conditional independence used for testing IEQ may also use a predictive algorithm (e.g., the logistic regression algorithm used by nested likelihood-ratio tests).

Thus, there may be special cases of predictive algorithms where MEQ implies IEQ, but in general this is not the case. Specifically, it is not the case if combining both feature sets leads to a better set of features (i.e., if the feature sets were not Markov blankets). For example, let X, Y and W be binary variables, and $T = W \vee (X \oplus Y)$, where $\oplus$ is the logical XOR operator. Assuming an optimal classifier $\mathcal{H}$, $\{X, W\}$ and $\{Y, W\}$ are MEQ relative to $\mathcal{H}$, as knowing only X or Y does not provide information for T, and thus models constructed using either variable set will give the same predictions based on W. The feature sets are not IEQ, as X and Y provide information for T conditional on X or Y respectively. The reason for the above is that neither of the feature sets is a Markov blanket, and combining them leads to a better feature set(which is the Markov blanket in this example).

6.2.2 Testing Statistical Equivalence of Feature Sets and its Relation to the Model Selection Problem

The problem of model selection can be defined as identifying the best fitting model out of a set of candidate models. Most approaches address the more general problem of selecting among two or more competing models (e.g., choosing between a normal and log-normal distribution), whereas in our case we are interested in the special case of comparing two sets of features relative to the same model (e.g., selecting the best feature set for linear regression); we will focus on the latter hereafter. Testing for equivalence is related to model selection, as it deals with the problem of identifying models that fit the data equally well instead of identifying the better one. We will briefly mention some approaches that are related to the problems of equivalence testing; interested readers may refer to [119] for an in-depth overview, as well as a discussion about the

[1] This definition is essentially identical to the one given in [93,138]. Their definition requires also that $T \not\perp \mathbf{X}$ and $T \not\perp \mathbf{Y}$ hold which always holds in the context of feature selection, and thus can be dropped.

similarities of model selection and hypothesis testing.

Vuong's Variance Test

In the context of model selection, Vuong [154] proposed the **variance test** to test if two models fit the data equally well, that is, if they are MEQ. The statistic is computed as the variance of the log likelihood-ratio [2] of the models (that is, the variance of the difference of log likelihoods). Let $\mathcal{LL}$ denote the log-likelihood function.

For two sets of features $\mathbf{X}$ and $\mathbf{Y}$, and predictive algorithm $\mathcal{H}$, the statistic is defined as follows.

$$\text{Statistic} \equiv \text{var}[\mathcal{LL}(\mathcal{H}^*(T|\mathbf{X})) - \mathcal{LL}(\mathcal{H}^*(T|\mathbf{Y}))]$$

The variance is defined with respect to the true joint distribution of T, $\mathbf{X}$ and $\mathbf{Y}$. In practice it can be estimated using the sample variance of the log-likelihood ratio of the estimated models $\hat{\mathcal{H}}(T|\mathbf{X})$ and $\hat{\mathcal{H}}(T|\mathbf{Y})$.

In case MEQ holds the statistic equals zero, and follows a sum of scaled chi-squares otherwise. As no closed-form expression for a sum of scaled chi-squares exist, it is hard to compute. Furthermore, in practice the statistic is also hard to estimate accurately, especially for small sample sizes [134]. One way to overcome those problems is to use resampling-based methods (see Section 7.2 in [119] for an example of using bootstrap tests [45]). A disadvantage of resampling-based methods such as bootstrapping is that it requires fitting two models and computing their log-likelihood for each bootstrap sample, which is very computationally demanding. Another, computationally faster approach, is to use a permutation test instead; the test is described in more detail in Section 6.2.3.

The Comprehensive Approach

Atkinson [8] proposed the **comprehensive** approach, which constructs a third model that contains both initial models as special cases. When comparing feature sets relative to the same model, this reduces to creating a third model $\hat{\mathcal{H}}(T|\mathbf{X} \cup \mathbf{Y})$, and testing whether it provides additional information compared to $\hat{\mathcal{H}}(T|\mathbf{X})$ and $\hat{\mathcal{H}}(T|\mathbf{Y})$ (see Section 3.1 in [8]). This is done by performing two likelihood-ratio tests, comparing the original models with the combined model. In case the tests are not rejected, the sets $\mathbf{X}$ and $\mathbf{Y}$ can be considered equivalent. Note that, the likelihood-ratio tests between $\hat{\mathcal{H}}(T|\mathbf{X}\cup\mathbf{Y})$ and the models $\hat{\mathcal{H}}(T|\mathbf{X})$ and $\hat{\mathcal{H}}(T|\mathbf{Y})$ corresponds to the conditional independence tests $\textsc{Test}(T; \mathbf{Y}|\mathbf{X})$ and $\textsc{Test}(T; \mathbf{X}|\mathbf{Y})$ respectively.

[2] Extensions for other loss functions also exist [58, 134].

Thus, this is a direct application of the IEQ definition using likelihood-ratio based conditional independence tests.

The J-Test

Davidson and MacKinnon [38, 99] propose several methods for choosing among two competing models, similar in spirit to Atkinson's approach. One such method is the J-test, which we describe next. Let $\hat{\mathcal{P}}(T|\mathbf{X})$ denote the predictions of model $\hat{H}(T|\mathbf{X})$ on the dataset $\mathcal{D}$ used for learning it. The idea of the J-test is similar to the comprehensive approach, but instead of testing $\textsc{Test}(T; \mathbf{X}|\mathbf{Y})$ and $\textsc{Test}(T; \mathbf{Y}|\mathbf{X})$, the J-test tests $\textsc{Test}(T; \hat{\mathcal{P}}(T|\mathbf{X})|\mathbf{Y})$ and $\textsc{Test}(T; \hat{\mathcal{P}}(T|\mathbf{Y})|\mathbf{X})$, (i.e., it tests whether the predictions of the models provide additional information for T). Basically, the J-test uses the predictions of the model as proxies for the information provided by the feature sets. That way, it avoids fitting models using the union of feature sets, and may have higher power than the comprehensive approach. However, it may not be as accurate, as it only uses the predictions of one set instead of all features, and thus interactions between feature sets $\mathbf{X}$ and $\mathbf{Y}$ may be missed. Thus, it can be seen as a more sample efficient alternative to the comprehensive approach for (approximately) testing IEQ. An overview of the J-test can be found in [21].

Paired Two-Sample Tests

Other methods for testing PEQ or MEQ, not based on model selection, are paired two-sample tests such as paired t-tests or the Wilcoxon signed rank test. MEQ can be tested by comparing the predictions of models $\hat{H}(T|\mathbf{X})$ and $\hat{H}(T|\mathbf{Y})$, while PEQ can be tested using the losses of each prediction. We note that, the latter is only applicable for loss functions which can be computed on a per-sample basis (e.g., mean squared error), and thus is not applicable for measures like the area under the ROC curve.

6.2.3 Practical Considerations and Recommendations

Discovering Multiple Markov Blankets

Although all definitions and tests of statistical equivalence are for arbitrary feature sets and not restricted to Markov blankets, in practice we are often interested in identifying multiple solutions that are Markov blankets, i.e., solving the multiple feature selection problem (see Definition 7). Recall that the solution to the multiple feature selection problem consists of all feature sets that are statistically equivalent to the reference solution $\mathbf{S}$ (a solution to the single feature selection problem).

In order to identify multiple Markov blankets, it is necessary to be able to test whether a feature set $\mathbf{X}$ is a Markov blanket. We will assume that $\mathbf{X}$ already is

minimal, i.e., no features can be removed from it without losing information about T. Given a reference solution $\mathbf{S}$ (which is a Markov blanket by definition), a way to test whether $\mathbf{X}$ is a Markov blanket is by performing a test of statistical equivalence with $\mathbf{S}$. Depending on the type of equivalence (PEQ, MEQ and IEQ) there are different guarantees for $\mathbf{X}$. If $\mathbf{X}$ is a Markov blanket all equivalences are implied, but the opposite is only guaranteed for IEQ. Thus, *when the goal is identification of multiple Markov blankets, we recommend using tests of information equivalence.*

Tests with Feature Sets that are not Markov Blankets

One subtle issue is when at least one of the tested feature sets is not a Markov blanket. This can happen in practice due to statistical errors or violations of assumptions of the tests.

Assume that the reference $\mathbf{S}$ is minimal but not a Markov blanket (i.e., some features are missing), and that we want to test whether a feature set $\mathbf{X}$, which is a Markov blanket, is equivalent to it. $\mathbf{X}$ will not be considered equivalent to $\mathbf{S}$, as it actually provides more information about T, and consequently it will not be identified as a Markov blanket. Thus, no algorithm would be able to identify any Markov blanket in this case.

Another problem is if $\mathbf{S}$ is a Markov blanket, and a feature set $\mathbf{X}$ is a superset of a Markov blanket (i.e., not minimal). In this case, $\mathbf{X}$ will falsely be identified as a Markov blanket. Although this may still be acceptable for some practical applications, as $\mathbf{X}$ contains the optimal predictive information for T, it can be problematic if an algorithm for identifying multiple solutions does not ensure that the tested feature sets are minimal, as this can lead to a large number of non Markov blanket solutions.

Both of the above are hard to detect in practice. *We propose to perform extensive hyper-parameter tuning of any multiple feature selection algorithm, in order to get reasonably good approximations of the Markov blankets* and to minimize situations where the above problems occur.

Power of IEQ Tests

Recall that in order to perform a test of IEQ between two feature sets $\mathbf{X}$ and $\mathbf{Y}$, one has to perform two tests on all of the features ($\textsc{Test}(T;\mathbf{Y}|\mathbf{X})$ and $\textsc{Test}(T;\mathbf{X}|\mathbf{Y})$). For instance, when likelihood-ratio tests are used, a model using the union of features $\mathbf{X}\cup\mathbf{Y}$ has to be created to test for IEQ. Because of that, tests of IEQ require significantly larger sample sizes than PEQ or MEQ tests, which do not fit models using the union of feature sets. When tests of IEQ are performed with small sample sizes, they may not have enough power to distinguish between non-equivalent feature sets, and will falsely consider them equivalent. To avoid this problem, *we recommend using tests for PEQ*

or MEQ as a "safe lock" before applying the IEQ test. To further reduce false positives, we recommend to combine the above with a high significance level (relative to the available sample size) for all tests.

Reliability of PEQ and MEQ Tests

To improve the reliability of the two-sample and variance tests (due to violations of assumptions or low sample size) we recommend using permutation-based variants instead. Another advantage is that permutation tests are not limited to any specific class of loss functions, and can be also applied to performance measures such as the area under the ROC curve, which are computed using the a vector of predictions. Under the null hypothesis of PEQ or MEQ, the losses or predictions can be permuted across paired samples (i.e., exchange randomly whether a prediction comes from model $\hat{H}(T|\mathbf{X})$ or $\hat{H}(T|\mathbf{Y})$). Thus, a permutation test can be performed as follows: (a) compute the statistic s on the original sample, (b) randomly permute the input vectors (predictions or losses) across the same (paired) samples, each one with probability 0.5, (c) compute the statistic s_i (of the i-th permutation) of interest on the permuted sample, and (d) repeat (b-c) B times (e.g., 1000 times). For the variance test, the p-value is then computed as the proportion of permutation statistics (including the statistic on the original sample from step (a) [40]) smaller than or equal to the sample statistic s

$$p\text{-value} \equiv \frac{1 + \sum_{i=1}^{B} I(s_i \leq s)}{B+1}$$

where I is the indicator function.

Summary

For knowledge discovery purposes, we recommend aiming for information equivalent solutions. In order to reduce the chance of false positive equivalences, we recommend to (a) perform extensive tuning of the hyper-parameter values of the feature selection algorithm, in order to increase the chance of identifying Markov blankets, (b) first apply a permutation-based variance test for PEQ or MEQ to quickly filter out false equivalences and (c) afterwards apply an IEQ test using the comprehensive approach to decide for equivalence, and (d) use relatively high significance levels to further reduce the number of false positives. In anecdotal experiments we found that all of the above were important to reduce the number of non-equivalent solutions, which in many cases was extremely high otherwise.

Algorithm 10 Template for Greedy Forward-Backward Feature Selection (TFBS)

Output: Function ORDERVARIABLES, Function BACKWARDPHASE
Input: Dataset $\mathcal{D}$, Current Solution $\mathbf{S}'$ **Output:** Solution $\mathbf{S}$
  $\mathrm{C} \leftarrow$ ORDERVARIABLES($\mathcal{D}$, $\mathbf{S}'$)
  if $\mathrm{C} = \emptyset$ **then** //*Check if forward phase terminated.*
    return BACKWARDPHASE($\mathcal{D}$, $\mathbf{S}'$)
  end if
  return TFBS($\mathcal{D}$, $\mathbf{S}' \cup \{C_1\}$) //*Select best variable C_1 and call* TFBS *recursively.*

Algorithm 11 Instantiation of TFBS for the Forward-Backward Selection Algorithm using Independence Tests

  function ORDERVARIABLES($\mathcal{D}$, $\mathbf{S}'$) //*Returns dependent variables ordered by p-value*
    //*Sort all non-selected variables that are conditionally dependent given $\mathbf{S}'$ by p-value*
    $\mathrm{C} \leftarrow$ SORTASCENDING($\{C_i \in \{\mathbf{F} \setminus \mathbf{S}'\} : \mathrm{PVALUE}(T, C_i|\mathbf{S}') \leq \alpha\}$)
    return C
  end function
  function BACKWARDPHASE($\mathcal{D}$, $\mathbf{S}'$)
    while $\mathbf{S}'$ changes **do**
      $S_i \leftarrow \underset{S_i}{\mathrm{argmax}}\,\mathrm{PVALUE}(T, S_i|\mathbf{S}' \setminus \{S_i\})$ //*Find S_i with highest p-value given $\mathbf{S}' \setminus \{S_i\}$*
      if $\mathrm{PVALUE}(T, S_i|\mathbf{S}' \setminus \{S_i\}) > \alpha$ **then**
        $\mathbf{S}' \leftarrow \mathbf{S}' \setminus \{S_i\}$ //*Remove S_i if conditionally independent given $\mathbf{S} \setminus \{S_i\}$*
      end if
    end while
    return $\mathbf{S}'$
  end function

6.3 A General Template for Forward-Backward Algorithms

We propose a general template for greedy feature selection algorithms, which we will later extend to select multiple, statistically equivalent solutions. This template can express a class of stepwise methods, namely algorithms that consist of two phases: (a) a greedy forward phase, where features are selected one at a time, and (b) an optional backward phase, applied after the forward phase terminates, to remove false positives.

The algorithm is shown in Algorithm 10, and will be referred to as TFBS hereafter. We present a recursive version of the algorithm, as it will lead to a natural extension for multiple solutions. For the sake of brevity, constant input arguments like the target variable T and hyper-parameter values are omitted. TFBS has two main components: (a) a variable ordering strategy ORDERVARIABLES, and (b) a function BACKWARDPHASE that performs the backward phase of the algorithm. In order for those functions to be **admissible**, they have to satisfy the following conditions. ORDERVARIABLES must return an ordered set of candidate variables $\mathbf{C}$ for selection, such that: (a) $\mathbf{C}$ is empty if no

more variables should be selected, (i.e., if $\mathbf{S}' \supseteq \mathbf{S}$) (b) $\mathbf{C}$ does not contain any already selected variable (i.e., $\mathbf{C} \cap \mathbf{S}' = \emptyset$), and (c) $\mathbf{C}$ contains all variables that could be selected at that iteration, in order of preference. An alternative way to look at (c) is that any variable C_{j+1} would be selected if the algorithm were to be executed on $\mathcal{D}^{C_{1:j}}$ (i.e., after excluding variables $C_{1:j}$). The BackwardPhase function must remove all and only the false positive variables. We note that, for the single solution case OrderVariables does not have to provide a complete ordering, but can only return the next variable to select. That presentation was chosen to allow for an easier extension for multiple solutions.

A large class of feature selection algorithms can be expressed as instantiations of TFBS. Examples include the forward-backward selection (FBS) algorithm [88, 156] and variations or extensions of it [100, 101, 147], information-theoretic feature selection methods [22], as well as causal-based algorithms [4] like the MMPC [146] and HITON-PC [5] algorithms. An instantiation of FBS using p-values of conditional independence tests to order variables is shown in Algorithm 11. The forward phase tests for each variable $C_i \in \{\mathbf{F} \setminus \mathbf{S}'\}$ if it is dependent with T given $\mathbf{S}'$, and selects the one with the lowest p-value. Thus, the OrderVariables function for FBS returns all variables that are conditionally dependent dependent given the current set of selected variables $\mathbf{S}'$ in ascending order of p-values. The backward phase removes at each iteration the least dependent variable given all selected variables (i.e., the one with the highest p-value which is higher than the significance level α), until no more variables can be removed.

6.4 Extending TFBS for Multiple Solutions

The forward phase of TFBS can be seen as a search on the space of feature sets [85]. Each state of the search space contains a set of selected variables $\mathbf{S}'$, and its neighbors are all states which additionally contain one of the variables in $\mathbf{C}$. As the search is only in one direction (i.e., only when variables are added), we will refer to the neighbors of a state t containing an extra variable as its **children**, the previous state as its **parent**, and all children of its parent (except for t itself) as its **siblings**. Thus, TFBS traverses that search space by only visiting the first child (i.e., the one where C_1 is selected). Given this view, we use a simple idea to extend it for multiple solutions: instead of exploring a single child at each iteration, we use **backtracking** [129] to explore all children and consider multiple solutions. A candidate solution is then returned if it is equivalent with the reference solution (i.e., the one obtained by TFBS).

The naive approach is not very practical, as it may consider the same solutions multiple times. For example, the state containing $\mathbf{S}' = \{X, Y\}$ can be reached twice by selecting the variables in different order. In general, each solution can be obtained in $m!$ ways, where m is the size of the solution. Thus, in the worst case, up to $p!$ solutions (where p is the number of variables) may be considered, even though there are only

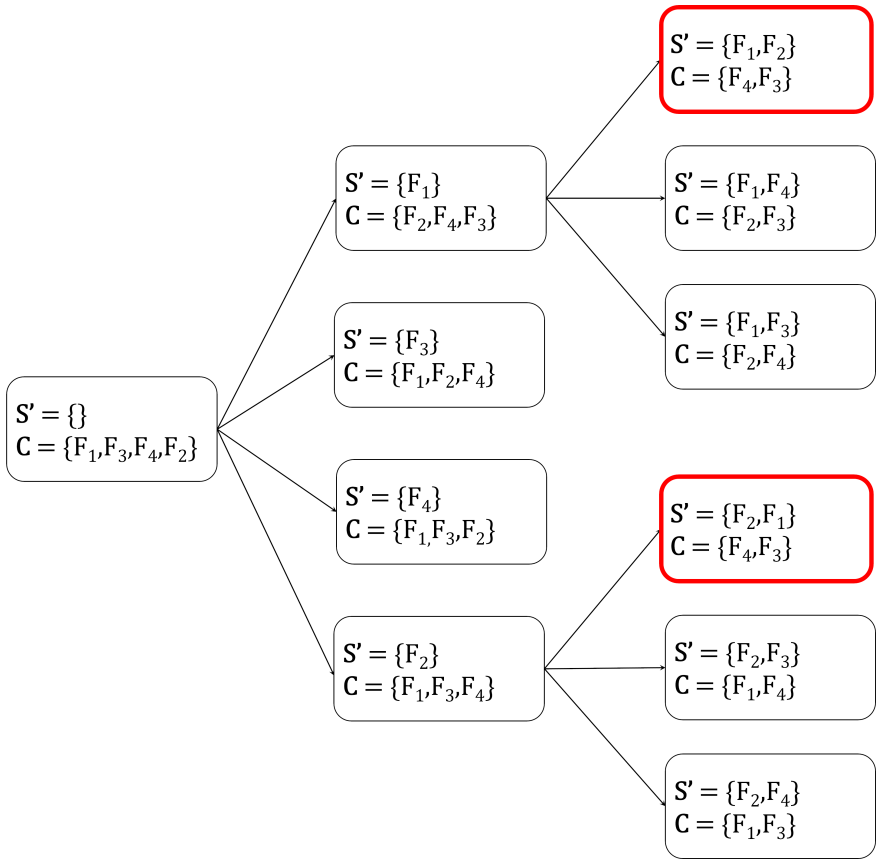


Figure 6.1: An example showing that naive backtracking can explore the same state twice. The set of currently selected variables is denoted as $\mathbf{S}'$, and $\mathbf{C}$ denotes the set of candidate variables returned by ORDERVARIABLES using $\mathbf{S}'$. For simplicity, we consider only 4 variables, assume that ORDERVARIABLES does not remove any variables, and only show part of the search space. We can see that there are two states (highlighted in red) with the exact same set of selected variables.

