Bias Correction of the Cross-Validation Performance Estimate and Speed Up of its Execution Time

Elissavet Greasidou

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
University Campus, Voutes, Heraklion, GR-70013, Greece

Thesis Supervisor: Associate Professor *Ioannis Tsamardinos*

Heraklion, February 2017

This work was partially funded by the ERC Consolidator Grant No 617393 CAUSALPATH and the Toshiba project: "Feasibility study towards the Next Generation of statistical Text to Speech Synthesis System"

University of Crete
Computer Science Department

Bias Correction of the Cross-Validation Performance Estimate and Speed Up of its Execution Time

Thesis submitted by
Elissavet Greasidou
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Elissavet Greasidou

Committee approvals: _____
Ioannis Tsamardinos
Associate Professor, Thesis Supervisor

_____
Ioannis Tollis
Professor, Committee Member

_____
Ioannis Stylianou
Professor, Committee Member

Departmental approval: _____
Antonios Argyros
Professor, Director of Graduate Studies

Heraklion, February 2017

Abstract

Cross Validation (CV) is a de-facto standard in applied statistics and supervised machine learning both for model selection and assessment. The procedure is applied on a set of candidate configurations (i.e. a set of sequences of modelling steps with specified algorithms and their hyper-parameter values for each step) for model production, and the one with the best performance, according to a pre-specified criterion, is selected. However, the "best" performance achieved during CV is known to be an optimistically biased estimation of the generalization performance of the final model. To date, a relatively limited amount of research has been devoted to the correction of this bias, and all proposed methods either tend to over-correct or have limitations which can make their use impractical.

In this thesis, we propose a Bootstrap-based Bias Correction method (BBC) which works regardless of the data analysis task (e.g. classification, regression), or the structure of the models involved, and requires only a small computational overhead with respect to the basic CV procedure. BBC corrects the bias in a conservative way, providing an almost unbiased estimate of performance. Its 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 (NCV) [1], and a method by Tibshirani and Tibshirani (TT) [2], BBC is computationally more efficient, yields performance estimates competitive to those of NCV and is applicable to any CV procedure. Subsequently, we also employ the idea of bootstrapping the out-of-sample predictions in order to speed up the execution time of the CV procedure. Specifically, using a bootstrap-based hypothesis test we stop training of models on new folds of statistically-significantly inferior configurations. The Bootstrap-based Early Dropping (BED) method significantly reduces the computational time of CV with a negligible or no effect on performance. The two methods can be combined leading to the BED-BBC procedure that is both efficient and provides accurate estimates of performance.

Περίληψη

Η μέθοδος διασταυρωμένη επικύρωση (Cross Validation - CV) αποτελεί ένα ντεφάκτο πρότυπο στον τομέα της εφαρμοσμένης στατιστικής και εποπτευόμενης μηχανικής μάθησης (supervised machine learning) τόσο για την επιλογή ενός μοντέλου αλλά και την αξιολόγηση του. Η διαδικασία αυτή εφαρμόζεται σε ένα σύνολο υποψήφιων διαμορφώσεων (configurations) (δηλαδή, ένα σύνολο ακολουθιών βημάτων μοντελοποίησης με καθορισμένους αλγορίθμους και τιμές για τις ύπερ-παραμέτρους τους για κάθε βήμα) και εκείνη με την καλύτερη απόδοση, σύμφωνα με ένα προκαθορισμένο κριτήριο, επιλέγεται. Ωστόσο, η "καλύτερη" απόδοση που επιτυγχάνεται κατά τη διαδικασία του CV είναι γνωστό ότι είναι μία αισιόδοξα μεροληπτική (biased) εκτίμηση της γενίκευσης της απόδοσης του τελικού μοντέλου. Μέχρι σήμερα, ένα σχετικά περιορισμένο μέρος της έρευνας έχει αφιερωθεί στη διόρθωση αυτής της μεροληψίας (bias), και όλες οι προτεινόμενες μέθοδοι είτε έχουν την τάση να την διορθώνουν περισσότερο από όσο χρειάζεται ή έχουν περιορισμούς που μπορούν να κάνουν τη χρήση τους ανέφικτη.