2^p unique combinations! An example is shown in Figure 6.1. Next, we propose a strategy to avoid such repetition.

6.4.1 A Strategy to Avoid Repeating States

Let $\mathbf{S}'$ be the current set of selected variables, $\mathbf{C} = \{C_1, \ldots, C_k\}$ be the set of candidate variables and $\{t_1, \ldots, t_k\}$ be the corresponding states obtained after selecting one of the variables in $\mathbf{C}$. To avoid repeating states, each variable C_i is excluded from consideration in all subsequent sibling states of t_i (i.e., $t_{i+1}, \ldots, t_k$). Therefore, $t_{i+1}, \ldots, t_k$ will never lead to the same feature sets as t_i, as C_i is in all feature sets explored after t_i but in none of the ones explored by $t_{i+1}, \ldots, t_k$. On a more intuitive level, once C_i

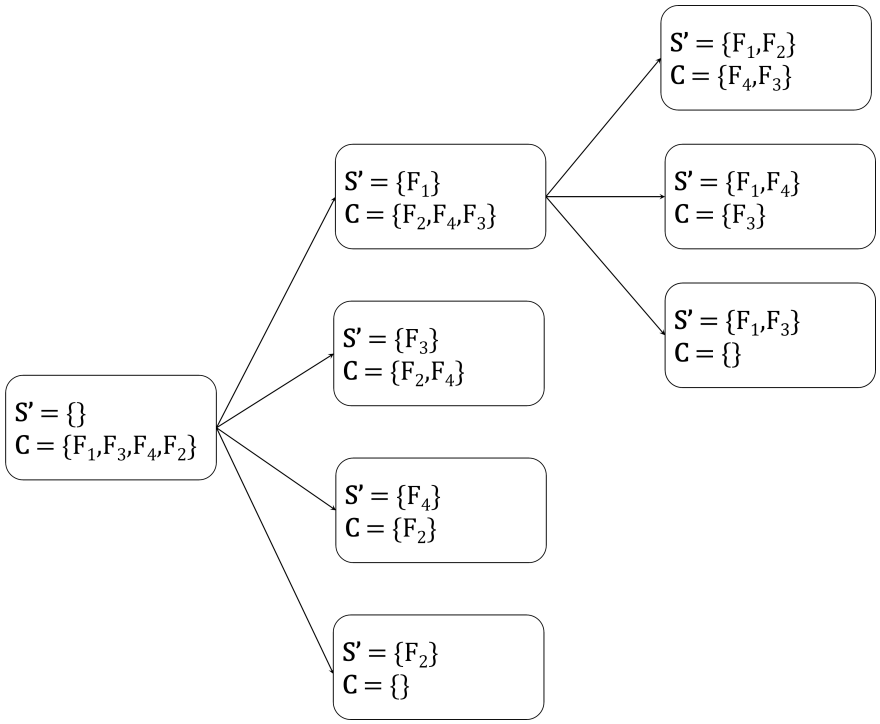


Figure 6.2: An example showing how the proposed strategy can avoid repeating states on the example considered in Figure 6.1. Note that the set of candidate variables **C** of any state does not contain the selected variables of any of its siblings that come before that (i.e., are above it). For example, variable F_1 is selected only once (top state in the middle column), and variable sets containing it are explored only in its children states, but not on any of its siblings. However, variable F_2 which is selected in the top right state is also considered for selection in the bottom state, as they are neither children nor siblings of each other.

is selected and all children states of t_i are fully explored, the algorithm is given the opportunity to consider all feature sets that contain $\mathbf{S}' \cup \{C_i\}$, and thus there is no need to further consider it from that point on. Note that, C_i may still be considered in other tree branches which contain a different set of variables, i.e., where the set of variables is not a subset of $\mathbf{S}'$. Figure 6.2 shows how this strategy can avoid repeating states in the previous example of Figure 6.1.

The above strategy is equivalent to executing the algorithm twice with different input datasets, starting both times with the set of selected variables initialized to $\mathbf{S}'$: once with $\mathcal{D}$ (which contains C_i), and once with the embedded dataset $\mathcal{D}^{C_i}$ (which does not contain C_i). This can be shown by simply noting that, up to that point, the algorithm would select the exact same variables $\mathbf{S}'$ using $\mathcal{D}^{C_i}$, and then would select C_{i+1} instead of C_i, as C_i is not contained in the dataset (unless of course C_i was the last variable, in which case it would terminate). The above observation is summarized

Algorithm 12 TFBS for Multiple Solutions (TMFBS)

Output: Function ORDERVARIABLES, Function BACKWARDPHASE
Input: Dataset $\mathcal{D}$, Current Solution $\mathbf{S}'$ **Output:** Set of Solutions $\mathbf{M}$

```
//Let S be a reference solution returned by TFBS on D using OrderVariables and BackwardPhase
C ← OrderVariables(D, S')
if C = ∅ then //Check if forward phase terminated.
    S' ← BackwardPhase(D, S')
    //Return the candidate solution S' if it is equivalent to the reference solution S
    if Equivalent(S, S') then
        return {S'}
    else
        return ∅
    end if
end if
M ← ∅
for C_i ∈ C do //For each candidate variable in the order given by OrderVariables.
    //Let D^{C_{1:i-1}} be the dataset D without variables C_1, ..., C_{i-1}
    M ← M ∪ TMFBS(D^{C_{1:i-1}}, S' ∪ {C_i}) //Exclude C_{1:i-1} and find all solutions.
end for
return M
```

in the below.

Lemma 1. *Let $\mathcal{D}$ be the input dataset, $\mathbf{S}'$ be the current set of selected variables, $\mathbf{C} = \{C_1, \ldots, C_k\}$ be the current set of candidate variables and $\{t_1, \ldots, t_k\}$ be the corresponding states obtained after selecting one of the variables in $\mathbf{C}$. Excluding C_i from consideration from states $t_{i+1}, \ldots, t_k$ is equivalent to re-running the algorithm on the embedded dataset $\mathcal{D}^{C_i}$.*

6.4.2 The TMFBS Algorithm for Multiple Solutions

We propose the TMFBS algorithm, an extension of TFBS which uses backtracking, as well as the proposed strategy for avoiding repeated states, in order to identify multiple statistically equivalent solutions. For each identified candidate solution, TMFBS performs a test for statistical equivalence of feature sets, to test if it is equivalent with the reference solution $\mathbf{S}$, which is assumed to be known. In practice, $\mathbf{S}$ is initialized to the first solution identified during the search (which coincides with the solution that would be returned by TFBS). The algorithm is shown in Algorithm 12.

Theoretical Properties of TMFBS

We proceed with the theoretical properties of TMFBS. Let $\mathcal{A}$ be an admissible pair of functions $\langle$ORDERVARIABLES, BACKWARDPHASE$\rangle$. We show that TMFBS identifies all

equivalent solutions if the following assumption[3] holds.

Assumption 1. *TFBS instantiated with $\mathcal{A}$ identifies the Markov blanket of T in any dataset $\mathcal{D}$.*

It is important to notice that Assumption 1 refers to *any* dataset $\mathcal{D}$, and therefore implies that TFBS using $\mathcal{A}$ can also find a Markov blanket of T in any embedded dataset in $\mathcal{D}$ (or any dataset which contains $\mathcal{D}$); however, note that a Markov blanket in an embedded dataset is not necessarily a Markov blanket in the original $\mathcal{D}$.

Depending on $\mathcal{A}$, the distributions for which Assumption 1 holds differ. For example, for FBS[4] it has been shown that Assumption 1 holds (assuming an oracle for testing conditional independence) for distributions that satisfy the local composition property [138], i.e., if $T \perp \mathbf{X} \mid \mathbf{Z} \wedge T \perp \mathbf{Y} \mid \mathbf{Z} \Rightarrow T \perp \mathbf{X} \cup \mathbf{Y} \mid \mathbf{Z}$ holds for any sets $\mathbf{X}, \mathbf{Y}$ and $\mathbf{Z}$.

Conditions under which HITON-PC [5] identifies all solutions are given in [138]. We note that, there is no general recipe to identify for which distributions Assumption 1 holds for arbitrary $\mathcal{A}$. However, for algorithms that can be connected to probabilistic graphical models (such as FBS and HITON-PC), one can use the theory of probabilistic graphical models as a guide to find conditions under which it holds (see [138]). We proceed with the main result.

Theorem 3. *TMFBS using $\mathcal{A}$ will identify all and only the solutions equivalent with the reference solution in any dataset $\mathcal{D}$, if (a) $\mathcal{A}$ satisfies Assumption 1 and, (b) it has access to an oracle for deciding equivalence.*

Proof. See Appendix A.5. □

Sound Rules for Pruning the Search Space

Theorem 3 states that TMFBS will find all solutions in any $\mathcal{D}$, and therefore also in any embedded dataset $\mathcal{D}^{\mathbf{E}}$ in $\mathcal{D}$. An immediate consequence of this is that, if no solution is found in some dataset $\mathcal{D}^{\mathbf{E}}$, no solution can be found in any embedded dataset of $\mathcal{D}^{\mathbf{E}\cup\mathbf{E}'}$; if there was one in $\mathcal{D}^{\mathbf{E}\cup\mathbf{E}'}$, it would also be contained $\mathcal{D}^{\mathbf{E}}$ as $\mathcal{D}^{\mathbf{E}\cup\mathbf{E}'}$ is embedded in $\mathcal{D}^{\mathbf{E}}$.

Corollary 3. *Let $\mathcal{D}^{\mathbf{E}}$ and $\mathcal{D}^{\mathbf{E}\cup\mathbf{E}'}$ be two datasets, where the latter is embedded in the former. If no equivalent solution is contained in $\mathcal{D}^{\mathbf{E}}$, then no equivalent solution is contained in $\mathcal{D}^{\mathbf{E}\cup\mathbf{E}'}$.*

Based on this, we propose rules for pruning the search space.

[3] This assumption is basically identical to admissibility rule I made by TIE* (see Figure 7 in [138])

[4] The proof is for IAMB [147], which is FBS with a different, but provably correct, backward phase.

Pruning Rule 1. *If* TMFBS$(\mathcal{D}^{C_{1:i-1}}, \mathbf{S} \cup \{C_i\})$ *does not return any equivalent solution, stop and return* $\mathbf{M}$.

Pruning Rule 2. *Before calling* TMFBS$(\mathcal{D}^{C_{1:i-1}}, \mathbf{S} \cup \{C_i\})$*, check if for some* $\mathcal{D}^{\mathbf{E}}$, $\mathbf{E} \subseteq C_{1:i-1}$ *no equivalent solution was returned, and if so stop and return* $\mathbf{M}$.

In TMFBS, Rule 2 is checked before the recursive call to TMFBS($\mathcal{D}^{C_{1:i-1}}$, $\mathbf{S} \cup \{C_i\}$), while Rule 1 is checked afterwards. We note that Rule 2 is one of the conditions of the IGS procedure used by TIE* (see fourth bullet and step 1 of Figure 9 in [138]).

Recall that, after including a variable C_i in state t_i, none of its subsequent siblings $t_{i+1}, \ldots, t_k$ will consider C_i again, and thus increasingly smaller embedded datasets are explored. Thus, if TMFBS($\mathcal{D}^{C_{1:i-1}}$, $\mathbf{S} \cup \{C_i\}$) does not lead to a solution, by Corollary 3 neither can any call to TMFBS with datasets embedded in $\mathcal{D}^{C_{1:i-1}}$. Although Rule 1 identifies many cases implied by Corollary 3 that can be pruned, it does not necessarily identify all of them. Combining it with Rule 2, which is basically a direct application of Corollary 3, ensures completeness. We note that, theoretically Rule 1 is not required; however, in contrast to Rule 2, which requires to keep track of all embedded datasets that did not lead to any solution, Rule 1 can be implemented trivially and efficiently.

The previous rules only consider the forward phase of TMFBS. We identified another pruning rule, which regards the backward phase of the algorithm.

Pruning Rule 3. *If no solution returned by* TMFBS$(\mathcal{D}^{C_{1:i-1}}, \mathbf{S} \cup \{C_i\})$ *contains* C_i*, stop and return* $\mathbf{M}$.

Rule 3 is checked after the recursive call to TMFBS($\mathcal{D}^{C_{1:i-1}}$, $\mathbf{S} \cup \{C_i\}$).

Pruning Rule 3 regards cases where variable C_i is selected at some step but is not in any solution in $\mathcal{D}^{C_{1:i-1}}$, which can happen if C_i is removed during the backward phase from all of them. Basically, if including C_i lead to some solutions (i.e., Rule 1 does not apply), but none of them actually contains C_i, C_i was a false positive which got removed by the backward phase. Thus, the call TMFBS($\mathcal{D}^{C_{1:i-1}}$, $\mathbf{S} \cup \{C_i\}$) would give the same results as TMFBS($\mathcal{D}^{C_{1:i}}$, $\mathbf{S}'$), which again equals the union of results of the recursive call with all remaining candidate variables $C_{i+1:k}$. Because of that, there is no reason to consider the remaining candidates $C_{i+1:k}$, which have already been implicitly considered, and $\mathbf{M}$ can be returned.

Despite all attempts to speed-up TMFBS, the number of candidate solutions may still be exponential in the number of variables, and thus TMFBS may not terminate in a reasonable time frame. This is not a weakness of TMFBS, but an inherent property of the problem. Thus, to avoid such cases in practice, we recommend setting a limit on the number of candidate states to consider, the number of solutions to return, or a combination of both. This problem also motivated us to develop an algorithm for summarizing and visualizing multiple solutions, presented in Section 6.5.

Algorithm 13 TIE* (Figure 6 in [138]

Output: Function ORDERVARIABLES, Function BACKWARDPHASE
Input: Dataset $\mathcal{D}$, Target T, Markov blanket induction algorithm $\mathcal{X}$, Procedure to generate embedded datasets $\mathcal{Y}$, Criterion $\mathcal{Z}$ to verify Markov blankets for T
Use $\mathcal{X}$ to find a Markov blanket $\mathbf{S}$ in $\mathcal{D}$
repeat
Use $\mathcal{Y}$ to generate dataset $\mathcal{D}^{\mathbf{E}}$ by removing a subset of variables $\mathbf{E}$ from the full set of variables $\mathbf{F}$
Use $\mathcal{X}$ to find a Markov blanket $\mathbf{S}_{new}$ from $\mathcal{D}^{\mathbf{E}}$
Use $\mathcal{Z}$ to determine if $\mathbf{S}_{new}$ is a Markov blanket in the original distribution
until All datasets $\mathcal{D}^{\mathbf{E}}$ generated by $\mathcal{Y}$ have been considered
return All identified solutions equivalent to $\mathbf{S}$ according to $\mathcal{Z}$

function IGS(instantiation of $\mathcal{Y}$ for TIE* (Figure 9 in [138]))
Inputs"
1) Markov blankets $\mathbf{M} = \{\mathbf{S}_1, \ldots, \mathbf{S}_n\}$ obtained so far by TIE* and ordered by the time of discovery from earliest ($\mathbf{S}_1$) to latest ($\mathbf{S}_n$)
2) Subsets $\mathbf{E}_1, \ldots, \mathbf{E}_n$ that were used in previous calls to IGS to generate embedded datasets that led to the discovery of the above Markov blankets ($\mathbf{E}_1 = \emptyset$)
3) Subsets $\mathbf{E}^*_1, \ldots, \mathbf{E}^*_m$ that were used in previous calls to IGS to generate embedded datasets that did not lead to Markov blankets

Generate the smallest subset of variables $\mathbf{E}$: $\mathbf{E}_i \subset \mathbf{G} \subseteq (\mathbf{S}_i \cup \mathbf{G}_i)$ for some $i = 1, \ldots, n$ that neither includes $\mathbf{G}^*_j$ nor coincides with $\mathbf{G}_k$ for any $j = 1, \ldots, m$ and $k = 1, \ldots, n$
Return embedded dataset $\mathcal{D}^{\mathbf{E}}$
end function

6.4.3 Relation to the TIE* Algorithm

TIE* [138] is a general method, that can use any feature selection algorithm as a black-box for identifying multiple solutions. It uses three components: (a) a Markov blanket discovery algorithm $\mathcal{X}$, (b) a function $\mathcal{Y}$ that generates embedded datasets, and (c) a criterion $\mathcal{Z}$ that tests if a solution is a Markov blanket in the original distribution. The TIE* algorithm, along with the IGS method for creating embedded datasets (slightly modified to use similar notation as TMFBS) are shown in Algorithm 13. As a criterion $\mathcal{Z}$, any test for equivalence of feature sets can be used.

The main idea of TIE* is to (a) identify a solution $\mathbf{S}$ on the original dataset $\mathcal{D}$, (b) remove variables of $\mathbf{S}$ from $\mathcal{D}$ and run $\mathcal{X}$ on the embedded datasets to identify additional solutions, and (c) repeat (a,b) for each new solution found, until no more solutions can be found. All of this is done without running $\mathcal{X}$ multiple times on the same dataset, and without running $\mathcal{X}$ on a dataset if no solution was found in a superset of it (same as Pruning Rule 2). It has the same theoretical properties as

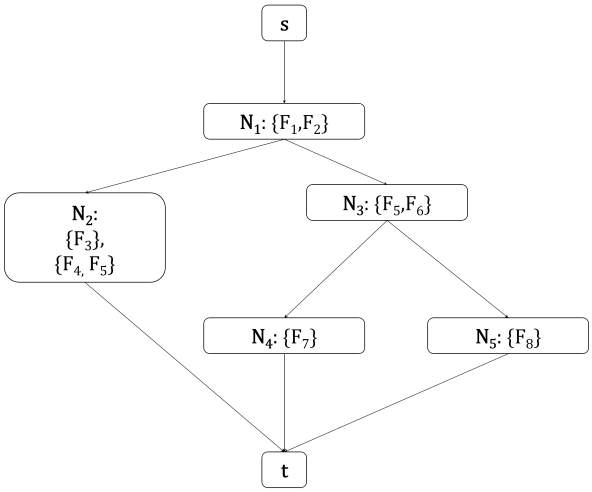


Figure 6.3: An example of a multiple solution graph.

TMFBS, that is, it will identify all equivalent solutions if Assumption 1 holds and it has access to an oracle for deciding equivalence. In practice, the results of TIE* and TMFBS may differ due to statistical errors, although, as we will see in the experiments, they return very similar sets of solutions.

Although more general than TMFBS, TIE* has two main drawbacks: (a) it treats $\mathcal{X}$ as a black-box and thus does not take advantage of its search strategy, leading to unnecessary repetition of computations, and (b) no details are provided for an efficient implementation, which may make it hard to use or compare against (for instance, for our implementation we had to come up with an efficient method to incrementally explore embedded datasets).

6.5 Summarizing and Visualizing Multiple Solutions

The more solutions are identified, the harder it is to interpret them. Typically, most solutions have some overlap (features that are indispensable) and only differ for a few features (replaceable features), enabling a more compact representation. We propose a data structure for compactly representing feature sets, as well as an algorithm to construct it next.

6.5.1 Multiple Solution Graphs

Let $\mathbf{M}$ denote a set of solutions $\mathbf{M} = \{\mathbf{S_1}, \ldots, \mathbf{S_k}\}$. We propose to represent them with a **multiple solution graph** (MSG), which is a directed acyclic graph $\mathcal{G}$ with the following properties: (a) $\mathcal{G}$ contains exactly one root and leaf node, called s and t, (b) each other node in $\mathcal{G}$ is associated with one or more sets of features, and has in and out degree

Algorithm 14 Feature Set Compression

Input: Solutions $\mathbf{M}$
Output: MSG $\mathcal{G}$ representing $\mathbf{M}$

1: $\mathcal{G} \leftarrow \{s, t\}$
2: **for** each $\mathbf{S_i} \in \mathbf{M}$ **do**
3: $\quad \mathcal{G} \leftarrow \mathcal{G} \cup \mathbf{S_i}$
4: $\quad \mathcal{G} \leftarrow \mathcal{G} \cup (s \rightarrow \mathbf{S_i})$
5: $\quad \mathcal{G} \leftarrow \mathcal{G} \cup (\mathbf{S_i} \rightarrow t)$
6: **end for**
7: $\mathcal{G} \leftarrow$ FORWARDCOMPRESSION$(\mathcal{G}, s)$
8: $\mathcal{G} \leftarrow$ BACKWARDCOMPRESSION$(\mathcal{G}, t)$
9: $\mathcal{G} \leftarrow$ ORCOMPRESSION$(\mathcal{G}, t)$
10: **return** $\mathcal{G}$

at least one (c) each directed path p from s to t represents a solution, which can be obtained by choosing one of the feature sets of each node on p and taking the union of the chosen feature sets, and (d) $\mathcal{G}$ does not encode any additional solutions.

An example of an MSG is shown in Figure 6.3. All nodes contain a single set of features, except for N_2 which contains two. The solutions this MSG represents are $\{F_1, F_2, F_3\}$ $(s \rightarrow N_1 \rightarrow N_2 \rightarrow t)$, $\{F_1, F_2, F_4, F_5\}$ $(s \rightarrow N_1 \rightarrow N_2 \rightarrow t)$, $\{F_1, F_2, F_5, F_6, F_7\}$ $(s \rightarrow N_1 \rightarrow N_3 \rightarrow N_4 \rightarrow t)$ and $\{F_1, F_2, F_5, F_6, F_8\}$ $(s \rightarrow N_1 \rightarrow N_3 \rightarrow N_5 \rightarrow t)$. There are two paths through N_2, one for each feature set. Nodes with multiple features sets represent OR relations: along paths through such nodes, all solutions contain the same features, but only contain one of the sets of that node. Note that, feature sets in such nodes are equivalent in the context of the remaining features on that path.

Hereafter, we will use the names N_i to refer to the i-th node in $\mathcal{G}$ and $var[N_i]$ to refer to the sets of features associated with N_i. In case $var[N_i]$ contains only a single set of features, it will directly refer to that set. If it is clear from the context, we will refer to a node with its associated feature set. We will use $parents[N_i]$ and $children[N_i]$ to refer to the parents and children of N_i in $\mathcal{G}$ respectively.

6.5.2 An Algorithm for Constructing Multiple Solutions Graphs

We propose a greedy algorithm to construct an MSG $\mathcal{G}$ to compactly represent a set of solutions $\mathbf{M} = \{\mathbf{S_1} \ldots, \mathbf{S_k}\}$ (see Algorithm 14). First, k nodes are created, one for each feature set in $\mathbf{M}$. Then, edges from s into each of those nodes, as well as edges into t out of them are included in $\mathcal{G}$. It is easy to see that $\mathcal{G}$ exactly represents $\mathbf{M}$, as it contains k paths from s to t, one for each feature set. Afterwards, forward and backward compression steps are performed to reduce the size of $\mathcal{G}$. Both are performing operations on $\mathcal{G}$ with the goal of simplifying it. Until that step, all nodes

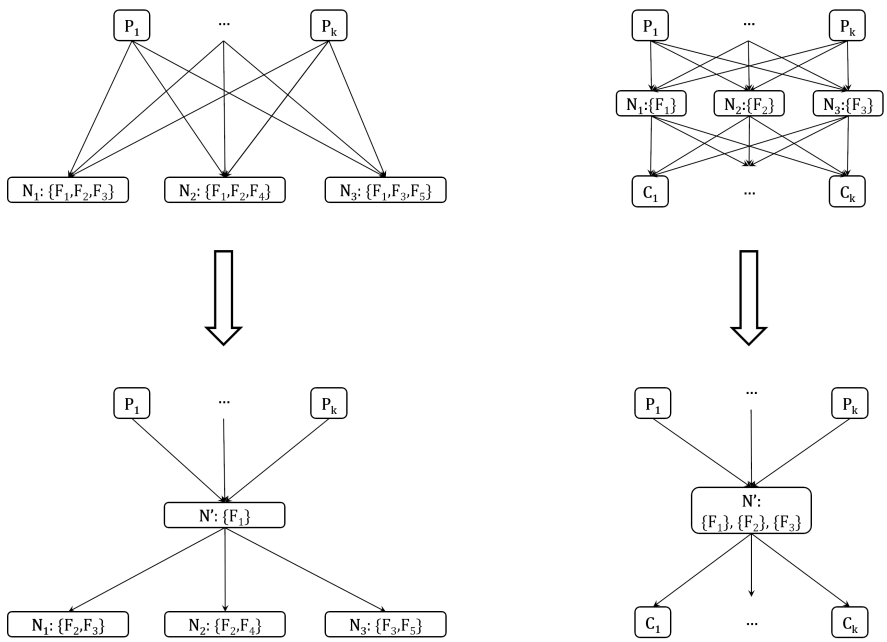


Figure 6.4: Examples of the forward merging (left) and OR merging (right) operations.

contain a single feature set. The final step is to merge nodes, creating nodes that contain multiple feature sets. Before describing everything in detail, we proceed by presenting the operations used, along with proofs of correctness.

6.5.3 Compression Operations

Operation 1 (Forward Merging)**.** *Let* $\mathbf{N} = N_1, \ldots, N_n$ *be a set of nodes. If* $\mathbf{F}' = \bigcap_{N_i} var[N_i] \neq \emptyset$ *and all of them have exactly the same set of parents* $\mathbf{P}$*, then, a new node* N' *is created with* $var[N'] = \mathbf{F}'$*,* $parents[N'] = \mathbf{P}$*,* $children[N'] = \mathbf{N}$*, and remove all incoming edges from* $\mathbf{N}$*, as well as all features* $\mathbf{F}'$ *from* $\mathbf{N}$*.*

Operation 2 (Backward Merging)**.** *Let* $\mathbf{N} = N_1, \ldots, N_n$ *be a set of nodes. If* $\mathbf{F}' = \bigcap_{N_i} var[N_i] \neq \emptyset$ *and all of them have exactly the same set of children* $\mathbf{C}$*, then, a new node* N' *is created with* $var[N'] = \mathbf{F}'$*,* $parents[N'] = \mathbf{N}$*,* $children[N'] = \mathbf{C}$*, and remove all outgoing edges from* $\mathbf{N}$*, as well as all features* $\mathbf{F}'$ *from* $\mathbf{N}$*.*

Operation 3 (OR Merging)**.** *Let* $\mathbf{N} = N_1, \ldots, N_n$ *be a set of nodes. If all nodes have the same sets of parents* $\mathbf{P}$ *and children* $\mathbf{C}$*, then, a new node* N' *is created with* $var[N'] = \{var[N_1], \ldots, var[N_n]\}$*, add edges from* $\mathbf{P}$ *to* N' *as well as edges from* N' *to* $\mathbf{C}$*, and remove all nodes* $\mathbf{N}$ *from* $\mathcal{G}$*.*

Figure 6.4 shows examples of the forward merging and OR merging operations; the backward merging operation is not shown, as it is symmetric to the forward merging

operation, requiring common children instead of common parents. In the example of the forward merging operation, the nodes N_1, N_2 and N_3 share the same parents and have the feature F_1 in common. The resulting node N' contains F_1, which is removed from the feature sets of the other nodes. The OR merging operation requires that the nodes share the same parents and children. If so, the nodes can be merged into a single node, containing as feature sets all features sets of the merged nodes, as shown in the example. As can be seen in the previous examples, all operations simplify the graph by removing nodes, features and edges. Next we will quantify those effects for all operations.

We start with the forward and backward merging operations. Without loss of generality, we consider the forward merging operation; the same also holds for backward merging. Before its application on node set $\mathbf{N}$ and their parents $\mathbf{P}$, the graph contains $|\mathbf{N}| \cdot |\mathbf{P}|$ edges between them. The resulting graph contains $|\mathbf{P}|$ edges from $\mathbf{P}$ to N', and $|\mathbf{N}|$ edges from N' to $\mathbf{N}$, much fewer than the ones in the initial graph. In addition, the total number of features contained in the graph is reduced by $(|\mathbf{N}| - 1) \cdot |\mathbf{F}'|$. Note that application of those operations may result in a node containing no features. In this case the node can be removed, and edges from all its parents to all its children have to be added. Finally, OR merging always decreases the number of nodes and edges, while maintaining the total number of features. Specifically, for node set $\mathbf{N}$, the number of nodes is reduced by $|\mathbf{N} - 1|$, and the edges are reduced by $(|\mathbf{P}| - 1)|\mathbf{N}|) + (|\mathbf{C}| - 1)|\mathbf{N}|)$, where $\mathbf{P}$ and $\mathbf{C}$ are the sets of parents and children of $\mathbf{N}$ respectively. Next, we will show that all operations are preserve the number of represented solutions.

Theorem 4. *Application of Operation 1 and 2 does not affect the set of represented solutions by $\mathcal{G}$*

Proof. See Appendix A.6. □

A proof sketch for the OR merging operation follows. Observe that OR merging basically only groups some nodes together into one "super-node". Based on this observation, it can easily be shown that it does not alter the solutions represented by the graph, as nodes are merged if and only if they have the same parents and children. Furthermore, this operation can be applied locally and independently to any part of the graph in any order, without affecting the final outcome.

Next, we will present algorithms that perform the forward and backward compression steps, using the respective merging operations. We do not provide any algorithm for the OR compression, as it simply is repeated application of OR merging until no more nodes can be merged.

Algorithm 15 Forward Compression

Input: Graph $\mathcal{G}$, Node N
Output: Graph $\mathcal{G}$

1: **Groups** $\leftarrow$ SplitChildrenByFeatures($\mathcal{G}, N$)
2: **for** each $G_i \in$ **Groups do**
3: $\quad \mathbf{F}' \leftarrow \bigcap_{N_j \in G_i} var[N_j]$
4: $\quad var[N'] \leftarrow \mathbf{F}'$ //*Create node N' with features* $\mathbf{F}'$
5: $\quad \mathcal{G} \leftarrow \mathcal{G} \cup N'$ //*Add N' to $\mathcal{G}$*
6: $\quad \forall_{N_j \in G_i} var[N_j] \leftarrow var[N_j] \setminus \mathbf{F}'$ //*Remove common features from G_i*
7: $\quad \mathcal{G} \leftarrow \mathcal{G} \cup (N \rightarrow N')$ //*Add edge from N to N'*
8: $\quad \forall_{N_j \in G_i} \mathcal{G} \leftarrow \mathcal{G} \cup (N' \rightarrow N_j)$ //*Set N' as parent of each N_j*
9: $\quad \forall_{N_j \in G_i} \mathcal{G} \leftarrow \mathcal{G} \setminus (N \rightarrow N_j)$ //*Remove N as parent from N_j*
10: $\quad \mathcal{G} \leftarrow$ ForwardCompression($\mathcal{G}, N'$)
11: **end for**
12: **return** $\mathcal{G}$

6.5.4 Algorithms for Forward and Backward Compression

The forward compression starts from the root node s and separates its children into groups as follows. First it identifies the feature with the most occurrences among all its children and groups together all children that contain that feature (function SplitChildrenByFeatures in Algorithm 15). This is repeated for all children that have not been grouped yet, until none remains. Next, after all children have been grouped, Operation 1 is performed on each such group G_i (lines 3–9 in Algorithm 15). Application of Operation 1 is possible since all nodes in each group share common features and because all of them share the same parents, by construction. The aforementioned steps are repeated recursively for each newly created node N' in place of s, until the leaf node t is reached. The procedure is summarized in Algorithm 15.

After completing the forward compression, an additional step is employed to further reduce the size of the DAG. This step is very similar to the forward compression, with a small modification. The backward compression starts from the leaf node t and groups all parents of t such that all nodes in the same group have the same children (function SplitParentsByChildren in Algorithm 16). This is necessary in order to perform the backward operation. It was not required for the forward compression, as there it was guaranteed by construction that all nodes always have the same parents. Here however, it may happen that some parent of t is also a parent of some other parent of t, complicating things. Next, each such group is further split into sets of nodes that have common features, similarly to the forward step (function SplitsGroupsByFeatures in Algorithm 16). Thus, after both splitting steps, nodes in each group have the same children and share features, allowing application of Operation 2 (lines 4–10 in

Algorithm 16 Backward Compression

Input: Graph $\mathcal{G}$, Node N
Output: Graph $\mathcal{G}$

1: **Groups** $\leftarrow$ SPLITPARENTSBYCHILDREN($\mathcal{G}, N$)
2: **Groups** $\leftarrow$ SPLITGROUPSBYFEATURES(**Groups**)
3: **for** each $G_i \in$ **Groups** **do**
4: $\quad \mathbf{F}' \leftarrow \bigcap_{N_j \in G_i} var[N_j]$
5: $\quad var[N'] \leftarrow \mathbf{F}'$ *//Create node N' with features $\mathbf{F}'$*
6: $\quad \mathcal{G} \leftarrow \mathcal{G} \cup N'$ *//Add N' to $\mathcal{G}$*
7: $\quad \forall_{N_j \in G_i} var[N_j] \leftarrow var[N_j] \setminus \mathbf{F}'$ *//Remove common features from G_i*
8: $\quad \mathcal{G} \leftarrow \mathcal{G} \cup (N' \rightarrow N)$ *//Add edge from N' to N*
9: $\quad \forall_{N_j \in G_i} \mathcal{G} \leftarrow \mathcal{G} \cup (N_j \rightarrow N')$ *//Set N' as child of each N_j*
10: $\quad \forall_{N_j \in G_i} \mathcal{G} \leftarrow \mathcal{G} \setminus (N_j \rightarrow N)$ *//Remove N as child from N_j*
11: **end for**
12: *//Iterate over all parents. Additional parents may be created after BackwardCompression and must also be considered.*
13: **for** each $N_i \in Parents(N)$ **do**
14: $\quad \mathcal{G} \leftarrow$ BACKWARDCOMPRESSION($\mathcal{G}, N_i$)
15: **end for**
16: **return** $\mathcal{G}$

Algorithm 16). Again, this procedure is applied recursively for all parents of t, *including the ones that are created after a recursive call of backward compression*[5], until the root node s is reached.

6.5.5 Related Methods

The problem of compactly representing feature sets is closely related to several other problems that have appeared in the computer science literature, which we briefly summarize and compare below.

Binary decision diagrams (BDDs) [6,23] are directed acyclic graphs that are used to compactly represent a Boolean function. Each node is associated with a Boolean feature and has two outgoing edges, one labeled "0" (or false) and one labeled "1" (or true), corresponding to the respective assignment of x. It has one root node, which is one of the Boolean features, and two leaf nodes "0" and "1". Each path from the root node to one of the leaf nodes represents a feature assignment for the represented Boolean function. Depending on the leaf node, this assignment evaluates the represented Boolean function to true or false. **Ordered binary decision diagrams** (OBDDs) [23] are a special type of BDDs. They have the property that there is a

[5]In an actualy implementation of Algorithm 16, the set of parents of N at line 13 has to be updated after each recursive call in the loop, as it may change during the loop.

unique structure for a given feature ordering, which is not necessarily the case for BDDs. The size of the OBDD highly depends on the feature ordering. The problem of finding the minimal OBDD is NP-complete [12] [6]. There are various heuristics to find a good feature ordering; see [124] for a survey on such methods. Another interesting type of BDDs are **zero-suppressed binary decision diagrams** (ZDDs) [107]. Often, especially when there are only a few solutions for a Binary function, ZDDs can be much smaller than OBDDs. It is straightforward to use BDDs to represent feature sets (which are a set of sets). Feature sets can be represented with Boolean functions by converting each set to an AND function and use an OR function between all such sets. For example, if $\mathbf{M} = \{\{F_1, F_2\}, \{F_2, F_3, F_4\}\}$, the Boolean function $(F_1 \wedge F_2) \vee (F_2 \wedge F_3 \wedge F_4)$ represents all solutions in $\mathbf{M}$. In fact, OBDDs and ZDDs have already been used in this context [108]. The reason we chose not to use BDDs for our case is that they aren't as easy to interpret, and it is harder to identify represented solutions visually. A path may contain "0" edges, which have to be filtered out in order to retrieve the respective feature set.

Acyclic deterministic finite-state automata (ADFA) (also known as directed acyclic word graphs (DAWG)) [36,73,123] are used to represent a set of strings (called lexicon) in a compact way. ADFAs are directed acyclic graphs with nodes representing states and edges representing transitions between them. They contain one root and one leaf node, and each edge is associated with a letter. Each directed path from the root node to the leaf node represents a string, by concatenating the letters associated with each edge on that path. There are fast algorithms to incrementally construct a minimal size ADFA [36], or to minimize a given ADFA [123]; see [35] for a review and comparison of such methods. In our case, ADFAs could be used by converting each feature set to a string, and then using them to encode the whole set of feature sets. One way to do this is to choose a feature ordering, and to convert feature sets to strings by sorting them according to that ordering. This however is sub-optimal, as it unnecessarily restricts the resulting DAG to some feature ordering, which is not needed to actually represent feature sets. On the other hand, ADFAs also allow the repetition of letters, which is not needed in our problem as we deal with sets. Both of those reasons may potentially reduce their efficiency for compactly representing feature sets, which is why we decided to not use them.

We did not further investigate the possibility of using one of those data structures for our problem. We note that, due to their similarity with our proposed data structure, it may be that techniques used for minimization of BDDs or ADFAs could be applied in our case.

[6] Specifically, the decision version of the problem is NP-complete

Table 6.1: Summary of the datasets used for the experimental evaluation. We used 6 regression datasets and 5 binary classification datasets, with number of variables ranging from 46 to 970, and samples sizes between 1994 and 60021.

	Dataset	#Samples	#Variables
Regression	CnC Non-violent	2118	102
	CnC Violent	1994	102
	BlogData	60021	276
	CT Slice	53500	379
	UJI Latitude	21048	520
	UJI Longitude	21048	520
Classification	Ada	4562	46
	Musk	6598	166
	Sylva	14394	213
	Madelon	2600	500
	Gina	3468	970

6.6 Experimental Evaluation

We evaluated TMFBS and compared it to the TIE* algorithm [138]. In our first experiment we compared (a) the number of solutions returned by each algorithm, as well as how many of them are statistically equivalent, (b) the predictive performance of the returned solutions, and (c) how the algorithms compare in terms of computational performance. Then, we investigated how the number of solutions and running time of both algorithms is affected by sample size. Finally, we show some examples of multiple solution graphs obtained on solutions returned by TMFBS.

Data. We considered binary classification and regression datasets. The data were collected from the UCI ML repository [41], using the following criteria: (a) they contain at least 1000 samples, to ensure that the equivalence tests have sufficient power, and (b) they contain at most 1000 features, so that all algorithms can terminate in a reasonable time frame. The datasets are shown in Table 6.1. More details about the data collection and pre-processing are given in Appendix C.3.

Feature Selection Algorithms and Hyper-parameters. For a fair comparison, we instantiated both TIE* and TMFBS with the FBS algorithm, as presented in Section 6.3. For continuous outcomes, we used the partial correlation test, and for binary outcomes we used a likelihood-ratio test based on logistic regression (see Section 2.4.1). For the significance level α of the conditional independence tests we considered 100 values uniformly spaced in the exponent of $10^{[-8,\ldots,\log_{10}(0.05)]}$, (i.e., the minimum α is 10^{-8} and the maximum is 0.05). A wide range of values for α is considered to allow for better tuning of FBS (see Section 6.2.3 for the motivation behind this).

Predictive Modeling. As predictive algorithms, we used ridge logistic and linear regression for binary classification and regression outcomes respectively[7]. For the regularization parameter λ of ridge regression we considered values $2^{[-30,\ldots,30]}$, with a step size of 0.5 on the exponent (a total of 121 values).