Σε αυτή την εργασία, προτείνουμε μια μέθοδο διόρθωσης της μεροληψίας βασισμένη στην μέθοδο του bootstrap (Bootstrap-biased Bias Correction method - BBC) η οποία λειτουργεί ανεξάρτητα από την εργασία ανάλυσης δεδομένων (π.χ. ταξινόμηση, παλινδρόμηση), ή τη δομή των μοντέλων που εμπλέκονται, και απαιτεί μόνο μια μικρή υπολογιστική επιβάρυνση σε σχέση με τη βασική διαδικασία του CV. Η BBC μέθοδος διορθώνει την μεροληψία με συντηρητικό τρόπο παρέχοντας μια σχεδόν αμερόληπτη εκτίμηση της απόδοσης. Η βασική ιδέα είναι να εφαρμοστεί η μέθοδος του bootstrap σε όλη τη διαδικασία της επιλογής της καλύτερης μεθόδου στις εκτός εκπαιδευμένου δείγματος (out-of-sample) προβλέψεις της κάθε διαμόρφωσης (configuration), χωρίς πρόσθετη εκπαίδευση μοντέλων. Σε σύγκριση με τις εναλλακτικές μεθόδους, δηλαδή την εμφωλευμένη διασταυρωμένη επικύρωση (Nested Cross Validation - NCV) [1], και την μέθοδο των Tibshirani και Tibshirani (TT) [2], η BBC μέθοδος είναι υπολογιστικά πιο αποδοτική, είναι εφαρμόσιμη σε οποιαδήποτε διαδικασία CV, και η εκτίμηση της απόδοσης που παρέχει είναι ανταγωνιστική σε σχέση με εκείνη του NCV. Επίσης, χρησιμοποιούμε την ιδέα της εφαρμογής της bootstrap μεθόδου στις εκτός εκπαιδευμένου δείγματος (out-of-sample) προβλέψεις για την επιτάχυνση του χρόνου εκτέλεσης της CV διαδικασίας. Συγκεκριμένα, χρησιμοποιώντας ένα στατιστικό έλεγχο υποθέσεων (hypothesis test) βασισμένο στη μέθοδο του bootstrap, σταματάμε την εκπαίδευση μοντέλων σε καινούργια υποσύνολα των δεδομένων (folds) για στατιστικά-σημαντικά (statistically-signicantly) υποδεέστερες διαμορφώσεις (configurations). Η μέθοδος Bootstrap-based Early Dropping (BED) μειώνει σημαντικά τον υπολογιστικό χρόνο του CV με αμελητέα ή καμία επίδραση στην απόδοση. Οι δύο μέθοδοι μπορούν να συνδυαστούν οδηγώντας στην BED-BBC μέθοδο η οποία είναι αποδοτική και παρέχει ακριβείς εκτιμήσεις της απόδοσης.

Acknowledgements

First and foremost I would like to thank my supervisor, Ioannis Tsamardinos. He provided great support and advice throughout my programme.

I would also like to thank Giorgos Borboudakis, Michalis Tsagris and Pavlos Charonyktakis for their valuable comments, discussions and cooperation.

Special thanks go to the members of my dissertation committee, Ioannis Tollis and Ioannis Stylianou.

Most of all I would like to thank my parents Eleni and Manolis, and my sister Maria, for their continual love and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An important task in various areas of science (e.g. biology, finance, chemistry) and in industry is to find a systematic way of predicting a phenomenon given a set of measurements. For instance, financial analysts try to predict how a company's stock will perform based on such factors as current company performance measures, financial trends, competition and global events. A cancer researcher will try to predict whether a patient is likely to develop cancer in the next few years, on the basis of clinical measurements and history for that patient, and demographics. For years, experts would rely on accumulated knowledge or they would need to derive theoretical frameworks from first order principles in order to study such problems. However, for highly complex problems, the cognitive abilities of humans or simple approaches are not enough.

In recent years, alongside with great advances in terms of technology, algorithms and techniques have been developed within the theoretical computer science field, machine learning, and statistics to address the limitations of previous methods. Machine learning, today, provides numerous tools and algorithms for the automatic analysis of large amounts of data in order to reveal the predictive structure of complex problems.

Machine learning is the study of algorithms that can infer predictive models from data in regards to some task such as classification or regression. In supervised machine learning the data comprises of pairs of input variables (i.e. the features) and an output variable (i.e. the target).

1.1 Motivation

Performance estimation is, undoubtedly, one of the most important tasks of machine learning. Its importance is twofold; it is a measure to evaluate the generalization ability of a predictive model (i.e. how well it will perform given new, unseen data from the same distribution) and it is also used to compare a set of models against each other in order to choose the best one (i.e. to perform model selection). Therefore, it is crucial to have methods that can produce reliable and robust performance estimates that are not affected by extraneous factors such as sampling or partitioning of the data.

The ideal (also simplest) scenario for assessing the performance of a model would be to do so on a Hold-Out test set (a set of observations that was not included in the training of the model). However, when a large enough dataset cannot be held-out or when it is difficult to collect new observations, resampling methods such as Cross-Validation and the bootstrap are employed for estimating generalization performance.

In a typical supervised machine learning analysis multiple models are often constructed through a series of steps that include, among others, preprocessing of the data, feature (variable) selection, and the application of a learning algorithm, and the one with the smallest Cross-Validation error rate is selected and its associated performance is reported. For each of these steps there exists a wide selection of algorithms to choose from and most of them will have hyper-parameters[1]that need tuning. This leads to a large number of configurations[2]to be evaluated, and to an even larger number of models that will be constructed from these configurations.

Unfortunately, as the number of configurations under evaluation is getting larger, the risk of overfitting increases and the performance estimated by Cross-Validation is no longer an effective estimate of generalization but it is rather optimistically biased [4–11]. This phenomenon is called the problem of multiple comparisons in induction algorithms and has been analyzed in detail by Jensen in [12]. In addition, with increasing number of learned models, Cross-Validation becomes more computationally demanding which makes its use prohibitive.

A simple mathematical proof of the bias, due to the problem of multiple comparisons in induction algorithms, is as follows. Let μ_i be the average true performance (loss) of the models produced by configuration i when trained on data of size $|D_{train}|$ from the given data distribution. The sample estimate of μ_i on the tuning sets (if there are several as in Cross-Validation) is m_i, and so we expect that $\mu_i = E\{m_i\}$ for estimations that are unbiased. Returning the estimate of the configuration with the smallest loss returns $min\{m1, ..., m_n\}$, where n is the number of configurations tried. On average, the estimate on the best configuration on the tuning sets is $E\{min\{m1, ..., m_n\}\}$ while the estimate of the true best is $min\{\mu_1, ..., \mu_n\} = min\{E\{m_1\}, ..., E\{m_n\}\}$. The optimism (bias) is $Bias = min\{E\{m1\}, ..., E\{m_n\}\} = E\{min\{m_1, ..., m_n\}\} \geqslant 0$. For metrics such as classification accuracy and AUC, where higher is better, the min is substituted with max and the inequality is reversed.

The problem of bias of the estimated error rate based on K-Fold Cross-Validation (CV) has been addressed by researchers only in the last few years, even though it was pointed out as early as in 1984 by [13]. Proposed solutions to the problem include methods that use new procedures for performance estimation, and methods that try to assess the bias of the minimum error rate of the K-Fold CV and subtract it from the performance estimate. However, all proposed methods either tend to over-correct or have limitations which can make their use impractical.

[1]The term hyper-parameters refers to the algorithm parameters whose values are user defined.

[2]A sequence of modelling steps with specified algorithms and hyper-parameter values for each step (see Section 3.4).

1.2 Contribution

In this thesis we propose a new, general method for correcting the bias in Cross-Validation procedures, which works regardless of the data analysis task (e.g. classification, regression) or the structure of the models being involved in it. It has low computational overhead with respect to the Cross-Validation procedure and produces an almost unbiased expected error estimate even when the number of training samples is small. Unlike other methods, it is also suitable for correcting the bias of all Cross-Validation procedures including the Leave-One-Out CV and the Hold-Out CV.

The *bootstrap bias correction* (BBC) for Cross-Validation takes advantage of the bias correction properties of the bootstrap [14]. BBC is a simple method both conceptually and to implement, and only requires the predicted values of the models from which we want to choose the best one and assess its performance.

We also present and evaluate a method for dropping under-performing configurations along the way of Cross-Validation in order to speed it up. It uses the bootstrap in order to compare the configurations in terms of their performance in each iteration of CV.

1.3 Outline

The rest of the thesis is organized as follows. Chapter 2 surveys existing work on model selection and performance estimation and on the speeding up of the Cross-Validation procedure by specifically dropping under-performing configurations.

Chapter 3 defines terms and provides an overview of concepts from supervised machine learning. It presents and reviews some commonly used methods for performance assessment of a predictive model, model selection, and for simultaneously performing the two tasks.

In Chapter 4 the bootstrap and some of its primary applications are introduced. Then, the proposed bootstrap-based method for bias correction (BBC) is presented together with a method for calculating the confidence intervals of the performance estimate.

Chapter 5 introduces a method for eliminating under-performing configurations early in the K-Fold Cross-Validation procedure in order to speed it up.