Equivalence Test for Solutions. The goal of the experiments is to identify multiple, statistically equivalent Markov blankets. Following the recommendations given in Section 6.2.3, we combine a PEQ test with an IEQ test. Specifically, we (a) use a permutation-based variant of the variance test [154] for PEQ with 1000 permutations (see Section 6.2.3 for details), and (b) an IEQ test based on likelihood-ratio tests using logistic and linear regression models for classification and regression outcomes respectively. As sample sizes are relatively large, we set the significance level to 0.05 for both tests to minimize the number of false solutions.

Analysis Protocol. We employed a train/validation/test protocol, splitting the data to 60%/20%/20% respectively. As performance metrics we used the out-of-sample R^2 for regression and the area under the ROC curve for classification. We used the following procedure: (a) on the training set, we trained a ridge regression model using the features identified by FBS for each combination of λ and α (a total of $121 \cdot 100 = 12100$ combinations), (b) we selected the best combination based on its performance on the validation set, (c) we executed TMFBS and TIE* on the combined training and validation set using the best α and trained one model for each solution using the best λ, and (d) estimated their predictive performance on the test set.

Implementations. All algorithms were implemented by us in Matlab, except for ridge logistic regression, for which we used the implementation provided by the LIBLINEAR package [48].

6.6.1 Evaluation of TMFBS and Comparison with TIE*

For the first experiment we employed the aforementioned analysis protocol on the datasets shown in Table 6.1 datasets. In order to measure the speed-up of TMFBS over TIE*, we used the number of independence tests performed by each algorithm as a proxy of running time[8]. For each solution, we compute the predictive performance obtained on the test set, as explained previously. Ideally, all identified equivalent solutions should have similar predictive performance. In order to verify that, we performed a test of performance equivalence for each identified solution with the

[7]We chose those models to compare equivalent solutions on an equal footing, as the conditional independence tests used by FBS only identify linear (or monotonic) dependencies. Using non-linear predictive models might obfuscate the results, as they would favor solutions which happen to identify variables that are non-linearly related to the outcome.

[8]The number of tests is used instead of running time, as this is independent of the implementations used for the algorithms and tests. This is common practice when comparing algorithms based on independence tests (see [4] for example)

Table 6.2: The table shows the summary of the comparison. It shows the speed-up of TMFBS over TIE*, the performance of the reference solution as well as the range of performances over all returned solutions, the number of additional solutions returned by each algorithm (#Sol.) (i.e., without counting the reference) and how many of them are statistically equivalent with the reference on the test set (#Eq.). Both algorithms produce very similar results in terms of the number of solutions and equivalent solutions identified, and all identified solutions have similar predictive performance. In terms of number of statistical tests performed, TMFBS performs around 2 times fewer tests than TIE*.

				Performance Range		#Sol. (#Eq.)	
	Dataset	Speed-up	Performance	TMFBS	TIE*	TMFBS	TIE*
Regression	CnC Non-violent	2.18	0.585	-	-	-	-
	CnC Violent	2.31	0.588	[0.583, 0.590]	[0.583, 0.590]	2 (2)	2 (2)
	BlogData	1.91	0.304	-	-	-	-
	CT Slice	2.29	0.834	[0.834, 0.834]	[0.834, 0.834]	19 (16)	19 (16)
	UJI Latitude	2.20	0.911	-	-	-	-
	UJI Longitude	2.40	0.938	[0.938, 0.938]	[0.938, 0.938]	12 (9)	13 (10)
Classification	Ada	1.97	0.904	-	-	-	-
	Musk	2.24	0.991	-	-	-	-
	Sylva	1.66	0.999	-	-	-	-
	Madelon	1.81	0.646	-	-	-	-
	Gina	2.09	0.932	[0.932, 0.934]	[0.932, 0.934]	1 (1)	1 (1)

reference solution on the test set. As a performance equivalence test we employed the permutation-based variance test, as described above. As the significance level is set to 0.05, we expect around 5% of equivalent solutions to be rejected on average. The results are summarized in Table 6.2.

First of all, we notice that both algorithms return a similar number of solutions. In fact, the solutions are identical, except for the UJI Longitude dataset, where TMFBS returned 12 solutions while TIE* returns 13. Those results agree with what we would expect from theory, as both algorithms have the same theoretical guarantees, although the results might differ in practice (see Section 6.4.3).

In terms of total number of returned solutions, we see that in most cases the algorithms identify only a single solution (7 out of 11 datasets), while in the rest the number of additional solutions is at most 19. Most of them are statistically equivalent, and even the ones that are not have very similar predictive performance (the difference is less than 0.1% in all cases). Even though the number of solutions is low, this is important *evidence that multiple solutions indeed exist in practice*. It is unlikely that those are false positives, given that the analysis is designed so that the number of false positive equivalences is minimized: we used large sample sizes, a relatively high

threshold for the equivalence tests, filtered out solutions using a PEQ test, and used extensive tuning of the algorithms (see Section 6.2.3 for explanations of how the above affect the number of solutions). Furthermore, we there is no reason to believe a priori that the selected datasets do contain equivalent solutions, as the criteria used for selecting them are based on their size.

Finally, regarding speed-up, in all cases TMFBS is around 1.5-2.5 times faster than TIE*, showing that TMFBS is able to successfully take advantage of the search structure of FBS; larger speed-ups are expected with increasing number of features and solutions (see also the results of the next experiment, where speed-ups of 1-2 orders of magnitude are the norm).

6.6.2 Number of Solutions and Speed-up with Increasing Sample Size

Next, we performed an experiment to investigate how the number of solutions is affected by lower sample sizes, where more false positive solutions are expected due to lower power of the equivalence tests. Furthermore, we also check how the increased number of solutions affects the speed-up of TMFBS over TIE*. For this experiment, we only used the regression datasets, as the experiment is too time consuming for the classification datasets. The reason is that the logistic regression based test are significantly more computationally expensive than partial correlation tests, making such a large experiment infeasible.

We used the same experimental setup as before, but instead of using the full training set (i.e., the 60% of the original samples), we sampled $10\%, 20\%, \ldots, 90\%$ of the training data and used that as a training set to tune the hyper-parameters of FBS. The sampling was performed 20 times for each value, i.e., we performed a total of $20 \cdot 9 = 180$ runs of the analysis protocol for each dataset, and we report averages over the 20 runs. A limit of 1000 solutions was set, as in some small sample cases the TIE* algorithm would not terminate otherwise (the number of solutions often ranged in the millions).

Figure 6.5 shows how the number of solutions identified by TMFBS and TIE* varies with sample size. As before, the results are very similar for both algorithms. First we notice that, as expected, *the number of solutions tends to decrease with increasing sample size*. The only exception is for the CnC Non-violent dataset, where the number of solutions increases a bit for 70% of the samples or higher. Furthermore, for some cases (e.g., 40% and 70% for the UJI Longitude dataset) the number of solutions increases temporarily, and decreases afterwards. We were not able to identify the cause of this, but believe it may be an artifact of the experimental setup. Specifically, we believe it is due a combination of the relatively small number of runs (we used only 20 repetitions, due to the large computational cost) and the limit of 1000 solutions (again, to reduce

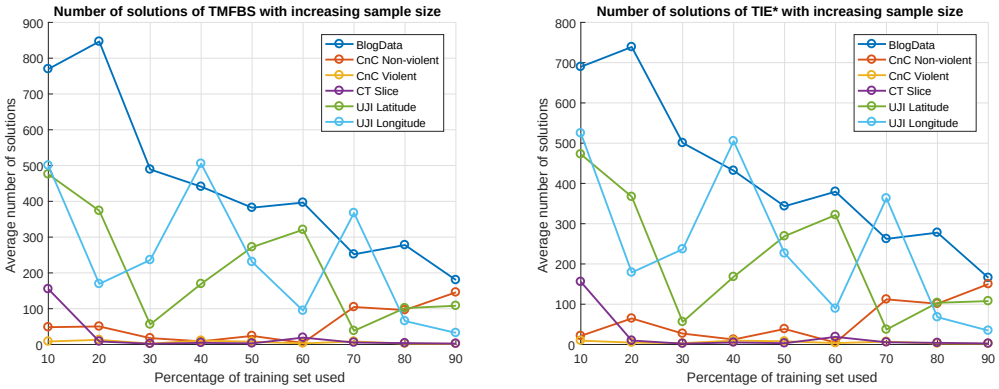


Figure 6.5: The figures show the average number of solutions over 20 runs with increasing sample size for TMFBS (left) and TIE* (right). In both cases, we can see that the number of solutions tends to decrease with increasing sample size.

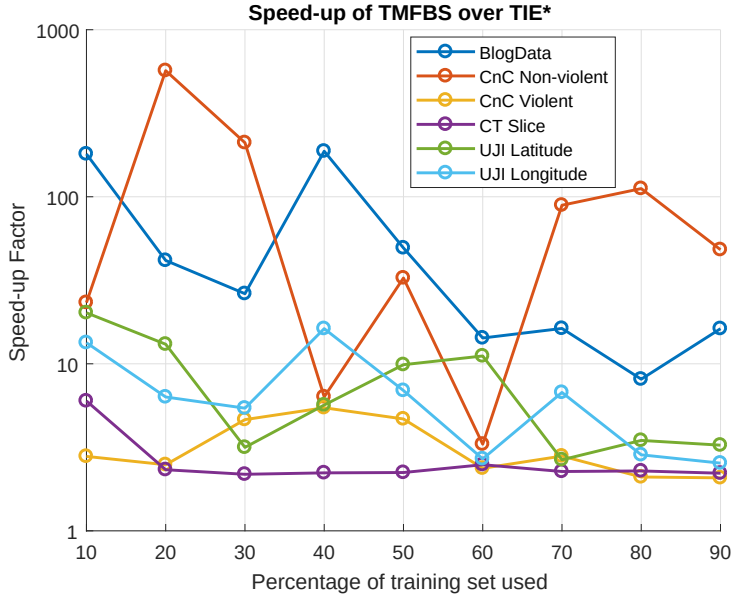


Figure 6.6: The figure shows the speed-up of TMFBS over TIE* with increasing sample size. We can see that TMFBS is typically 1-2 orders of magnitude faster than TMFBS on average.

the total computational cost). In any case, even though the number of solutions is not strictly monotonically decreasing with sample size, overall there is a clear monotonic trend for most datasets.

Figure 6.6 shows the speed-up of TMFBS over TIE*, computed as the ratio of statistical tests performed by TIE* over TMFBS, and averaged over all runs. We see that, on average, TMFBS significantly outperforms TIE*, typically being 1-2 orders of magnitude faster. The largest speed-ups are observed for lower sample sizes, where

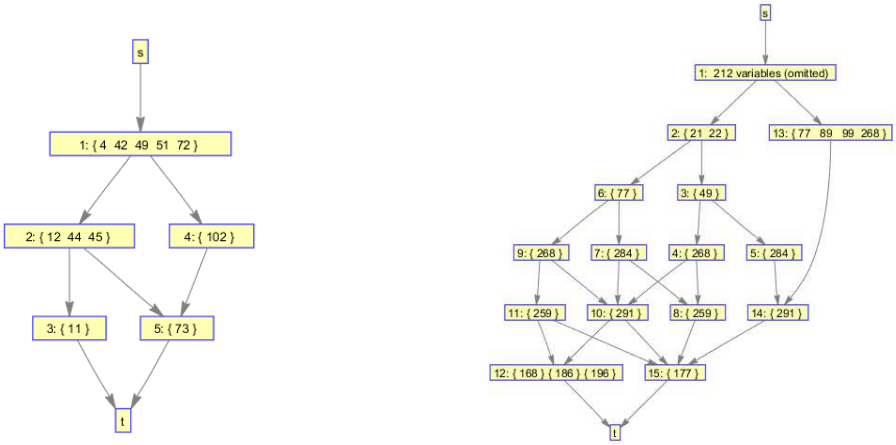


Figure 6.7: Multiple solutions graphs for the solutions on the CnC violent (left) and CT slice (right) datasets. The graphs contain 3 and 20 solutions, and require only 5 and 15 nodes respectively to represent them (excluding s and t). The first node contains 5 and 212 variables respectively, which correspond to variables that are contained in all solutions, i.e., variables that are indispensable.

the number of identified solutions is also larger. Thus, *the more solutions are identified, the larger the speed-up of TMFBS over TIE* is*. Recall however that we set an upper limit of 1000 solutions for each run and, given that TIE* requires more tests per solution, we expect the speed-up to be even larger if no limit is enforced. Those results were expected, and can be explained by the search strategy of TMFBS which efficiently reuses computations, in contrast to TIE* which has to restart the search for each candidate solution.

6.6.3 Multiple Solutions Graphs

We show two multiple solution graphs (constructed using Algorithm 14) on the solutions of first experiment for the CnC violent and CT slice datasets in Figure 6.7. The number of solutions are 3 and 20 for the CnC violent and CT slice datasets respectively. We see that the multiple solution graphs are able to efficiently encode all solutions, requiring only 5 and 15 nodes respectively. They also allow us to quickly identify interesting patterns. Recall that a solution can be read-off the graph by taking the union of features present in a path from s to t. Thus, the first node, which in both cases contains most of the features, corresponds to indispensable features. An example of replaceable features can be seen for the CnC dataset (graph on the left), where features 11 and 73 can be interchanged in all solutions. Another example can be seen on node 12 (graph on the right), which contains three sets of features (features 168, 186 and 196), which are replaceable for all solutions ob

Chapter 7
Conclusions

In this work we proposed several extensions of stepwise feature selection methods. We reviewed approaches from different fields and showed that a large class of existing methods can be expressed as stepwise selection methods. Not only does this unify them under a common framework, but it also enables the usage of techniques from different algorithms, such as the ones proposed in this work, by other algorithms that fit in this framework.

We presented a heuristic to speed-up the forward-backward feature selection algorithm (FBS), which gives rise to a family of algorithms, called forward-backward selection with early dropping (FBED^K). FBED^K is a general algorithm that can be adapted to handle different variable types (for example, continuous, categorical, ordinal), cross-sectional and time-course data, linear and non-linear dependencies, as well as different analysis tasks (for example, regression, classification, survival analysis) by using an appropriate conditional independence test. In contrast, algorithms like LASSO [139], although being computationally fast and performing well in terms of predictive performance for common problems like regression and classification, are not as general [77, 102, 131] and are computationally demanding for some problems [47, 60, 143]. Furthermore, we investigated the theoretical properties of three of its members, namely FBED^0, FBED^1 and FBED$^\infty$. We proved that, if the distribution of the data can be faithfully represented by a Bayesian network or maximal ancestral graph, FBED^1 and FBED$^\infty$ respectively can identify the optimal solution (the Markov blanket) of the target variable. In experiments we demonstrated that FBED^K behaves similarly to FBS in terms of predictive performance and number of selected variables, while being 1-2 orders of magnitude faster. Compared to other feature selection algorithms like LASSO [139] and MMPC [146], FBED^K has competitive predictive performance, while selecting the fewest variables, which is especially important if feature selection is performed for knowledge discovery. An interesting result is that FBED^K and LASSO perform about equally well, when limited to select the same number of variables. This, combined with the fact that FBED^K is more general, makes

it an attractive alternative to LASSO, especially for problems where no efficient solution to the LASSO problem exists.

Afterwards we proposed an extension of FBED^K for Big Data settings called Parallel, Forward-Backward with Pruning (PFBP). PFBP works on both dense and sparse data, and can scale to millions of predictive quantities (i.e., features, variables) and millions of training instances (samples). PFBP enables *computations that can be performed in a massively parallel way* by partitioning data both *horizontally* (over samples) and *vertically* (over features) and using meta-analysis techniques to combine results of local computations. Similar meta-analysis tricks can combine local logistic regression coefficients to global models with excellent results in practice against the global logistic regression models produced by MLlib. PFBP is equipped with *heuristics that can quickly and safely drop from consideration some of the redundant and irrelevant features* to significantly speed up computations. Bootstrapping testing allows PFBP to determine whether enough samples have been seen to safely apply the heuristics and forgo computations on the remaining samples. Our empirical analysis confirms that, PFBP exhibits a super-linear speedup with increasing sample size and a linear scalability with respect to the number of features and processing cores. A comparative evaluation shows that PFBP dominates other alternative map-reduce algorithms in its class in terms of computational performance, number of selected features, and predictive performance. Against information theoretic algorithms, specialized for sparse, discrete data it is slower, but returns models with higher predictive performance.

Finally, we presented a novel strategy for extending single-solution feature selection algorithms to identify multiple statistically equivalent solutions, and showed under which conditions it is able to identify all solutions. We extended the taxonomy of features proposed by John et al. [78] to take multiplicity into account. We also proposed three definitions of statistical equivalence of solutions, as well as methods for testing them. In experiments, we showed that the proposed algorithm is faster than the TIE* algorithm [138], the only other method with the same theoretical guarantees, while returning similar solutions. This happens because our algorithm directly takes advantage of the computations performed during the search, while TIE* does not.

Open Problems and Future Work

Next, we list several possible future directions for research.

Selecting Variables using Correlation with the Residuals

For all algorithms considered in this work, the forward phase selects the variable that, once selected, results in the best predictive model. This is computationally expensive,

as the algorithm has to fit one model for each candidate variable for selection. An alternative approach is to approximate this by selecting the variable which has the highest correlation with the residuals of the current model, similar to methods like orthogonal matching pursuit and least angle regression. This would dramatically reduce the computational time required for feature selection. An open problem is if and under which conditions this approximation retains the theoretical properties or the predictive performance of forward-backward selection.

Automatically Tuning the Significance Level

There are two main directions related to methods for automatically tuning the significance level used for selecting or removing variables. One is related to tuning the threshold for optimizing predictive performance, and the other regards minimizing false positive selections due to multiple testing. The latter is a problem of stepwise selection methods in general as, due to multiple testing, the p-values of the tests are too small [56, 67], leading to a high false discovery rate. One research direction is to perform a large-scale evaluation of information criteria for feature selection, as they can be seen as methods that automatically set the significance level (see Section 2.4.1). Example of such approaches are the extended Bayesian information criterion (EBIC) [29], the generalized information criterion (GIC) [49, 83], and the corrected risk information criterion (RIC_c) [165]. Another important direction is to combine feature selection methods (like the proposed FBED^K algorithm) with methods that deal with the problem of sequential testing of stepwise procedures [61, 140].

Improvements to the PFBP Algorithm

There are several improvements that can be made to the PFBP algorithm. As presented, PFBP requires the user to set the maximum number of features to select. This is mainly done to determine the number of samples to assign to each sample set, as the data are partitioned once before running the algorithm. One way to overcome this is to devise a principled criterion to automatically re-partition the data. Apart from removing the hard limit on the maximum number of features to select, this also has the advantage that it can speed-up the algorithm (as seen on the SNP data application in Section 5.5.4, where re-partitioning was used after the univariate phase). Other, relatively simple extensions, which could lead to significant reductions in running time, are to exploit the sparsity of the data and to use specialized GPU implementations of conditional independence tests.

Multiple Solutions for Stepwise Methods

The proposed strategy for multiple solutions is for methods that can be expressed using separate forward and backward phases. An important future direction is to extend the proposed strategy and heuristics for a more general class of algorithms, namely methods that search in the space of solutions by adding or removing one or multiple variables at each iteration. This would allow extending other methods for multiple solutions, like recursive feature elimination, lasso [139], and stepwise selection [88,156] methods, to name a few.

Learning Multiple Equivalent Causal Networks

The problem of feature selection and causal discovery are closely related. Many feature selection methods, such as the ones proposed in this work, are inspired by causal modeling. Furthermore, feature selection is often used as a first step before learning a full causal network. For example, the max-min hill-climbing Bayesian network structure learning algorithm [149] first applies the MMPC feature selection algorithm [146] on each variable to find its neighbors, and then uses a scoring phase to piece them together and learn a Bayesian network. Methods for learning causal networks assume the existence of a single underlying causal network, and learn a Markov equivalence class (i.e., a set of networks which encode the same conditional independencies). However, it may be the case that there exist multiple, statistically equivalent networks that are not Markov equivalent, similar to the existence of multiple Markov blankets of a target variable. Methods, such as the one proposed in this work, could possibly be used as building blocks for learning multiple networks, similar to how single solution methods are used to learn a single network.