In Chapter 6 we empirically compare the BBC method against the most popular methods for performance estimation and present results validating our method. We also describe the automated pipeline for supervised machine learning which we used to evaluate our proposed algorithms.

Finally, we conclude in Chapter 7 and overview possibilities for future work.

Chapter 2

Literature Survey

2.1 Model Selection and Assessment

Estimating the performance of a predictive model is an important task not only to assess how well the model generalizes to new, unseen data drawn from the same distribution, but also for selecting the combination of algorithms (e.g. for feature selection, for learning a classifier) and their hyper-parameter values which will induce the final, best performing model (a process also called model selection). These are fundamental tasks of a supervised machine learning analysis and they are usually performed simultaneously.

The Hold-Out method, where an independent test set is sequestered for assessing the performance of the selected model, is the simplest and most straightforward protocol for performance estimation. When the sample size is large enough, the Hold-Out estimate is a good approximation of the true performance.

Popular empirical estimators of performance based on resampling are the bootstrap [14] and the Cross-Validation (CV) protocol. CV was independently introduced by Stone [15], Allen [16], and Geisser [17] as a model selection method. Stone proposed the use of The Leave-One-Out Cross-Validation (LOO-CV) procedure as a way to choose the best algorithm hyper-parameter values and to assess the performance of the resulting diagnostic model. He was the first to differentiate between the use of CV for model selection and performance estimation. Allen in [16] proposed the Prediction Sum of Squares (PRESS) statistic for model selection. PRESS is similar to the LOO-CV where every measurement (sample) is considered in turn as a test case, for the model trained on all but the held out measurement. Geisser, independently in [17] introduced the Predictive Sample Reuse Method for model selection and assessment, a method equivalent to K-fold Cross-Validation.

Kohavi in [4], studied the use of K-Fold Cross-Validation and the bootstrap as performance estimation methods under different settings and compared them on real-world datasets. He concluded that the K-Fold Cross-Validation method is generally preferable to the bootstrap and has lower variance compared to the LOO-CV, and recommends the use of stratified 10-Fold

Cross-Validation for model selection. He also shows, through large scale experiments, that both the K-Fold CV and the bootstrap produce biased estimates of performance.

When comparing a large number of models in order to choose the one with the best performance, bias, towards better performance, is always present, unless the sample size is really large. This phenomenon is called the problem of multiple comparisons in induction algorithms and has been analyzed in detail in [12]. The problem of the optimistically biased K-Fold Cross-Validation estimate was pointed out as early as in 1984 by Breiman et al. in [13]. A more recent publication [7] empirically validates that the risk of overfitting increases with the number of models that are being tested and that the performance estimate of the K-Fold Cross-Validation is no longer a good approximation of the generalization performance. To date, a relatively limited amount of research has been devoted to the correction of this bias or to the development of new methods for performance estimation, and all proposed methods either tend to over-correct or have limitations which can make their use impractical.

Varma and Simon in [6] also showed in practice how optimistically biased is the K-Fold CV estimate and suggested the Nested K-Fold Cross-Validation (NCV) as a more reliable alterantive (see Section 3.4.4). NCV introduces an outer loop to the K-Fold CV procedure which results, essentially, in the "cross-validatory assessment of the cross-validatory choice" as was defined by Stone [15]. The NCV protocol, according to [6] and [10], produces an almost unbiased (but conservative) estimate of performance; however it is computationally expensive as the number of models that need to be trained is quadratic to the number of folds K. When the number of models is in the order of hundreds, the computational cost increases dramatically.

Tibshirani and Tibshirani (TT) [2] proposed a simple method for approximately estimating the bias of the K-Fold Cross-Validation protocol in order to correct it (see Section 3.4.3). The method requires minimum computational overhead in regards to the K-Fold CV procedure and [10] have shown (within the scope of their experiments) that it is robust and its results are statistically equivalent to the ones of NCV. However, TT is not suitable for LOO-CV or when the size of the test set is too small, and it was shown to over-estimate the bias in some settings [18].

Bernau et al. in [9] introduced two variants of a weighting-based method as a smooth analytical alternative to NCV; the WMC and the WMCS. The method is based on repeated subsumpling and computes a weighted mean of the resampling error rates for different models. The WMCS variant yields the best estimates between the two. It is competitive compared to the NCV on real data and has lower computational cost. However, the method is quite complex and is based on the assumption that the unconditional error rate estimates follow a multivariate normal distribution. If the number of tested models is really high, this assumption might be difficult to hold true.