Bibliography

[1] Alan Agresti. *Categorical Data Analysis*. Wiley Series in Probability and Statistics. Wiley-Interscience, 2 edition, 2002.

[2] Hirotogu Akaike. Information theory and an extension of the maximum likelihood principle. In *Second International Symposium on Information Theory*, pages 267–281, Budapest, 1973. Akadémiai Kiado.

[3] Anastasios Alexandridis, Giorgos Borboudakis, and Athanasios Mouchtaris. Addressing the data-association problem for multiple sound source localization using DOA estimates. In *Signal Processing Conference (EUSIPCO), 2015 23rd European*, pages 1551–1555. IEEE, 2015.

[4] Constantin F. Aliferis, Alexander Statnikov, Ioannis Tsamardinos, Subramani Mani, and Xenofon D. Koutsoukos. Local causal and Markov blanket induction for causal discovery and feature selection for classification part i: Algorithms and empirical evaluation. *Journal of Machine Learning Research*, 11(Jan):171–234, 2010.

[5] Constantin F. Aliferis, Ioannis Tsamardinos, and Alexander Statnikov. HITON: a novel Markov blanket algorithm for optimal variable selection. In *AMIA Annual Symposium Proceedings*, volume 2003, page 21. American Medical Informatics Association, 2003.

[6] Henrik Reif Andersen. An introduction to binary decision diagrams. *Lecture notes, available online, IT University of Copenhagen*, 1997.

[7] Larry Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, 1966.

[8] Anthony C. Atkinson. A method for discriminating between models. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 323–353, 1970.

[9] Betsy Jane Becker and Meng-Jia Wu. The synthesis of regression slopes in meta-analysis. *Statistical Science*, pages 414–429, 2007.

[10] Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2):813–852, 2016.

[11] Thomas Blumensath and Mike E. Davies. On the difference between Orthogonal Matching Pursuit and Orthogonal Least Squares. Technical report, 2007.

[12] Beate Bollig and Ingo Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

[13] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. *Feature Selection for High-Dimensional Data*. Springer Publishing Company, Incorporated, 1 edition, 2015.

[14] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. Recent advances and emerging challenges of feature selection in the context of Big Data. *Knowledge-Based Systems*, 86:33–45, 2015.

[15] Verónica Bolón-Canedo, Konstantinos Sechidis, Noelia Sánchez-Marono, Amparo Alonso-Betanzos, and Gavin Brown. Exploring the consequences of distributed feature selection in DNA microarray data. In *International Joint Conference on Neural Networks*, pages 1665–1672, 2017.

[16] Giorgos Borboudakis, Taxiarchis Stergiannakos, Maria Frysali, Emmanuel Klontzas, Ioannis Tsamardinos, and George E Froudakis. Chemically intuited, large-scale screening of MOFs by machine learning techniques. *npj Computational Materials*, 3(1):40, 2017.

[17] Giorgos Borboudakis and Ioannis Tsamardinos. Bayesian network learning with discrete case-control data. In *UAI*, pages 151–160, 2015.

[18] Giorgos Borboudakis and Ioannis Tsamardinos. Towards robust and versatile causal discovery for business applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1435–1444. ACM, 2016.

[19] Joseph K. Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for L1-regularized loss minimization. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 321–328, 2011.

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

[21] Dale S. Bremmer. J-tests: To nest or not to nest, that is the question. In *79th Annual Conference of the Western Economics Association*, 2003.

[22] Gavin Brown, Adam Pocock, Ming-Jie Zhao, and Mikel Luján. Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. *Journal of Machine Learning Research*, 13:27–66, January 2012.

[23] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on computers*, 100(8):677–691, 1986.

[24] Krisztian Buza. Feedback prediction for blogs. In *Data analysis, machine learning and knowledge discovery*, pages 145–152. Springer, 2014.

[25] Oriol Canela-Xandri, Andy Law, Alan Gray, John A. Woolliams, and Albert Tenesa. A new tool called DISSECT for analysing large genomic data sets using a Big Data approach. *Nature communications*, 6, 2015.

[26] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27, 2011.

[27] Christopher C. Chang, Carson C. Chow, Laurent C. A. M. Tellier, Shashaank Vattikuti, Shaun M. Purcell, and James J. Lee. Second-generation PLINK: rising to the challenge of larger and richer datasets. *Gigascience*, 4(1):7, 2015.

[28] M. Aslam Chaudhry and Syed M. Zubair. *On a class of incomplete gamma functions with applications*. CRC press, 2001.

[29] Jiahua Chen and Zehua Chen. Extended Bayesian information criteria for model selection with large model spaces. *Biometrika*, 95(3):759–771, 2008.

[30] Sheng Chen, Stephen A Billings, and Wan Luo. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of control*, 50(5):1873–1896, 1989.

[31] Ronald Christensen. *Plane answers to complex questions: the theory of linear models*. Springer Science & Business Media, 2011.

[32] Francis S. Collins and Harold Varmus. A new initiative on precision medicine. *New England Journal of Medicine*, 372(9):793–795, 2015.

[33] International HapMap Consortium. A haplotype map of the human genome. *Nature*, 437(7063):1299–1320, 2005.

[34] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[35] Jan Daciuk. Comparison of construction algorithms for minimal, acyclic, deterministic, finite-state automata from sets of strings. In *International Conference on Implementation and Application of Automata*, pages 255–261. Springer, 2002.

[36] Jan Daciuk, Stoyan Mihov, Bruce W. Watson, and Richard E. Watson. Incremental construction of minimal acyclic finite-state automata. *Computational linguistics*, 26(1):3–16, 2000.

[37] Samuel A. Danziger, S. Joshua Swamidass, Jue Zeng, Lawrence R. Dearth, Qiang Lu, Jonathan H. Chen, Jianlin Cheng, Vinh P. Hoang, Hiroto Saigo, Ray Luo, et al. Functional census of mutation sequence spaces: the example of p53 cancer rescue mutants. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(2):114–125, 2006.

[38] Russell Davidson and James G. MacKinnon. Several tests for model specification in the presence of alternative hypotheses. *Econometrica: Journal of the Econometric Society*, pages 781–793, 1981.

[39] Geoffrey M. Davis, Stephane G. Mallat, and Zhifeng Zhang. Adaptive time-frequency decompositions. *Optical engineering*, 33(7):2183–2192, 1994.

[40] Anthony Christopher Davison and David Victor Hinkley. *Bootstrap methods and their application*, volume 1. Cambridge university press, 1997.

[41] Thomas G. Dietterich, Ajay N. Jain, Richard H. Lathrop, and Tomas Lozano-Perez. A comparison of dynamic reposing and tangent distance for drug activity prediction. *Advances in Neural Information Processing Systems*, pages 216–216, 1994.

[42] Edward R. Dougherty and Marcel Brun. On the number of close-to-optimal feature sets. *Cancer informatics*, 2:189–196, 2006.

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

[44] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004.

[45] Bradley Efron and Robert J. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[46] Robert F. Engle. Wald, likelihood ratio, and Lagrange multiplier tests in econometrics. *Handbook of econometrics*, 2:775–826, 1984.

[47] Jianqing Fan, Yang Feng, Yichao Wu, et al. High-dimensional variable selection for Cox's proportional hazards model. In *Borrowing Strength: Theory Powering Applications–A Festschrift for Lawrence D. Brown*, pages 70–86. Institute of Mathematical Statistics, 2010.

[48] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

[49] Yingying Fan and Cheng Yong Tang. Tuning parameter selection in high dimensional penalized likelihood. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(3):531–552, 2013.

[50] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems. *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

[51] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors, *AutoML: Methods, Sytems, Challenges*, chapter 1, pages 3–37. Springer, December 2018. To appear.

[52] Livio Finos, Chiara Brombin, and Luigi Salmaso. Adjusting stepwise p-values in generalized linear models. *Communications in Statistics-Theory and Methods*, 39(10):1832–1846, 2010.

[53] R. A. Fisher. *Statistical methods for research workers*. Edinburgh Oliver & Boyd, 1932.

[54] Ronald Aylmer Fisher. The distribution of the partial correlation coefficient. *Metron*, 3:329–332, 1924.

[55] François Fleuret. Fast binary feature selection with conditional mutual information. *Journal of Machine Learning Research*, 5(Nov):1531–1555, 2004.

[56] Peter L. Flom and David L. Cassell. Stopping stepwise: Why stepwise and similar selection methods are bad, and what you should use. In *NorthEast SAS Users Group Inc 20th Annual Conference*, 2007.

[57] Robert V. Foutz and R. C. Srivastava. The performance of the likelihood ratio test when the model is incorrect. *The Annals of Statistics*, 5(6):1183–1194, 1977.

[58] Richard M. Golden. Discrepancy risk model selection test theory for comparing possibly misspecified or nonnested models. *Psychometrika*, 68(2):229–249, 2003.

[59] Franz Graf, Hans-Peter Kriegel, Matthias Schubert, Sebastian Pölsterl, and Alexander Cavallaro. 2D image registration in CT images using radial image descriptors. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 607–614. Springer, 2011.

[60] Andreas Groll and Gerhard Tutz. Variable selection for generalized linear mixed models by L1-penalized estimation. *Statistics and Computing*, 24(2):137–154, 2014.

[61] Max Grazier G'Sell, Stefan Wager, Alexandra Chouldechova, and Robert Tibshirani. Sequential selection procedures and false discovery rate control. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 78(2):423–444, 2016.

[62] Isabelle Guyon, Amir Reza Saffari Azar Alamdari, Gideon Dror, and Joachim M. Buhmann. Performance prediction challenge. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 1649–1656. IEEE, 2006.

[63] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[64] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the NIPS 2003 feature selection challenge. In *Advances in neural information processing systems*, pages 545–552, 2004.

[65] Isabelle Guyon, Jiwen Li, Theodor Mader, Patrick A Pletscher, Georg Schneider, and Markus Uhr. Feature selection with the CLOP package. Technical report, 2006.

[66] Mazin Abdulrasool Hameed. Comparative analysis of orthogonal matching pursuit and least angle regression. Master's thesis, Michigan State University, Electrical Engineering, 2012.

[67] Frank Harrell. *Regression Modeling Strategies*. Springer, corrected edition, January 2001.

[68] Trevor Hastie, Jonathan Taylor, Robert Tibshirani, Guenther Walther, et al. Forward stagewise regression and the monotone lasso. *Electronic Journal of Statistics*, 1:1–29, 2007.

[69] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

[70] Trevor Hastie, Robert Tibshirani, and Ryan J Tibshirani. Extended comparisons of best subset selection, forward stepwise selection, and the lasso. *arXiv preprint arXiv:1707.08692*, 2017.

[71] He He, Hal Daumé III, and Jason Eisner. Cost-sensitive dynamic feature selection. In *ICML Inferning Workshop*, 2012.

[72] Larry V. Hedges and Jack L. Vevea. Fixed-and random-effects models in meta-analysis. *Psychological methods*, 3(4):486, 1998.

[73] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[74] David W. Hosmer, Jr., Stanley Lemeshow, and Rodney X. Sturdivant. *Introduction to the Logistic Regression Model*. John Wiley & Sons, Inc., 2013.

[75] Grace T Huang, Ioannis Tsamardinos, Vineet Raghu, Naftali Kaminski, and Panayiotis V Benos. T-ReCS: stable selection of dynamically formed groups of features with application to prediction of clinical outcomes. In *Pacific Symposium on Biocomputing Co-Chairs*, pages 431–442. World Scientific, 2014.

[76] Jing-Shiang Hwang and Tsuey-Hwa Hu. A stepwise regression algorithm for high-dimensional variable selection. *Journal of Statistical Computation and Simulation*, 85(9):1793–1806, 2015.

[77] Stéphane Ivanoff, Franck Picard, and Vincent Rivoirard. Adaptive lasso and group-lasso for functional Poisson regression. *J. Mach. Learn. Res.*, 17(1):1903–1948, January 2016.

[78] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Machine learning: Proceedings of the Eleventh International Conference*, pages 121–129. Morgan Kaufmann, 1994.

[79] Markus Kalisch and Peter Bühlmann. Estimating high-dimensional directed acyclic graphs with the PC-algorithm. *Journal of Machine Learning Research*, 8(Mar):613–636, 2007.

[80] Alexandros Kalousis, Julien Prados, and Melanie Hilario. Stability of feature selection algorithms: a study on high-dimensional spaces. *Knowledge and information systems*, 12(1):95–116, 2007.

[81] Karen-Inge Karstoft, Isaac R. Galatzer-Levy, Alexander Statnikov, Zhiguo Li, and Arieh Y. Shalev. Bridging a translational gap: using machine learning to improve the prediction of PTSD. *BMC Psychiatry*, 15(1):30, March 2015.

[82] Randy Kerber. Chimerge: Discretization of numeric attributes. In *Proceedings of the tenth national conference on Artificial intelligence*, pages 123–128. Aaai Press, 1992.

[83] Yongdai Kim, Sunghoon Kwon, and Hosik Choi. Consistent model selection criteria on high dimensions. *Journal of Machine Learning Research*, 13(Apr):1037–1057, 2012.

[84] Jonas R. Klasen, Elke Barbez, Lukas Meier, Nicolai Meinshausen, Peter Bühlmann, Maarten Koornneef, Wolfgang Busch, and Korbinian Schneeberger. A multi-marker association method for genome-wide association studies without the need for population structure correction. *Nature communications*, 7:13299, 2016.

[85] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324, December 1997.

[86] Daphne Koller and Mehran Sahami. Toward optimal feature selection. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 284–292, 1996.

[87] Pradap Konda, Arun Kumar, Christopher Ré, and Vaishnavi Sashikanth. Feature selection in enterprise analytics: A demonstration using an R-based data analytics system. *Proceedings of the VLDB Endowment*, 6(12):1306–1309, August 2013.

[88] Michael H. Kutner, Christopher J. Nachtsheim, John Neter, and William Li. *Applied Linear Statistical Models*. McGraw-Hill/Irwin, 5 edition, August 2004.

[89] Vincenzo Lagani, Giorgos Athineou, Alessio Farcomeni, Michail Tsagris, and Ioannis Tsamardinos. Feature selection with the R package MXM: Discovering statistically equivalent feature subsets. *Journal of Statistical Software*, 80(7), 2017.

[90] Vincenzo Lagani, George Kortas, and Ioannis Tsamardinos. Biomarker signature identification in "omics" data with multi-class outcomes. *Computational and structural biotechnology journal*, 6(7):1–7, 2013.

[91] Vincenzo Lagani and Ioannis Tsamardinos. Structure-based variable selection for survival data. *Bioinformatics*, 26(15):1887–1894, 2010.

[92] Seunghak Lee, Jin Kyu Kim, Xun Zheng, Qirong Ho, Garth A. Gibson, and Eric P. Xing. On Model Parallelization and Scheduling Strategies for Distributed Machine Learning. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2834–2842, 2014.

[93] Jan Lemeire. Learning causal models of multivariate systems and the value of it for the performance modeling of computer programs, 2007.

[94] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Computing Surveys*, 50(6):94:1–94:45, December 2017.

[95] Qingyang Li, Shuang Qiu, Shuiwang Ji, Paul M. Thompson, Jieping Ye, and Jie Wang. Parallel lasso screening for Big Data optimization. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1705–1714, New York, NY, USA, 2016. ACM.

[96] Huawen Liu, Lei Liu, and Huijie Zhang. Ensemble gene selection by grouping for microarray data classification. *Journal of biomedical informatics*, 43(1):81–87, 2010.

[97] Thomas M. Loughin. A systematic comparison of methods for combining p-values from independent tests. *Computational statistics & data analysis*, 47(3):467–485, 2004.

[98] Jinchi Lv and Jun S. Liu. Model selection principles in misspecified models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(1):141–167, 2014.

[99] James G. MacKinnon. Model specification tests against non-nested alternatives. *Econometric Reviews*, 2(1):85–110, 1983.

[100] Dimitris Margaritis. Toward provably correct feature selection in arbitrary domains. In *Advances in Neural Information Processing Systems*, pages 1240–1248, 2009.

[101] Dimitris Margaritis and Sebastian Thrun. Bayesian network induction via local neighborhoods. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 505–511. MIT Press, 2000.

[102] Lukas Meier, Sara Van De Geer, and Peter Bühlmann. The group Lasso for logistic regression. *Journal of the Royal Statistical Society, Series B*, 2008.

[103] Nicolai Meinshausen and Peter Bühlmann. High-dimensional graphs and variable selection with the lasso. *The Annals of Statistics*, pages 1436–1462, 2006.

[104] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D. B. Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Research*, 17(1):1235–1241, January 2016.

[105] Stefan Michiels, Serge Koscielny, and Catherine Hill. Prediction of cancer outcome with microarrays: a multiple random validation strategy. *The Lancet*, 365(9458):488–492, 2005.

[106] Alan Miller. *Subset selection in regression*. CRC Press, 2002.

[107] Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *30th Conference on Design Automation*, pages 272–277. IEEE, 1993.

[108] Shin-ichi Minato. Zero-suppressed BDDs and their applications. *International Journal on Software Tools for Technology Transfer*, 3(2):156–170, 2001.

[109] Thomas P. Minka. A comparison of numerical optimizers for logistic regression. *Unpublished draft*, 2003.

[110] Ryuhei Miyashiro and Yuichi Takano. Subset selection by Mallows' Cp: A mixed integer programming approach. *Expert Systems with Applications*, 42(1):325–331, 2015.

[111] Andrew Y. Ng. Feature selection, L1 vs. L2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[112] Sarah Nogueira and Gavin Brown. Measuring the stability of feature selection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 442–457. Springer, 2016.

[113] Yagyensh Chandra Pati, Ramin Rezaiifar, and Perinkulam Sambamurthy Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, pages 40–44. IEEE, 1993.

[114] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[115] Judea Pearl. *Causality, Models, Reasoning, and Inference*. Cambridge University Press, Cambridge, U.K., 2000.

[116] Peter Peduzzi, John Concato, Elizabeth Kemper, Theodore R. Holford, and Alvan R. Feinstein. A simulation study of the number of events per variable in logistic regression analysis. *Journal of clinical epidemiology*, 49(12):1373–1379, 1996.

[117] Jose M. Peña, Roland Nilsson, Johan Björkegren, and Jesper Tegnér. Towards scalable and data efficient learning of Markov boundaries. *International Journal of Approximate Reasoning*, 45(2):211–232, 2007.

[118] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8):1226–1238, 2005.

[119] M. Hashem Pesaran and Melvyn Weeks. Non-nested Hypothesis Testing: An Overview. Cambridge Working Papers in Economics 9918, September 1999.

[120] Junyang Qian, Ttrevor Hastie, Jerome Friedman, Rob Tibshirani, and Noah Simon. Glmnet for Matlab, 2013.

[121] S. Ramírez-Gallego, H. Mouriño-Talín, D. Martínez-Rego, V. Bolón-Canedo, J. M. Benítez, A. Alonso-Betanzos, and F. Herrera. An information theory-based feature selection framework for Big Data under Apache Spark. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, PP(99):1–13, 2017.

[122] Michael A. Redmond and Timothy Highley. Empirical analysis of case-editing approaches for numeric prediction. In *Innovations in Computing Sciences and Software Engineering*, pages 79–84. Springer, 2010.

[123] Dominique Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(1):181–189, 1992.

[124] Michael Rice and Sanjay Kulhari. A survey of static variable ordering heuristics for efficient BDD / MDD construction. Technical report, 2008.

[125] Thomas Richardson. Markov properties for acyclic directed mixed graphs. *Scandinavian Journal of Statistics*, 30(1):145–157, 2003.

[126] Thomas Richardson and Peter Spirtes. Ancestral graph Markov models. *Annals of Statistics*, pages 962–1030, 2002.

[127] Paul Roepman, Patrick Kemmeren, Lodewijk F. A. Wessels, Piet J. Slootweg, and Frank C. P. Holstege. Multiple robust signatures for detecting lymph node metastasis in head and neck cancer. *Cancer Research*, 66(4):2361–2366, 2006.

[128] Anna Roumpelaki, Giorgos Borboudakis, Sofia Triantafillou, and Ioannis Tsamardinos. Marginal causal consistency in constraint-based causal learning. In *CFA@ UAI*, pages 39–47, 2016.

[129] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[130] Toshiki Sato, Yuichi Takano, Ryuhei Miyashiro, and Akiko Yoshise. Feature subset selection for logistic regression via mixed integer optimization. *Computational Optimization and Applications*, 64(3):865–880, 2016.

[131] Jürg Schelldorfer, Peter Bühlmann, and Sara Van De Geer. Estimation for high-dimensional linear mixed-effects models using L1-penalization. *Scandinavian Journal of Statistics*, 38(2):197–214, 2011.

[132] Gideon Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[133] Stephen T. Sherry, M.-H. Ward, M. Kholodov, J. Baker, Lon Phan, Elizabeth M. Smigielski, and Karl Sirotkin. dbSNP: the NCBI database of genetic variation. *Nucleic acids research*, 29(1):308–311, 2001.

[134] Xiaoxia Shi. A nondegenerate vuong test. *Quantitative Economics*, 6(1):85–121, 2015.

[135] Sameer Singh, Jeremy Kubica, Scott Larsen, and Daria Sorokina. Parallel large scale feature selection for logistic regression. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 1172–1183. SIAM, 2009.

[136] Peter Spirtes, Clark N. Glymour, and Richard Scheines. *Causation, prediction, and search*. MIT press, 2 edition, 2000.

[137] Alexander Statnikov and Constantin F. Aliferis. Analysis and computational dissection of molecular signature multiplicity. *PLoS computational biology*, 6(5):1–9, 2010.

[138] Alexander Statnikov, Nikita I. Lytkin, Jan Lemeire, and Constantin F. Aliferis. Algorithms for discovery of multiple Markov boundaries. *Journal of Machine Learning Research*, 14(Feb):499–566, 2013.

[139] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[140] Ryan J. Tibshirani, Jonathan Taylor, Richard Lockhart, and Robert Tibshirani. Exact post-selection inference for sequential regression procedures. *Journal of the American Statistical Association*, 111(514):600–620, 2016.

[141] Joaquín Torres-Sospedra, Raúl Montoliu, Adolfo Martínez-Usó, Joan P Avariento, Tomás J Arnau, Mauri Benedito-Bordonau, and Joaquín Huerta. Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems. In *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 261–270. IEEE, 2014.

[142] Michail Tsagris, Giorgos Borboudakis, Vincenzo Lagani, and Ioannis Tsamardinos. Constraint-based causal discovery with mixed data. *International Journal of Data Science and Analytics*, pages 1–12, 2018.

[143] Michail Tsagris, Vincenzo Lagani, and Ioannis Tsamardinos. Feature selection for high-dimensional temporal data. *BMC bioinformatics*, 19(1):17, 2018.

[144] I. Tsamardinos, V. Lagani, and D. Pappas. Discovering multiple, equivalent biomarker signatures. In *7th Conference of the Hellenic Society for Computational Biology and Bioinformatics (HSCBB12)*, 2012.

[145] Ioannis Tsamardinos and Constantin F. Aliferis. Towards principled feature selection: relevancy, filters and wrappers. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 2003.

[146] Ioannis Tsamardinos, Constantin F. Aliferis, and Alexander Statnikov. Time and sample efficient discovery of Markov blankets and direct causal relations. In *Proceedings of the Ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 673–678. ACM, 2003.

[147] Ioannis Tsamardinos, Constantin F. Aliferis, and Alexander R. Statnikov. Algorithms for large scale Markov blanket discovery. In *FLAIRS conference*, volume 2, 2003.

[148] Ioannis Tsamardinos, Giorgos Borboudakis, Pavlos Katsogridakis, Polyvios Pratikakis, and Vassilis Christophides. A greedy feature selection algorithm for Big Data of high dimensionality. *Machine Learning*, Aug 2018.

[149] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine learning*, 65(1):31–78, 2006.

[150] Ioannis Tsamardinos, Elissavet Greasidou, and Giorgos Borboudakis. Bootstrapping the out-of-sample predictions for efficient and accurate cross-validation. *Machine Learning*, May 2018.

[151] Ioannis Tsamardinos and Asimakis P. Mariglis. Multi-source causal analysis: Learning Bayesian networks from multiple datasets. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 479–490. Springer, 2009.

[152] T. Verma and Pearl. Causal Networks: Semantics and Expressiveness. In *Proceedings, 4th Workshop on Uncertainty in Artificial Intelligence*, pages 352–359, August 1988.

[153] Eric Vittinghoff and Charles E. McCulloch. Relaxing the rule of ten events per variable in logistic and Cox regression. *American journal of epidemiology*, 165(6):710–718, 2007.

[154] Quang H. Vuong. Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica: Journal of the Econometric Society*, pages 307–333, 1989.

[155] Xiangyu Wang, David B. Dunson, and Chenlei Leng. DECOrrelated feature space partitioning for distributed sparse regression. In *Advances in Neural Information Processing Systems*, pages 802–810, 2016.

[156] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005.

[157] William J. Welch. Algorithmic complexity: three NP-hard problems in computational statistics. *Journal of Statistical Computation and Simulation*, 15(1):17–25, 1982.

[158] Halbert White. Maximum likelihood estimation of misspecified models. *Econometrica*, 50(1):1–25, 1982.

[159] Samuel S. Wilks. The large-sample distribution of the likelihood ratio for testing composite hypotheses. *The Annals of Mathematical Statistics*, 9(1):60–62, March 1938.

[160] Eric P. Xing, Qirong Ho, Pengtao Xie, and Dai Wei. Strategies and principles of distributed machine learning on Big Data. *Engineering*, 2(2):179–195, 2016.

[161] Howard Hua Yang and John Moody. Data visualization and feature selection: New algorithms for nongaussian data. In *Advances in Neural Information Processing Systems*, pages 687–693, 2000.

[162] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *HotCloud*, 2010.

[163] Yiteng Zhai, Yew-Soon Ong, and Ivor W. Tsang. The Emerging "Big Dimensionality". *IEEE Comp. Int. Mag.*, 9(3):14–26, 2014.

[164] Kun Zhang, Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. Kernel-based conditional independence test and application in causal discovery. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 804–813, 2011.

[165] Yongli Zhang and Xiaotong Shen. Model selection procedure for high-dimensional data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 3(5):350–358, 2010.

[166] Zheng Zhao, Ruiwen Zhang, James Cox, David Duling, and Warren Sarle. Massively parallel feature selection: an approach based on variance preservation. *Machine Learning*, 92(1):195–220, 2013.

[167] Peng Zhimin, Yan Ming, and Yin Wotao. Parallel and distributed sparse optimization. In *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, 2013.

[168] Yingbo Zhou, Utkarsh Porwal, Ce Zhang, Hung Q. Ngo, XuanLong Nguyen, Christopher Ré, and Venu Govindaraju. Parallel feature selection inspired by group testing. In *Advances in Neural Information Processing Systems*, pages 3554–3562, 2014.

[169] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67:301–320, 2005.

Appendix A
Proofs

For all of the proofs we assume that the algorithms have access to an **independence oracle** that can perfectly determine whether a given conditional dependence or independence holds. Furthermore, in all proofs we will use the terms d-connected/m-connected (d-separated/m-separated) and dependent (independent) interchangeably; this is possible due to the faithfulness assumption.

Lemma 2. *Let A, T be variables and* $\mathbf{B}$, $\mathbf{C}$ *sets of variables. Then* $T \perp A \mid \boldsymbol{B} \cup \mathbf{C} \wedge T \perp \boldsymbol{B} \mid \mathbf{C} \Rightarrow T \perp A \mid \mathbf{C}$ *holds for any such variables.*

Proof.

$T \perp A \mid \mathbf{B} \cup \mathbf{C} \wedge T \perp \mathbf{B} \mid \mathbf{C} \Rightarrow$ (**Contraction**)

$T \perp A \cup \mathbf{B} \mid \mathbf{C} \Rightarrow$ (**Decomposition**)

$T \perp A \mid \mathbf{C} \wedge T \perp \mathbf{B} \mid \mathbf{C}$

□

The following lemma will be useful for proving some of the theorems.

Lemma 3. *Let* $\mathbf{S}$ *be a set of variables selected for some target T and* $\mathbf{R} = \mathbf{F} \setminus \mathbf{S}$. *Assume that* $\forall F_r \in \mathbf{R}\ T \perp F_r \mid \mathbf{S}$ *holds. Then, if* $\exists F_s \in \mathbf{S}$ *such that* $T \perp F_s \mid \mathbf{S} \setminus F_s$ *holds,* $\forall F_r \in \mathbf{R}\ T \perp F_r \mid \mathbf{S} \setminus F_s$ *also holds.*

Proof. We are given that $\forall F_r \in \mathbf{R}\ T \perp F_r \mid \mathbf{S}$ holds. By applying Lemma 2 to each variable in $F_r \in \mathbf{R}$ with $A = F_r$, $\mathbf{B} = \{F_s\}$ and $\mathbf{C} = \mathbf{S} \setminus F_s$, we get that $T \perp F_r \mid F_s \cup (\mathbf{S} \setminus F_s) \wedge T \perp F_s \mid \mathbf{S} \setminus F_s \Rightarrow T \perp F_r \mid \mathbf{S} \setminus F_s$ holds for any such F_r, which concludes the proof. □

To put it simple, Lemma 3 states that if we remove any variable F_s from a set of selected variables $\mathbf{S}$ by conditioning on $\mathbf{S} \setminus F_s$, no variable that is not in $\mathbf{S}$ becomes conditionally dependent with T given $\mathbf{S} \setminus F_s$. In practice this means that removing variables using backward selection from a set of variables selected by forward selection

will not create any additional conditional dependencies, meaning that we do not have to reconsider them again.

A.1 Proof of Corollary 1

Proof. To show that **S** is minimal, we have to show the following

i $\forall F_s \in \mathbf{S}\ T \not\perp F_s \mid \mathbf{S} \setminus F_s$ (*No variable can be removed*)

ii $\forall F_r \in \mathbf{F}_{\mathcal{D}} \setminus \mathbf{S}, T \perp F_r \mid \mathbf{S}$ (*No variable can be added*)

Proof of (i): This holds trivially, as backward selection removes any variable $F_s \in \mathbf{S}$ if $T \perp F_s \mid \mathbf{S} \setminus F_s$ holds.
Proof of (ii): We know that after the termination of forward selection, no variable can be added, that is, $\forall F_r \in \mathbf{R}\ T \perp F_r \mid \mathbf{S}$ holds. Given that, Lemma 3 can be repeatedly applied after each variable removal by backward selection, and thus no variable in **R** can be added to **S**. □

A.2 Proof of Corollary 2

Proof. As is the case with FBS, the forward selection phase of FBED$^{\infty}$ stops if no more variables can be included. Using this fact, the proof is identical to the one of Theorem 1. □

A.3 Proof of Theorem 1

Proof. In the first run of FBED^1, all variables that are adjacent to T (that is, its parents and children) will be selected, as none of them can be d-separated from T by any set of variables. In the next run, all variables connected through a collider path of length 2 (that is, the spouses of T) will become d-connected with T, since the algorithm conditions on all selected variables (including its children), and thus will be selected. The resulting set of variables includes the Markov blanket of T, but may also include additional variables. Next we show that all additional variables will be removed by the backward selection phase. Let MB(T) be the Markov blanket of T and $\mathbf{S_{ind}} = \mathbf{S} \setminus \mathrm{MB}(T)$ be all selected variables not in the Markov blanket of T. By definition, $T \perp \mathbf{X} \mid MB(T)$ holds for any set of variables **X** not in MB(T), and thus also for variables $\mathbf{S_{ind}}$. By applying the weak union graphoid axiom, one can infer that $\forall S_i \in \mathbf{S_{ind}}, T \perp S_i \mid MB(T) \cup \mathbf{S_{ind}} \setminus S_i$ holds, and thus some variable S_j will be removed in the first iteration. Using the same reasoning and the definition of a Markov blanket, it can be shown that all variables in $\mathbf{S_{ind}}$ will be removed from MB(T) at

some iteration. To conclude, it suffices to use the fact that variables in MB(T) will not be removed by the backward selection, as they are not conditionally independent of T given the remaining variables in MB(T). □

A.4 Proof of Theorem 2

Proof. In the first run of FBED$^{\infty}$, all variables that are adjacent to T (that is, its parents, children and variables connected with T by a bi-directed edge) will be selected, as none of them can be m-separated from T by any set of variables. After each run additional variables may become admissible for selection. Specifically, after k runs all variables that are connected with T by a collider path of length k will become m-connected with T, and thus will be selected; we prove this next. Assume that after k runs all variables connected with T by a collider path of length at most $k-1$ have been selected. By conditioning on all selected variables, all variables that are into some selected variable connected with T by a collider path will become m-connected with T. This is true because conditioning on a variable Y in a collider $\langle X, Y, Z\rangle$ m-connects X and Z. By applying this on each variable on some collider path, it is easy to see that its end-points become m-connected. Finally, after applying the backward selection phase, all variables that are not in the Markov blanket of T will be removed; the proof is identical to the one used in the proof of Theorem 1 and thus will be omitted. □

A.5 Proof of Theorem 3

TMFBS using $\mathcal{A}$ will identify all and only the solutions equivalent with the reference solution in any dataset $\mathcal{D}$, if (a) $\mathcal{A}$ satisfies Assumption 10 and, (b) it has access to an oracle for deciding equivalence.

Proof. Let $\mathbf{S} \subseteq \mathbf{F}$ be an arbitrary solution equivalent to the reference solution. We will show inductively that any such solution can be obtained by running TMFBS.

Let $\mathbf{S}^j$ be the current solution after j steps, and let $\mathbf{C}^j$ be the corresponding candidate variables returned by ORDERVARIABLES given $\mathbf{S}^j$. Assume that $\mathbf{S}^j \subseteq \mathbf{S}$. We will show that, if progress can be made (i.e., if $\mathbf{S}^j$ is not a solution), then there is a neighbor state such that $\mathbf{S}^{j+1} \subseteq \mathbf{S}$.

If $\mathbf{S}^j$ equals $\mathbf{S}$, then $\mathbf{C}^j$ is empty (condition (a) of admissibility of ORDERVARIABLES), and the algorithm would terminate and return the solution. We will prove the $\mathbf{S}^j \subset \mathbf{S}$ case by contradiction. Assume that $\mathbf{C}^j$ does not contain any variable of $\mathbf{S}$. Because of that, $\mathbf{C}^j$ could be excluded without altering the solution, which is equivalent to executing the algorithm on $\mathcal{D}^{\mathbf{C}^j}$ by Lemma 9. After j steps this would lead to a state where $\mathbf{C}^j$ is empty, and the algorithm would terminate. However, as $\mathbf{S}^j$ is a subset of $\mathbf{S}$ it can't be a solution, given that

S is a solution, otherwise **S** wouldn't be minimal. This would violate Assumption 10 for $\mathcal{D}^{C^j}$ which contains **S**, leading to a contradiction. Therefore $\mathbf{C}^j$ must contain some variable from $\mathbf{S} \setminus \mathbf{S}^j$, and consequently there is a neighbor state such that, after $j+1$ steps, $\mathbf{S}^{j+1} \subseteq \mathbf{S}$ holds.

Finally, because TMFBS has access to an oracle for deciding statistical equivalence, any false positive solutions identified during the search will be discarded, retaining only equivalent solutions. □

A.6 Proof of Theorem 4

Proof. Without loss of generality, we only prove the correctness of Operation 1. The correctness of Operation 2 can be shown following the same reasoning.

To prove correctness, it suffices to show that all paths into any node in **P** and out of any node N_i contain the same set of variables. Let P_j be some parent of N_i. Initially, the set of variables on the path through P_j and N_i are $var[P_j] \cup var[N_i]$. After performing the forward merging operation, the set of represented variables is not affected, as $var[P_j] \cup var[N'] \cup (var[N_i] \setminus var[N']) = var[P_j] \cup var[N_i]$. Since the set of incoming edges of each P_j as well as the set outgoing edges of each N_i do not change, the set of represented solutions of the initial DAG is not altered. □

Appendix B
Implementation Details

B.1 Bootstrap Test For Comparing Algorithms

We used bootstrapping to compute the probability that algorithm A_i is better/worse or equal than all others in terms of some measure of interest f (for example, AUC), that is $P(\wedge_{j \neq i} f(A_i) \geq f(A_j))$. The procedure is described next. Let $f_{i,k}$ denote the measure of interest of algorithm i on test set k. We resample with replacement $B = 100000$ times the test sets and compute f, denoted as $f_{i,k,b}$ for the b-th sample of algorithm i and test set k. Then, $f_{i,k,b}$ are averaged over test sets, obtaining $\hat{f}_{i,b}$. The probability $P(\wedge_{j \neq i} f(A_i) \geq f(A_j))$ is then computed as $1/B \sum_b I(\hat{f}_{i,b} \geq \max_j \hat{f}_{j,b})$, where I is the indicator function.

B.2 Practical Considerations and Implementation Details for PFBP

In this section, we discuss several important details for an efficient and accurate implementation of PFBP. The main focus is on PFBP using conditional independence tests based on binary logistic regression, which is the test used in the experiments, although most details regard the general case or can be adapted to other conditional independence tests. We note that, everything described is not specific to PFBP, but can also be used by FBED^K and other feature selection algorithms.

B.2.1 Accurate Combination of Local p-values Using Fisher's method

In order to apply Fisher's combined probability test, *the data distributions of each data block should be the same for the test to be valid*. There should be no systematic bias on the data or the combining process may exacerbate this bias (see [151]). Such bias may occur if blocks contain data from the same departments, stores, or branches, or in consecutive time moments and there is time-drift on the data distribution. This problem is easily avoided if before the analysis the partitioning of samples to blocks is done randomly, as done by PFBP.

Another important detail to observe in practice, is to *directly compute the logarithm of the p-values for each conditional independence test* instead of first computing the p-value and then taking the logarithm. As p-values tend to get smaller with larger sample sizes (in case the null hypothesis does not hold), they quickly reach the machine epsilon, and will be rounded to zero. If this happens, then sorting and selecting features according to p-values breaks down and PFBP will select an arbitrary feature. This behavior is further magnified in case of combined p-values, as a single zero local p-value leads to a zero combined p-value no matter the values of the remaining p-values. The R language provides the option to directly compute the logarithm of the p-value (using the option $log.p = T$). Next, we give pointers to our implementation, since there was no other implementation available for Spark. For the χ^2 distribution, the p-value can be computed as $\frac{\Gamma(k/2, x/2)}{\Gamma(k/2)}$, where x is the test statistic, k the degrees of freedom, $\Gamma(\cdot,\cdot)$ the incomplete gamma function and $\Gamma(\cdot)$ the gamma function. Formulas for computing the incomplete gamma function can be found in [28]. For completeness, we provide them next. Equation 2.27 in [28] for $k/2 = n + 1/2, n = 0, 1, 2, \ldots$ is

$$\Gamma(n + \frac{1}{2}, x) = \Gamma(n + \frac{1}{2})\{\text{erfc}(\sqrt{x}) + e^{-x} \sum_{j=0}^{n-1} \frac{x^{j+\frac{1}{2}}}{\Gamma(j + \frac{3}{2})}\}$$

where erfc is is the Gauss error function, while Corollary 2.1 in [28] is for positive integer values of $k/2 = n, n = 0, 1, 2, \ldots$

$$\Gamma(n + 1, x) = n\Gamma(n, x) + x^n e^{-x}$$

To compute the logarithm of the p-value, one has to take compute the logarithm of the above formulas. As the formulas contain sums, one has to be careful when computing the logarithm. What is most important is to avoid computing exponents and values for Γ directly, as they may quickly lead to numerical overflows. Instead, one can compute them more accurately by working with logarithms, and then taking the exponent. For instance, $\frac{x^{j+\frac{1}{2}}}{\Gamma(j+\frac{3}{2})}$ can be computed more accurately as follows:

$$\frac{x^{j+\frac{1}{2}}}{\Gamma(j + \frac{3}{2})} = e^{\log \frac{x^{j+\frac{1}{2}}}{\Gamma(j+\frac{3}{2})}} = e^{\log x^{j+\frac{1}{2}} - \log \Gamma(j+\frac{3}{2})} = e^{j+\frac{1}{2} + \log x - \log \Gamma(j+\frac{3}{2})}$$

A function for accurately computing the logarithm of the Γ function exist in most math libraries and languages. Furthermore, another important observation is that low values are usually outweighed by very large ones (which may be several tens of orders of magnitude larger), and thus will not be important in the final computation. Thus, the logarithm of a sum can be computed as $\log(a+b) \sim \log(a)$ if $a >> b$. Using the above

observations, the logarithm of the p-value can be computed with very high accuracy (even $10^{-1000000}$).