Ding et al. in [11] proposed a resampling-based inverse power law (IPL) method for bias correction and compared its performance to those of NCV, TT, and WMC/WMCS on both simulated and real datasets. They estimate the error rate of each classifier by fitting a learning

curve which is constructed from repeatedly resampling the original dataset for different sample sizes and fitting an inverse power law function. The IPL method outperforms the other methods in terms of performance estimation but, as the authors point out, it exhibits significant limitations. Firstly, it is based on the assumption that the learning curve for each classifier can be fitted well by inverse power law. In addition, if the sample size of the original dataset is small, the method will provide unstable estimates. Finally, the IPL method has higher computational cost compared to TT and the WMC/WMCS methods.

Krstajic et al. in [19] describe and suggest algorithms for Repeated K-Fold Cross-Validation and Repeated Nested K-Fold Cross-Validation for model selection and assessment. The former is to deal with the variability in the reported optimal parameters of the simple K-Fold Cross-Validation which results from the arbitrary partitioning of the data in folds. The latter is to avoid the optimistically biased error estimation reported by the K-Fold Cross-Validation. Although both of the algorithms are shown to produce more reliable results compare to the simple K-Fold Cross-Validation, they are very computationally expensive.

2.2 Dropping of Under-Performing Configurations

To the best of our knowledge, the only work that focuses on the speeding up of the learning process by specifically eliminating , early in the CV procedure, under-performing configurations is the one by Krueger et al. [20]. Their method is called *Fast Cross-Validation via Sequential Testing* (CVST), and it uses nonparametric testing together with sequential analysis to choose the best performing configuration on the basis of linearly increasing subsets of data.

The CVST algorithm uses subsets of the data of size $m = s \cdot \frac{M}{S}$ as training sets, where s is the current step, S is the maximum number of steps of the algorithm and M is the total number of data points. The remaining $M - m$ data samples, serve as the test sets. By pairing this approach with an early stopping criterion, they aim to further speed up the model selection procedure.

The method starts with the whole set of configurations C and drops the under-performing ones at each step s. This is performed by applying the Friedman or the Cochrans's Q test (for regression and classification tasks respectively) on the predictions of the models produced by the k first configurations (ordered by their mean performance so far) on the $M-m$ data points until a value $\bar{k}$ for k has been found so that the tests indicate a significantly different performance of the configurations. Then, the $\bar{k}, ..., |C|$ configurations are further tested through sequential analysis (see [20]) to determine which of them will be discharged. If the remaining/active configurations have had similar performance in the last w_{stop} iterations, the execution stops. The final best configuration is then the one that has the best average ranking, based on performance, in the last w_{stop} iterations.

Chapter 3

Background

In this Chapter we briefly describe the supervised machine learning problem and formally define the notion of a predictive model (Section 3.1). In Section 3.2 we discuss the importance of estimating the performance (generalization error) of a model and present some of the most commonly used protocols for doing so. In Section 3.3, we describe the problem of model selection and how its solution is approximated in practice. Then in 3.4, we proceed with the description of procedures for simultaneously selecting and evaluating the (approximately) best model. Finally, we briefly explain stratified sampling and its benefits.

3.1 Supervised Machine Learning

In machine learning, the task of supervised learning refers to the process of constructing a predictive model from a finite set of acquired data. In order for the resulting model to efficiently predict the output variable, there needs to exist a relation between the input variables and the output. In cancer research, for instance, the goal is to infer a decision rule (i.e. the model) using different measurements from past cases, such as values of subsets of biomarkers, the location, stage, and size of the tumor, as well as the age, weight, risk behavior and clinical history of the patients (i.e. input variables), in order to predict or diagnose cancer on a new patient (i.e the output).

To define supervised learning more formally, we will first need to introduce some notation. Let $x^{(j)} \in \mathcal{X}_j$, for $j = 1, ..., N$ denote the value of the input variable X_j also known as *feature*, and let the n-dimensional vector $x_i = (x_i^{(1)}, x_i^{(2)}, ..., x_i^{(N)}) \in \mathcal{X}_1 \times \mathcal{X}_2, ..., \times \mathcal{X}_N = \mathcal{X}$ of input variables denote the i-th *sample* or *instance* of our data (i.e. a set of measurements). Finally, let $y_i \in \mathcal{Y}$ correspond to the value of the output variable Y for the i-th instance, also known as *target*. Then, our data D comprises of a finite set of data points or *training examples*:

$$D = \{(x_i, y_i), i = 1, ..., M\} \in \mathcal{X} \times \mathcal{Y},$$

where each training example (x_i, y_i) is a pair of input features and a target variable.

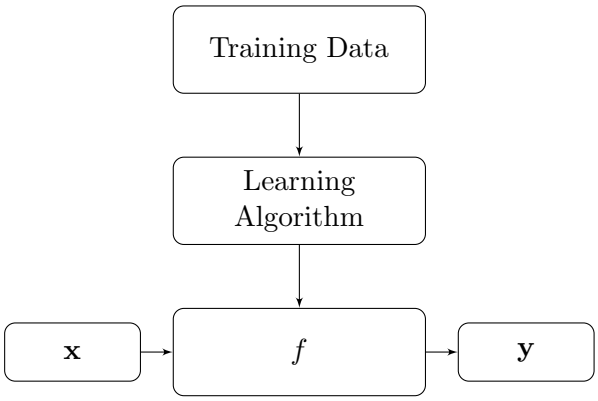

Figure 3.1: Supervised Learning

In the statistical sense, the input variables $X_1, X_1, ..., X_N$ and the output variable Y are drawn randomly from $\mathcal{X} \times \mathcal{Y}$ with respect to their joint probability distribution $P(X, Y)$ (population), where X is the vector of input variables. When implementing a learning algorithm, the data are usually represented as a 2-dimensional array $A \in \mathbb{R}^{M \times N}$ where the columns and the rows of the array correspond to the features and the samples accordingly.

In this context, the problem of supervised learning aims to learn a function:

$$f : \mathcal{X} \to \mathcal{Y},$$

from a collection of M training examples D, so that its predictions $f(X)$ are as close to being accurate as possible (i.e. are good estimators of the corresponding values of y). f is also called a *hypothesis*. Figure 3.1, simplistically, illustrates the process.

When the target variable Y that we are trying to predict is *qualitative*, also referred to as *categorical*, (i.e. Y takes on only a small number of discrete values), then the function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y} = \{c_1, c_2, ..., c_n\}$ is called a *classifier* and the learning task is called *classification*. When Y is *quantitative* or *numerical* then the function $f : \mathcal{X} \to \mathcal{Y}$, $\mathcal{Y} = \mathbb{R}$ is called a *regressor* and the learning problem is called *regression*.

Similarly to the target, we can distinguish input variables into numerical and categorical. An input variable x is called numerical when $\mathcal{X}_i = \mathbb{R}$, and categorical when $\mathcal{X}_i$ is a finite set of distinct values. Examples of the former case are age and weight, and of the latter case are gender and ethnicity. In the special case of the categorical variable taking only two distinct values (e.g. malignant, benign), the variable is called *binary*. Another type of variable is the *ordered categorical* or *ordinal* where there exists a natural ordering between the values that the variable takes. For example, suppose that you have a variable, that measures the economic status of a person, and takes the values “low”, “medium” and “high”.

3.2 Performance Estimation

In every supervised machine learning problem, it is important to obtain an accurate estimation of the performance of the inferred model; that is to assess how well the model generalizes to new, unseen data. In order to achieve this, we employ a *loss function* $L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ which measures the discrepancy between the truth Y and the predicted values $f(X)$.

The most commonly used loss function for classification is the *zero-one* loss function:

$$L(Y, f(X)) = \begin{cases} 1, & \text{if } Y \neq f(X) \\ 0, & \text{otherwise} \end{cases}$$

which penalizes the errors in predictions in a equal way. In the case of regression, the most used loss function is the *squared error* loss function:

$$L(Y, f(X)) = (Y - f(X))^2$$

where the larger the differences between the true values of Y, and the predicted values $f(X)$, the proportionally more they will be penalized.

In this framework, we can restate the problem of supervised machine learning as finding a model that minimizes the *generalization error*. We use $f(D)$ to denote the model built from the learning set D, and $f(X, D)$ to denote the output (predictions) of that model when applied on the set of input variables X.

Definition 3.2.1 *The generalization error, also known as test error, of a model $f(D)$ is* [1]*:*

$$Err_D = E_{X,Y}\{L(Y, f(X, D))|D\} \tag{3.1}$$

where D is the set of data points used to train $f(D)$, and L is the loss function for measuring errors between Y and $f(X, D)$. Here the training set D is fixed, and test error refers to the error for this specific training set [3].

In [3] the authors point out that the estimation of conditional error cannot easily be done effectively given only the information in the same training set. Most of the methods described later on in this Chapter, effectively estimate a related quantity which is the *expected generalization error*.