B.2.2 Implementation of the Conditional Independence Test using Logistic Regression for Binary Targets

The conditional independence test is the basic building block of PFBP, and thus using a fast and robust implementation is essential. Next, we briefly review optimization algorithms used for maximum likelihood estimation, mainly focusing on binary logistic regression, and in the context of feature selection using likelihood-ratio tests.

A comprehensive introduction and comparison of algorithms for fitting (i.e., finding the β that maximizes the likelihood) binary logistic regression models is provided in [109]. Three important classes of optimization algorithms are Newton's method, conjugate gradient descent and quasi-Newton methods. Out of those, Newton's method is the most accurate and typically converges to the optimal solution in a few tens of iterations. The main drawback is that each such iteration is slow, requiring $O(n \cdot d^2)$ computations, where n is the sample size and d the number of features. Conjugate gradient descent and quasi-Newton methods on the other hand require $O(n \cdot d)$ and $O(n \cdot d + d^2)$ time per iteration, but may take much longer to converge. Unfortunately, there are cases were those methods fail to converge to an optimal solution even after hundreds of iterations. This not only affects the accuracy of feature selection, but also leads to unpredictable running times. Most statistical packages include one or multiple implementations of logistic regression. Such implementations typically use algorithms that can handle thousands of predictors, with quasi-Newton methods being a popular choice. For feature selection however, one is typically interested to select a few tens or hundreds of variables. In anecdotal experiments, we found that for this case Newton's method is usually faster and more accurate, especially with fewer than 100-200 variables. Because of that, and because of the issues mentioned above, we used a fine-tuned, custom implementation of Newton's method.

There are some additional, important details. First of all, there are cases where the Hessian is not invertible [1]. If this the case, we switch to conjugate gradient descent using the fixed Hessian as a search direction for that iteration, as described in [109]. Finally, as a last resort, in case the fixed Hessian is not invertible we switch to simple gradient descent. Next, for all optimization methods there are cases in which the computed step-size has to be adjusted to avoid divergence, whether it is due to violations of assumptions or numerical issues. One way to do this is to use inexact

[1] One case where this happens is if the covariance matrix of the input data is singular, or close to singular. Note that, due to the nature of the feature selection method which considers one variable at a time, this can happen only if the newly added variable is (almost) collinear with some of the previously selected variables. If this is the case, the variable would not be selected anyway.

line-search methods, such as backtracking-Armijo line search [7], which was used in our implementation.

B.2.3 Score Tests for the Univariate Case

In the first step of forward selection where no variable has been selected, one can use a score test (also known as Lagrange multiplier test) instead of a likelihood-ratio test to quickly compute the p-value without having to actually fit logistic regression models. The statistic of the Score test equals [74]

$$\text{Statistic} \equiv \frac{\sum_{j=1}^{n} X_j (T_j - \bar{T})}{\sqrt{\bar{T}(1-\bar{T}) \sum_{j=1}^{n} (X_j - \bar{X})^2}}$$

where n is the number of samples, T is the binary outcome variable (using a 0/1 encoding), and X is the variable tested for independence. Note that, such tests can also be derived for models other than binary logistic regression, but it is out of the scope of the thesis. The score test is asymptotically equivalent to the likelihood ratio test, and in anecdotal experiments we found that a few hundred samples are sufficient to get basically identical results, justifying its use in Big Data settings. Using this in place of the likelihood ratio test reduces the time of the univariate step significantly and is important for an efficient implementation, as the first step is usually the most computationally demanding one in the PFBP algorithm, as a large portion of the variables will be dropped by the Early Dropping heuristic.

Appendix C
Dataset Generation and Preparation

C.1 Simulating Data from Bayesian Networks

To generate data with complex correlation structures, we chose to generate data from Bayesian networks. This is done in three steps: (a) generate a Bayesian network structure $\mathcal{G}$ with N nodes (variables) and M edges, (b) sample the parameters of $\mathcal{G}$, and (c) sample instances from $\mathcal{G}$. We will next describe the procedures used for each step.

C.1.1 Generating the Bayesian network structure

First, we need to specify the number of nodes N and the connectivity C between those nodes, which implicitly corresponds to some number of edges M. The connectivity parameter C corresponds to the (average) degree of each variable. Using the connectivity instead of setting the number of edges allows one to easily control the complexity of the network, as C directly corresponds to the average number of parents and children of each node. We proceed by showing how the edges were sampled. Let $F_1, \ldots, F_N$ be all nodes of $\mathcal{G}$, listed in topological order. To sample the edges of the network we iterate over all pairs of variables F_i and F_j (i ¡ j), and add an edge from F_i to F_j with probability $C/(N-1)$, ensuring acyclicity of the resulting graph. It can be easily shown that this will result in a network with an average degree of C.

C.1.2 Generating the Bayesian network parameters

The first step is to pick the variable that corresponds to the outcome variable T. We chose to use the node at position $\lceil N/2 \rceil$, as this node has the same number of parents and children on average. For our experiments, we chose T to be of binary type and all remaining variables to be of continuous type, but in principle everything stated can be easily adapted to other variable types. In Bayesian networks, the each variable F is

a function of its parents $\mathbf{Pa}(F)$. The functional form for continuous variables F_i is

$$F_i = \beta_0 + \sum_{F_j \in \mathbf{Pa}(F_i)} \beta_j F_j + \epsilon_i$$

where β_0 is the intercept, β_j is the coefficient of the j-th parent of F_i and ϵ_i is its error term. In our case, we set the intercept to 0, as it does not affect the correlation structure. The coefficients β_j are sampled uniformly at random from $[-1, -0.1] \cup [0.1, 1]$ to avoid coefficients which are close to 0. The error term ϵ_i follows a normal distribution with 0 mean and σ_i^2 variance, which was set to the default value of 1 in our experiments, unless stated otherwise. Note that all variables are normally distributed as they are sums of normally distributed variables. For each variable F_i the mean equals zero and the variance equals $\sigma_i^2 + \sum_{F_j \in \mathbf{Pa}(F_i)} \beta_j^2$. The fact that the variance increases may lead to numerical instabilities in practice, especially when generating large networks. Because of that, we standardize each variable to have unit variance by dividing it with its standard deviation, which is the square root of the variance as described above. For the target T, its log-odds ratio is again a linear function of its parents, defined as

$$\log\left(\frac{P(T=1)}{1-P(T=1)}\right) = \beta_0 + \sum_{F_j \in \mathbf{Pa}(T)} \beta_j F_j + \epsilon_T$$

As before, the log-odds ratio is standardized to have unit variance.

The value of T is set to 1 whenever the log-odds ratio is larger than some threshold t, and to 0 otherwise. Setting t to 0 results in a 50/50 class distribution of T. Other class distributions p_0/p_1 can be obtained by simply setting t to $N_{0,1}^{-1}(1-p_0)$, where $N_{0,1}^{-1}$ is the standard normal inverse cumulative distribution function. As a final note, the standardization method used above only guarantees that variables that come before T in the topological ordering are standard normal variables. As T is not normally distributed (nor does it have unit variance), all variables that are direct or indirect functions of T are not exactly normally distributed. However, as this neither alters the correctness of the data generation method, nor leads to any other issues, we leave it as is.

C.1.3 Sampling data from the generated Bayesian network

To generate a sample, one has to traverse the network in topological order and to compute the value of each variable separately, using the formulas described previously. By construction the network is already in topological order, which is simply given by the index of each variable. To compute the value of a variable one has to compute the sum of its parents (if it has any parents), and to add the error term, which is drawn

from a normal distribution.

C.2 SNP Data Generation

To generate the SNP dataset we followed the procedure described in [25]. We used the HAPGEN 2 software [27] with the Hapmap 2 (release 22) CEU population [33] to simulate 500000 individuals (samples). This population contains 2543887 SNPs, but only 592555 were kept, by filtering out the ones not available in the Illumina Human OmniExpress-12 v1.1 BeadChip [1]. The final dataset contains 500000 samples and 592555 SNPs. Each variable takes values in $\{0, 1, 2\}$, which correspond to the number of reference alleles. Thus, the dataset is dense, and requires approximately 2.16 TB memory (stored as double precision floats). Naturally, fewer bytes can be used to store SNP data as each variable only takes 3 values, but this would require a specialized implementation.

C.2.1 Phenotype Simulation

Let s_{ij} be the i-th value of the j-th SNP s_j, and p_j be the reference allele frequency of SNP j, that is, p_j is the average value of s_{ij} divided by 2. The standardized value of s_{ij}, z_{ij} is defined as

$$z_{ij} = (s_{ij} - \mu_j)/\sigma_j$$

where $\mu_j = 2p_j$ and $\sigma_j = \sqrt{2p_j(1 - p_j)}$.

The phenotype (outcome) y follows an additive genetic model

$$y_i = g_i + e_i = \sum_{j=1}^{M} z_{ij} u_j + e_i$$

where y_i is the i-th value of y, g_i is the genetic effect, e_i is the noise term, M is the number of variables influencing y, and u_j is the effect (coefficient) of z_j. The coefficients z_j were sampled from a normal distribution with zero mean and unit variance. The error terms e_i follow a normal distribution with zero mean and variance $\sigma_i^2(1 - h^2)/h^2$, where σ_i^2 is the variance of g_i and h^2 corresponds to the trait heritability. Naturally, the larger h^2, the more y depends on the SNPs. In our case, we chose $M = 100$ and set $h^2 = 0.7$, one of the values used in [25]. Finally, to obtain a binary outcome, we set the value of y_i to 1 if it is positive, and to 0 otherwise, resulting in an approximately balanced outcome.

[1] `https://support.illumina.com/array/array_kits/humanomniexpress-12-beadchip-kit.html`

Table C.1: Binary classification and regression datasets used in the experimental evaluation. n is the number of samples, p the number of variables, type is the type of variables

Dataset	n	p	Type	Outcome	Domain	Source
Ada	4562	46	Real	Binary	Census Data	WCCI 2006 Challenge [62]
Musk (v2)	6598	166	Real	Binary	Musk Activity Prediction	UCI ML Repository [41]
Sylva	14394	216	Mixed	Binary	Forest Cover Types	WCCI 2006 Challenge [62]
Madelon	2600	500	Integer	Binary	Artificial	NIPS 2003 Challenge [64]
Gina	3568	970	Real	Binary	Handwritten Digit Recognition	WCCI 2006 Challenge [62]
Communities & Crime Violent	1994	102	Real	Real	Crime Prediction	UCI ML Repository [122]
Communities & Crime Non-violent	2118	102	Real	Real	Crime Prediction	UCI ML Repository [122]
BlogData	60021	276	Mixed	Real	#Comments on Blog Posts	UCI ML Repository [24]
CT Slice	53500	379	Real	Real	CT Slice Localization	UCI ML Repository [59]
UJI Indoor Loc Latitude	21048	520	Mixed	Real	Indoor Localization	UCI ML Repository [141]
UJI Indoor Loc Longitude	21048	520	Mixed	Real	Indoor Localization	UCI ML Repository [141]

C.3 Dataset Collection and Preparation for the Evaluation of TMFBS

Table C.1 shows the list of all datasets, containing information about their number of samples n and variables p, the type of variables and outcomes, domain and sources [2]. For all datasets, we performed the following pre-processing steps: (a) we removed all constant variables as they do not provide any information about the outcome, (b) we removed variables related to sample ids and other types of metadata, and (c) we removed samples containing missing values. Missing values were only present in a few datasets, and only for a few samples. In a real-world application, this should be handled appropriately to avoid biased results, using either missing value imputation or predictive algorithms that can handle missingness. However, for our purposes it would only unnecessarily complicate the tuning procedure and is not that important

[2]The communities & crime data were originally collected from: (a) U. S. Department of Commerce, Bureau of the Census, Census Of Population And Housing 1990 United States: Summary Tape File 1a & 3a (Computer Files), (b) U.S. Department Of Commerce, Bureau Of The Census Producer, Washington, DC and Inter-university Consortium for Political and Social Research Ann Arbor, Michigan. (1992), (c) U.S. Department of Justice, Bureau of Justice Statistics, Law Enforcement Management And Administrative Statistics (Computer File) U.S. Department Of Commerce, Bureau Of The Census Producer, Washington, DC and Inter-university Consortium for Political and Social Research Ann Arbor, Michigan. (1992), and (d) U.S. Department of Justice, Federal Bureau of Investigation, Crime in the United States (Computer File) (1995).

given the rare presence of missing values.

The communities & crime data contain a total of 18 continuous outcome variables. Out of those, 8 measure the total number of committed crimes, 8 measure the number of crimes per 100K population, and 2 measure the number of violent and non-violent crimes per 100K population (by appropriately summing over the other outcomes). We chose to create two datasets, one measuring the violent and one the non-violent crimes. Note that, the number of samples differs, as there were missing values in the outcome variables.

The UJIIndoorLoc datasets also contain multiple outcome values, such as the location (longitude and latitude), the floor of the building, the building ID and others. Again, we chose to create two datasets, one using the latitude outcome and one for the longitude.

The CT Slice data consist of samples obtained from CT scans on 97 patients. In this case, the training/validation/test splits where performed on the patients, and not directly on the samples (i.e., stratified on the patient id). This was done to avoid having data from the same patients both on the training and on the held-out data.

Appendix D
Additional Results

D.1 Running Times of FBED^K, FBS, MMPC and LASSO

Table D.1 shows the running time of each feature selection algorithm and configuration, on all datasets. The values correspond to a single run on the complete dataset. All runs were performed on a single machine, and no runs were performed simultaneously. For FBED^K we only show running times for $K \in \{0, 1, 3, \infty\}$, and for LASSO-FS we show results for $\lambda_{max} \in \{25, 100, 500\}$. MMPC for a given value of $maxK$ is denoted as MMPC^{maxK}.

It can clearly be seen that LASSO-FS is the fastest in large datasets, irrespective of the number of λ values used. For smaller datasets (musk, sylva, madelon, secom, gina and hiva), FBED^0 and FBED^1 are often at least as fast as LASSO-FS. FBS and MMPC with $maxK \geq 3$ are the slowest among all algorithms. For the gisette and nova datasets, MMPC with $maxK \geq 3$ fails to terminate after a timeout limit of 2 days. For $maxK \leq 2$ MMPC often has competitive performance with the other algorithms. We note that MMPC was designed specifically for low sample sizes and high dimensional data, such as data from biological domains which contain a few tens or hundreds of samples and tens of thousands of variables, explaining the high running times on the datasets considered in our experiments, most of which contain thousands of samples.

The large difference between the running time of LASSO-FS and the other algorithms can largely be attributed to their implementations. For LASSO-FS the glmnet implementation was used, which is highly optimized and written in FORTRAN. In contrast, for FBED^K, FBS and MMPC we used a custom logistic regression implementation written in Matlab. A difference of 1-2 orders of magnitude can be expected between the same implementation in a low-level language such as FORTRAN, C or C++ and higher-level languages such as Matlab. Therefore, we would expect that an implementation in a lower-level language would perform similarly to LASSO-FS. Of course, LASSO-FS has the advantage that it returns the whole solution path, and thus would still be faster in practice if hyper-parameter optimization is also performed.

Table D.1: Running times in seconds on the full datasets.

	Algorithm	musk	sylva	madelon	secom	gina	hiva	gisette	p53	arcene	nova	dexter	dorothea
$\alpha = 0.001$	FBED^0	1.5	3.3	0.3	0.3	5.0	2.8	56.7	48.6	2.6	21.8	6.2	133.9
	FBED^1	3.2	7.6	0.7	0.3	11.3	2.8	186.5	137.8	6.0	90.6	20.1	377.9
	FBED^3	9.1	7.6	0.7	0.3	24.8	2.8	546.3	242.3	14.5	308.0	42.8	1033.0
	FBED$^\infty$	22.1	7.6	0.7	0.3	24.8	2.8	2962.7	242.3	14.5	536.1	42.8	1855.3
	FBS	46.0	36.3	1.3	1.4	140.4	22.4	5773.0	1186.4	66.8	3486.3	309.2	7233.3
	MMPC^1	3.9	9.5	0.3	0.5	21.2	5.0	309.2	115.0	4.1	112.9	14.9	91.8
	MMPC^2	7.9	17.9	0.3	0.5	84.0	5.4	1885.4	163.4	4.1	1524.9	17.9	92.0
	MMPC^3	7.9	20.9	0.3	0.6	171.2	5.6	N/A	175.9	4.1	N/A	19.9	92.0
	MMPC^4	9.8	21.8	0.3	0.5	292.4	5.6	N/A	210.7	4.1	N/A	21.3	93.1
$\alpha = 0.005$	FBED^0	1.8	3.9	0.2	0.3	6.7	3.3	79.2	60.4	2.8	31.3	6.5	139.9
	FBED^1	5.4	8.3	0.6	0.8	14.2	10.7	244.2	158.5	10.8	114.1	22.3	457.0
	FBED^3	11.7	12.9	0.6	1.6	36.2	28.1	756.1	418.0	19.9	435.7	77.9	865.6
	FBED$^\infty$	29.7	12.9	0.6	1.6	49.2	28.1	1105.3	700.5	19.9	1426.2	202.2	865.6
	FBS	82.2	50.4	2.1	3.1	192.7	82.7	10932.7	3864.6	66.8	7015.8	503.0	7233.3
	MMPC^1	4.6	10.4	0.3	0.6	25.8	6.0	384.6	142.2	4.7	238.3	17.4	106.3
	MMPC^2	10.5	24.5	0.3	0.6	126.0	6.4	2869.2	230.4	4.7	3993.7	30.2	107.1
	MMPC^3	10.2	43.6	0.3	0.6	307.1	6.9	N/A	330.3	4.7	N/A	36.6	115.0
	MMPC^4	11.5	45.0	0.3	0.6	470.4	6.9	N/A	469.6	4.7	N/A	54.2	126.6
$\alpha = 0.01$	FBED^0	2.0	4.3	0.2	0.3	8.3	3.6	88.0	64.2	3.0	39.0	6.8	144.7
	FBED^1	5.3	8.9	0.2	0.8	16.8	11.9	261.1	164.3	11.8	123.3	25.2	463.0
	FBED^3	16.5	14.1	0.2	1.7	46.0	32.3	824.3	450.7	11.8	505.8	95.4	884.6
	FBED$^\infty$	32.5	14.1	0.2	2.1	97.6	81.8	824.3	1615.0	11.8	760.0	136.7	884.6
	FBS	94.3	61.0	2.5	4.4	234.8	150.2	10932.7	5160.2	66.8	8811.8	503.0	7233.3
	MMPC^1	5.0	12.8	0.3	0.6	28.0	7.0	426.9	161.3	5.1	347.0	19.6	119.1
	MMPC^2	12.4	28.7	0.3	0.6	141.0	7.8	3570.4	269.8	5.3	8262.0	41.6	128.8
	MMPC^3	12.7	45.1	0.4	0.6	426.0	9.1	N/A	460.1	5.3	N/A	63.5	146.0
	MMPC^4	13.3	62.0	0.4	0.7	573.4	9.8	N/A	728.7	5.3	N/A	104.6	200.6
$\alpha = 0.05$	FBED^0	3.3	5.6	0.4	0.5	14.2	6.4	140.2	95.2	3.9	111.9	9.5	201.0
	FBED^1	6.4	11.4	0.9	1.1	34.9	17.0	398.6	242.8	3.9	380.7	42.0	644.6
	FBED^3	25.6	24.8	2.0	2.7	82.8	50.5	398.6	779.5	3.9	957.2	42.0	644.6
	FBED$^\infty$	76.1	49.9	4.6	7.7	221.6	206.5	398.6	7688.4	3.9	957.2	42.0	644.6
	FBS	191.9	108.6	18.6	15.4	1243.4	3490.9	10932.7	31681.6	66.8	10119.3	621.0	7233.3
	MMPC^1	6.4	15.0	0.6	0.8	38.3	16.6	604.2	266.6	7.9	1050.6	48.9	406.0
	MMPC^2	15.8	48.1	1.5	1.1	222.2	22.4	6279.9	566.7	10.2	45093.2	204.1	750.2
	MMPC^3	22.0	152.2	3.3	1.1	921.7	46.3	N/A	977.6	12.7	N/A	525.3	1499.5
	MMPC^4	23.3	209.4	5.6	1.3	1499.4	68.8	N/A	2103.9	12.2	N/A	624.8	3818.6
$\alpha = 0.1$	FBED^0	4.1	7.6	0.8	0.7	20.1	11.0	213.0	128.4	4.7	162.3	14.1	246.7
	FBED^1	8.9	17.3	1.8	1.9	49.0	37.2	858.6	359.8	4.7	428.5	53.7	246.7
	FBED^3	26.8	40.0	4.5	4.0	127.5	289.0	858.6	1280.7	4.7	980.2	53.7	246.7
	FBED$^\infty$	84.6	64.9	7.5	40.3	889.1	347.7	858.6	3476.2	4.7	980.2	53.7	246.7
	FBS	253.7	246.7	48.7	29.0	3688.9	9997.9	10932.7	31681.6	66.8	10784.8	621.0	7669.2
	MMPC^1	7.0	16.9	1.2	1.3	44.2	28.1	733.7	388.2	13.3	2063.8	117.0	1033.3
	MMPC^2	17.7	96.9	7.3	2.6	278.0	68.4	8313.7	955.9	24.5	122716.3	715.0	3873.2
	MMPC^3	25.2	187.9	52.4	4.2	1315.8	160.0	N/A	1675.6	29.5	N/A	2065.3	10196.9
	MMPC^4	33.6	371.5	236.8	6.3	2959.9	405.1	N/A	4359.6	39.5	N/A	3761.7	24259.0
	LASSO-FS^{25}	12.2	7.0	2.7	6.0	12.3	13.7	6.9	118.6	0.2	1.6	0.4	4.0
	LASSO-FS^{100}	12.2	11.3	4.1	7.3	16.2	16.3	8.9	65.0	0.3	3.3	0.8	11.0
	LASSO-FS^{500}	10.5	19.7	4.9	11.2	22.5	34.1	24.5	133.4	1.0	14.7	3.5	51.1