[1] $E_X f(X)$ denotes the expectation of $f(x), x \in \mathcal{X}$ with respect to the distribution of the random variable X. It is defined as

$$E_X\{f(X)\} = \sum_{x \in \mathcal{X}} P(X = x) f(x)$$

Definition 3.2.2 *The expected generalization error, also known as expected test error, of a model* $f(D)$ *is:*

$$Err = E_{X,Y}\{L(Y, f(X, D))\} = E_D\{Err_D\} \tag{3.2}$$

which averages over all that is random, including the randomness related to the training set D that was used to infer the model.

Typically, as the complexity of a model increases, it adapts more to the training data. This results in a model with lower bias but higher variance. What we are aiming for, is a useful trade-off between bias and variance in a model so that it gives minimum expected generalization error.

A naive and clearly poor estimate of the expected test error Err is the *training error* or *resubstitution estimate*:

$$\widehat{Err}_{train} = \bar{E}(D, f(D)) = \frac{1}{M}\sum_{i=1}^{M} L(y_i, f(x_i)) \tag{3.3}$$

which is essentially, the average loss over all training samples that we used to fit the model. As the model becomes more complex though, the training error decreases and could even drop to zero. In this case, the model will be *overfitting* the data and training error will be an overly optimistic estimate of how well the model generalizes to new data.

From this moment on, we will be using the notation $\bar{E}(D', f(D))$ to refer to the mean error/loss of model $f(D)$ on a specific test set D'. The datasets D and D' could differ or be equal as in equation 3.3.

In the following subsections we present a few of the most commonly used methods for performance assessment. A fair amount of research [3–7, 10, 21, 22] has focused on the study of their fitness for assessment of test error and for model selection (see Section 3.3).

3.2.1 Hold-Out Cross-Validation

In an ideal scenario, we would first want to build the predictive model $f(D)$ from a dataset D, make it operational on its intended environment, and then apply $f(D)$ on a newly collected set of training examples $D' \in (\mathcal{X} \times \mathcal{Y})\backslash D$ and estimate the model's performance on D'. However, in most real applications it is not always possible to draw additional data, thus making the estimation of the performance on set D' infeasible.

We could simulate this idea by randomly splitting the original dataset D into two disjoint subsets; the *training set*, D_{train}, consisting of m observations and the *test set*, D_{test}, of the remaining $M - m$ observations. The training set is used to fit the model and the test set, often referred to as the *hold-out* set, is used to evaluate the performance. This method is known as the *Hold-Out Cross-Validation* or *Hold-Out CV* and produces the *Hold-Out estimate* of the expected test error:

$$\widehat{Err}_{holdout} = \bar{E}(D_{test}, f(D_{train})) \tag{3.4}$$

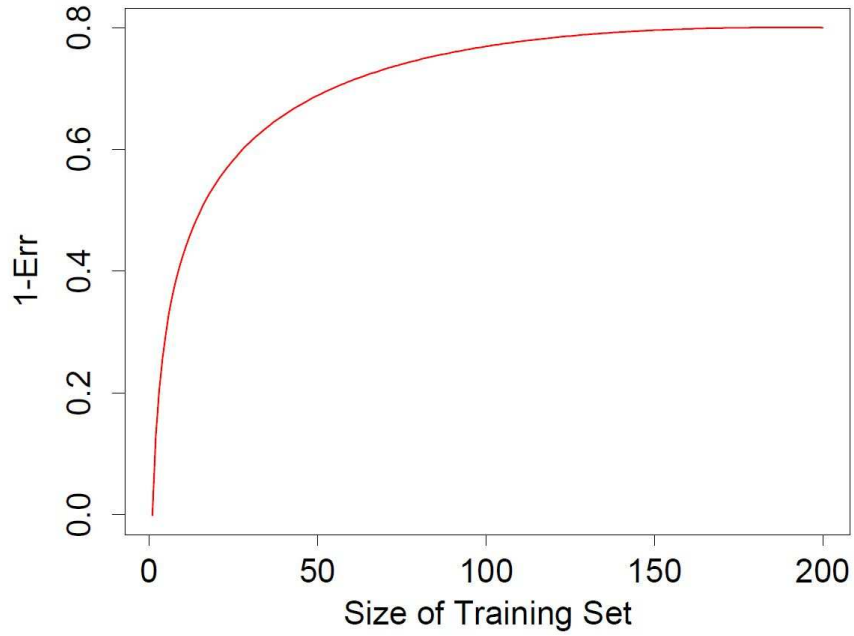

Figure 3.2: Hypothetical learning curve for a classifier on a given task: a plot of 1-Err versus the size of the training set M. (Figure from [3])

Since D_{train} and D_{test} are non-overlapping, we could consider them to be independent, avoiding thus the optimism of the training error estimation of equation 3.3. However, a *learning curve* related bias is still introduced due to the fact that the Hold-Out CV estimate is conditioned on less than M samples. The learning curve of a model is simply a graphical representation of the increase of performance (or decrease of error) with experience (i.e. the number of training examples). As seen in Figure 3.2, we typically expect the prediction error to decrease as the sample size increases, and to gradually reach a plateau.