D.2 Accuracy of p-value Combination using Meta-Analysis and Evaluation of the STD Rule

We evaluated the ability of the proposed p-value computation method (combination of local p-values using Fisher's combined probability test) in identifying the same variable

to select as when global p-values are used. We performed a computational experiment on simulated data to investigate the effect of the total sample size and number of data Blocks on the accuracy of the proposed approach. Furthermore, we compare the STD and EPV rules for setting the minimum number of samples in each Data Block. The EPV rule computes the sample size per Sample Group as $s = df \cdot c / \min(p_0, p_1)$, while STD uses $s = df \cdot c / \sqrt{p_0 \cdot p_1}$, where df is set to the maximum number of degrees of freedom (see 5.3.1 for more details), c is a positive constant (which may take different values for each rule), and p_0 and p_1 are the frequencies of the negative and positive class respectively.

D.2.1 Data Generation

To generate data with complex correlation structures, we chose to generate data from simulated Bayesian networks. All variables are continuous Gaussian and are linear functions of their parents. The target variable is binary, and the log-odds ratio is a linear function of its parents. The procedure is described in detail in Appendix C.1.

We used the following parameters to generate Bayesian networks and data from those networks: (a) the number of variables was set to 101 (100 variables plus the outcome T), (b) the connectivity parameter was set to 10 (i.e., the average degree of each node), (c) the frequency of the most frequent class of T was set to $\{50\%, 60\%, 70\%, 80\%, 90\%\}$ and (d) the standard deviation of the error terms was set to $\{0.01, 0.1, 1\}$. In total this results in 15 possible Bayesian network configurations. Note that, the connectivity is relatively high and the standard deviation of the error terms is relatively low so that all variables are highly correlated, increasing the difficulty of the problem of selecting the best variable. For each such parameter combination we generated 250 Bayesian networks, resulting in a total of $250 \times 15 = 3750$ networks. Next, we generated datasets with different sample sizes, by varying the sample size from $10^{2.5}$ to 10^4 in increments of 0.1 of the exponent, leading to 16 different sample sizes. Overall this resulted in 60000 datasets.

D.2.2 Simulation Results: Combined p-values vs Global p-values

We performed conditional independence tests on all generated datasets to simulate a forward Iteration using p-values from global tests (i.e., using all data) and from combined p-values using Fisher's method. We varied the following parameters: (a) the number of conditioning variables, which was set to 0, 1, 2 or 3, and (b) the number of Sample Sets each dataset was split to, which ranged from 1 (no split) to 25 with increments of 1 (a total of 25 cases). This allows us to investigate the effect of the total number of combined local p-values. The simulation of a forward Iteration was performed for each dataset and conditioning size k as follows: (a) k variables were

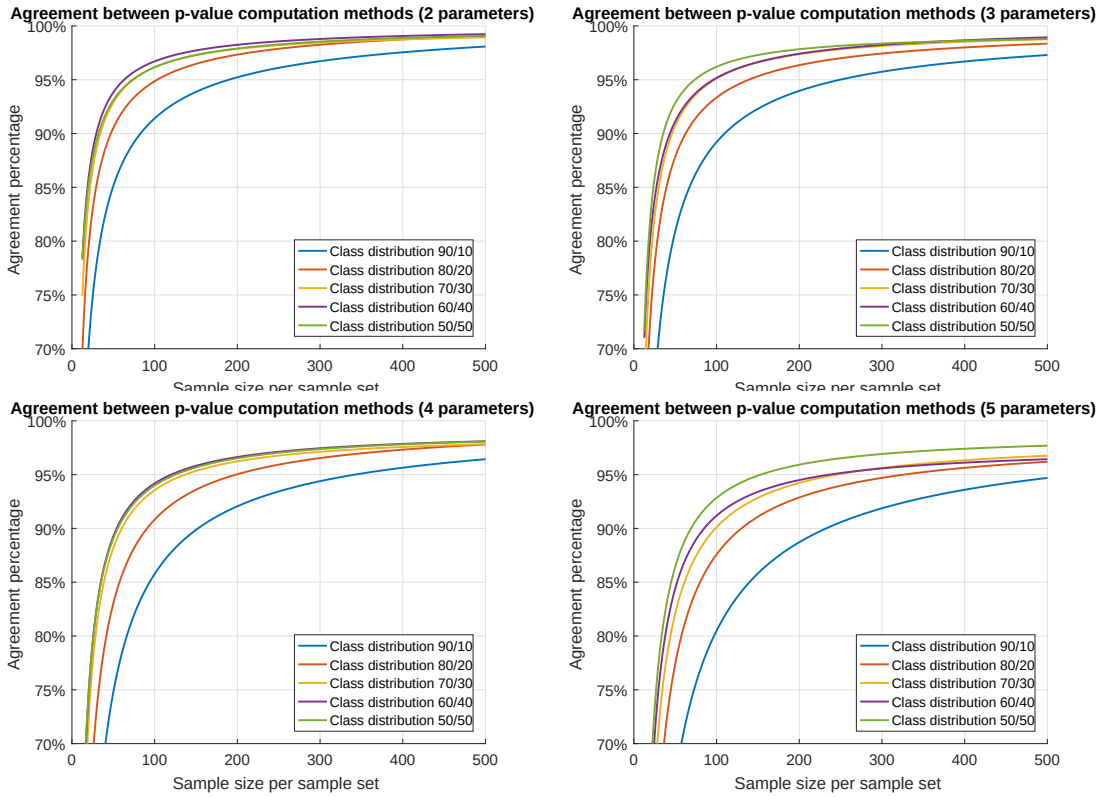


Figure D.1: The percentage of agreement is shown, which corresponds to how often combining local p-values and computing the p-value on all samples leads to the same decision. The y-axis shows how the sample size per sample set affects the agreement percentage. Both methods tend to agree asymptotically for various class distributions and conditioning set sizes.

randomly selected from the Markov blanket of T (simulating that k variables have already been selected), (b) the global conditional independence test was performed between T and the remaining variables over all samples (i.e. number of sample sets equals 1), (c) the same test was performed on all Sample Sets resulting by splitting the data randomly to m equally-sized sample sets (m ranging from 2 to 25) and combining the p-values using Fisher's combined probability test.

We compute the percentage of agreement between both methods, that is, how often both methods select the same variable. This is computed as the proportion of times both methods agreed on the 250 repetitions, leading to one value for each of the 15 Bayesian network configurations, each sample size, conditioning set size and number of Sample Sets. Thus, in total we have $15 \times 16 \times 4 \times 24 = 23040$ such values. The results are summarized in Figure D.1. There are 4 figures, one for each different conditioning size, and each figure contains 5 curves, one for each class distribution of

Table D.2: Median value of c to obtain an agreement percentage between 85% and 95%. p_{max} corresponds to the proportion of the most frequent class, while df is the degrees of freedom in the largest model. The relative differences for the STD rule are smaller (less than 2 against over 2.5 for the EPV rule), suggesting it is more appropriate. A minimum value of $c = 10$ with the proposed rule is recommended and used in the experiments.

	EPV Rule				STD Rule			
p_{max} | df	2	3	4	5	2	3	4	5
0.5	11.2	7.8	9.0	9.7	11.2	7.8	9.0	9.7
0.6	7.7	7.6	7.4	11.2	9.4	9.3	9.1	13.7
0.7	6.6	5.7	5.9	8.6	10.1	8.8	9.1	13.2
0.8	5.7	5.2	5.7	6.7	11.5	10.5	11.4	13.3
0.9	5.0	4.4	4.2	4.4	14.9	13.2	12.5	13.3

T. Each such curve summarizes the results over all error variances, sample sizes and number of Sample Sets (that is, $3 \times 16 \times 24 = 1152$ points). Note that the number of parameters of the largest model is always the conditioning size plus 2, as the model also includes the variable that is tested for (conditional) independence with T and the intercept. The x-axis shows the sample size per Sample Set, which is computed as the sample size divided by the number of Sample Sets. We only show the results up to 500 samples per Sample Set; the agreement percentage approaches 100% in all cases with increasing sample size, reaching at least 99% with 5000 samples per Sample Set. The y-axis shows how often both methods lead to the same decision. To avoid cluttering, we computed the curves by fitting a power regression model $y = a \cdot x^{\beta} + c$. We found that this model is appropriate, as it has R^2 values between 0.75 and 0.95. We conclude the following: *(a) both approaches tend to make the same decision with increasing sample size, (b) the sample size per Sample Set required depends on the number of parameters and the class distribution, and increases with increasing number of parameters and class imbalance.*

D.2.3 Simulation Results: STD vs EPV for Determining the Required Sample Size

We propose an alternative rule to EPV, which is computed as $df \cdot c / \sqrt{p_0 \cdot p_1}$. The denominator is the standard deviation of the class distribution, which follows a Bernoulli distribution. We call this the STD rule hereafter. For balanced class distributions the result is identical to the EPV rule, while for skewed distributions the value is always smaller.

To validate the STD rule, we used the results of the previous simulation experiment and computed the value of c by solving the equation for c and substituting in the values of the class distributions, degrees of freedom and sample size per Sample Set. We kept

the values of c that correspond to an agreement percentage between 85% and 95% (focusing on an interesting range of high agreement between p-value computation methods), and computed their median value for each class distribution, conditioning size k and for both rules. Ideally, one would expect c to be constant across rows (class distribution) and columns (conditioning size). A constant value of c for a rule means that the rule can exactly compute the required sample size to get an agreement percentage around 90%. Furthermore, we note that the values of c are not comparable between rules, and thus their exact values are not important; what matters is the relative difference between values of c for the same rule.

The results are shown in Table D.2. Although the value of c varies across class distributions and degrees of freedom, we can see that the relative differences are smaller the STD rule. Specifically, for EPV c ranges from 4.2 to 11.2, the latter being over 2.5 times larger, while for STD it ranges from 7.8 to 14.9, which is less than 2 times larger. This suggests that the STD rule performs better than EPV across various conditioning set sizes and class distributions, at least on the experiments considered here. Furthermore, the results suggest that a value of at least $c = 10$ should be used for STD to get reasonably accurate results. We note that, in practice this value is much higher in most cases for PFBP, as it partitions the samples initially by considering the worst case scenario (i.e., selecting *maxVars* variables). Thus especially in early Iterations, which are the most crucial ones, PFBP will typically have a sufficient number of samples even with $c = 10$ to select the best variables.

Appendix E
Publications

Publications on Feature Selection

The research related to this thesis has lead to the following publications.

(1) Giorgos Borboudakis, Ioannis Tsamardinos. *Forward-Backward Selection with Early Dropping*. Journal of Machine Learning Research, 2018. (third revision)

(2) Ioannis Tsamardinos, Giorgos Borboudakis, Pavlos Katsogridakis, Polyvios Pratikakis, Vassilis Christophides. *A Greedy Feature Selection Algorithm for Hybridly Partitioned Big Data*. Machine Learning, 2018 [148].

(3) Giorgos Borboudakis, Ioannis Tsamardinos. *Extending Greedy Feature Selection Algorithms to Multiple Solutions*. Data Mining and Knowledge Discovery, 2018. (to be submitted)

The contributions are described in detail in Chapters 4, 5 and 6 respectively.

Other Publications

In addition to the above, which are directly related to the main topic of this thesis, we have also published papers in the fields of causal discovery, general machine learning and applications, and signal processing. A list of publications follows.

Causal Discovery

(4) Michail Tsagris, Giorgos Borboudakis, Vincenzo Lagani, Ioannis Tsamardinos. *Constraint-based Causal Discovery with Mixed Data*. International Journal of Data Science and Analytics, 2018. [142].

Abstract: We address the problem of constraint-based causal discovery with mixed data types, such as (but not limited to) continuous, binary, multinomial, and ordinal variables. We use likelihood-ratio tests based on appropriate regression models and show how to derive symmetric conditional independence tests. Such tests can then be directly used by existing constraint-based methods with mixed data, such as the

PC and FCI algorithms for learning Bayesian networks and maximal ancestral graphs, respectively. In experiments on simulated Bayesian networks, we employ the PC algorithm with different conditional independence tests for mixed data and show that the proposed approach outperforms alternatives in terms of learning accuracy.

(5) Anna Roumpelaki, Giorgos Borboudakis, Sofia Triantafillou, Ioannis Tsamardinos. *Marginal Consistency of Constraint-Based Causal Learning*. Causation: Foundation to Application Workshop at UAI, 2016 [128].

Abstract: Maximal Ancestral Graphs (MAGs) are probabilistic graphical models that can model the distribution and causal properties of a set of variables in the presence of latent confounders. They are closed under marginalization. Invariant pairwise features of a class of Markov equivalent MAGs can be learnt from observational data sets using the FCI algorithm and its variations (such as conservative FCI and order independent FCI). We investigate the consistency of causal features (causal ancestry relations) obtained by FCI in different marginals of a single data set. In principle, the causal relationships identified by FCI on a data set D measuring a set of variables V should not conflict the output of FCI on marginal data sets including only subsets of V. In practice, however, FCI is prone to error propagation, and running FCI in different marginals results in inconsistent causal predictions. We introduce the term of marginal causal consistency to denote the consistency of causal relationships when learning marginal distributions, and investigate the marginal causal consistency of different FCI variations. Results indicate that marginal causal consistency varies for different algorithms, and is also sensitive to network density and marginal size.

(6) Giorgos Borboudakis, Ioannis Tsamardinos. *Towards Robust Causal Discovery for Business Applications*. Conference on Knowledge Discovery and Data Mining (KDD), 2016 [18].

Abstract: Causal discovery algorithms can induce some of the causal relations from the data, commonly in the form of a causal network such as a causal Bayesian network. Arguably however, all such algorithms lack far behind what is necessary for a true business application. We develop an initial version of a new, general causal discovery algorithm called ETIO with many features suitable for business applications. These include (a) ability to accept prior causal knowledge (e.g., taking senior driving courses improves driving skills), (b) admitting the presence of latent confounding factors, (c) admitting the possibility of (a certain type of) selection bias in the data (e.g., clients sampled mostly from a given region), (d) ability to analyze data with missing-by-design (i.e., not planned to measure) values (e.g., if two companies merge and their databases measure different attributes), and (e) ability to analyze data from different interventions (e.g., prior and posterior to an advertisement campaign). ETIO is an instance of the logical approach to integrative causal discovery that has been relatively

recently introduced and enables the solution of complex reverse-engineering problems in causal discovery. ETIO is compared against the state-of-the-art and is shown to be more effective in terms of speed, with only a slight degradation in terms of learning accuracy, while incorporating all the features above

(7) Giorgos Borboudakis, Ioannis Tsamardinos. *Bayesian Network Learning with Discrete Case-Control Data*. Conference on Uncertainty in Artificial Intelligence (UAI), 2015 [17].

Abstract: We address the problem of learning Bayesian networks from discrete, unmatched case- control data using specialized conditional independence tests. Those tests can also be used for learning other types of graphical models or for feature selection. We also propose a post- processing method that can be applied in conjunction with any Bayesian network learning algorithm. In simulations we show that our methods are able to deal with selection bias from case-control data.

General Machine Learning and Applications

(8) Ioannis Tsamardinos, Elissavet Greasidou, Giorgos Borboudakis. *Bootstrapping the Out-of-sample Predictions for Efficient and Accurate Cross-Validation*. Machine Learning, 2018 [150].

Abstract: Cross-Validation (CV), and out-of-sample performance-estimation protocols in general, are often employed both for (a) selecting the optimal combination of algorithms and values of hyper-parameters (called a configuration) for producing the final predictive model, and (b) estimating the predictive performance of the final model. However, the cross-validated performance of the best configuration is optimistically biased. We present an efficient bootstrap method that corrects for the bias, called Bootstrap Bias Corrected CV (BBC-CV). BBC-CV's main idea is to bootstrap the whole process of selecting the best-performing configuration on the out-of-sample predictions of each configuration, without additional training of models. In comparison to the alternatives, namely the nested cross-validation and a method by Tibshirani and Tibshirani , BBC-CV is computationally more efficient, has smaller variance and bias, and is applicable to any metric of performance (accuracy, AUC, concordance index, mean squared error). Subsequently, we employ again the idea of bootstrapping the out-of-sample predictions to speed up the CV process. Specifically, using a bootstrap-based statistical criterion we stop training of models on new folds of inferior (with high probability) configurations. We name the method Bootstrap Bias Corrected with Dropping CV (BBCD-CV) that is both efficient and provides accurate performance estimates.

(9) Giorgos Borboudakis, Taxiarchis Stergiannakos, Maria Frysali, Emmanuel Klontzas, Ioannis Tsamardinos, George E. Froudakis. *Chemically intuited, large-scale screen-*

ing of MOFs by machine learning techniques. Nature Computational Materials, 2017 [16].

Abstract: A novel computational methodology for large-scale screening of MOFs is applied to gas storage with the use of machine learning technologies. This approach is a promising trade-off between the accuracy of ab initio methods and the speed of classical approaches, strategically combined with chemical intuition. The results demonstrate that the chemical properties of MOFs are indeed predictable (stochastically, not deterministically) using machine learning methods and automated analysis protocols, with the accuracy of predictions increasing with sample size. Our initial results indicate that this methodology is promising to apply not only to gas storage in MOFs but in many other material science projects.

Signal Processing

(10) Anastasios Alexandridis, Giorgos Borboudakis, Athanasios Mouchtaris. *Addressing the Data-Association Problem for Multiple Sound Source Localization using DOA Estimates*. European Signal Processing Conference (EUSIPCO), 2015 [3].

Abstract: In this paper, we consider the data association problem that arises when localizing multiple sound sources using direction of arrival (DOA) estimates from multiple microphone arrays. In such a scenario, the association of the DOAs across the arrays that correspond to the same source is unknown and must be found for accurate localization. We present an association algorithm that finds the correct DOA association to the sources based on features extracted for each source that we propose. Our method results in high association and localization accuracy in scenarios with missed detections, reverberation, and noise and outperforms other recently proposed methods.