Ideally, the sizes of the D_{train} and D_{test} sets would be such, so that the trained model operates on the plateau of the learning curve, and yet there would be enough test samples left to get an accurate estimation of the performance. However, if the training set is relatively small, then the estimate of the expected test error would be biased upward. For example in Figure 3.2 we would expect this behavior for $|D_{train}| \lesssim 100$. Unfortunately though, the learning curve is not known a priori. In [23], Guyon investigates the problem and proposes a formula for efficiently splitting the data, which will lead in large enough training sets and yet small error rates. A rule of thumb is to have 2/3 of the data serve as the training set and the rest as the test set [4].

Another drawback of the method is that the choice of m is arbitrary as there are $S(M,2) \approx 2^M$ possible ways to split the data, where $S(M,2)$ is a Stirling number of the second kind [24]. This, could lead us to assume that there is variance in the estimate of the performance of the Hold-Out CV method depending on the specific choice of split for the data. This variance could be eliminated by trying all possible $S(M,2)$ splits of the samples or by repeatedly and randomly splitting the data and averaging the results. This leads to the so called *Repeated Hold-Out Cross-Validation* protocol, which will not be presented in detail here.

However, when we have large enough training and test sets, the Hold-Out CV is a generally acceptable and robust method for providing a good approximation of the generalization error. Its simplicity and low computational cost make it appealing in such cases.

In the case that the number of the training examples is rather small to medium, the Hold-Out CV estimate might not be reliable and therefore other protocols are preferred.

3.2.2 K-Fold Cross-Validation

The *K-Fold Cross-Validation* (K-Fold CV) protocol is probably the most widely used method for performance assessment for small and medium sample sizes. It has been proposed independently by Stone, Allen and Geisser in [15–17] respectively.

K-Fold CV consists of randomly splitting the data into $K \ll M$ mutually exclusive subsets $F_1, F_2, ..., F_K$, also known as *folds*, of approximately equal size. It then uses the samples in $D \backslash F_i, i = 1, ..., K$ to train the model $f(D \backslash F_i)$ and the remaining samples in F_i to estimate its performance. The *K-Fold CV estimate* of the expected test error is then defined as:

$$\widehat{Err}_{KCV} = \frac{1}{K} \sum_{i=1}^{K} \bar{E}(F_i, f(D \backslash F_i)) \tag{3.5}$$

which is the average of the prediction errors of the produced models $f(D \backslash F_i), i = 1, ..., K$ for their respective test sets F_i. Again here, $\bar{E}(D', f(D))$ refers to the mean loss of model $f(D)$ when applied on D'.

Figure 3.3 illustrates the K-Fold CV procedure for $K = 10$. The top rectangle represents the original dataset D which is divided into 10 folds represented by the smaller rectangles. Each smaller rectangle/fold is considered in turn as the test set (shown in blue), for the models trained on the rest of the rectangles/folds (shown in white).

K-Fold CV could be considered to be an extension of the Hold-Out CV repeating it K times. One of its major advantages is that each data point (x, y) serves once as a test case, thus making the test size equal to the total number of samples in the dataset $|D|$.

The K-Fold CV protocol effectively estimates the expected test error [3]. Similarly to the Hold-Out CV method, this estimate could be biased upward depending on the learning curve of the model under assessment on the given task, leading to an overestimation of the true prediction error. Increasing K, thus having larger training sets, would result in a less conservative estimate of the performance. Unfortunately, this would also result in higher variance in the estimation since the training sets would become more similar to one another, making the resulting models more correlated. K could be as large as $|D| = M$. This is a special case of K-Fold CV, known as the *Leave-One-Out Cross-Validation, (LOO-CV)* method.

LOO-CV is a variant of K-Fold CV, in which each data sample takes the role of the test set in turn and the rest serve as the training set. The LOO-CV is approximately unbiased, since almost the entire dataset is used for fitting the model each time, but can have higher variance than

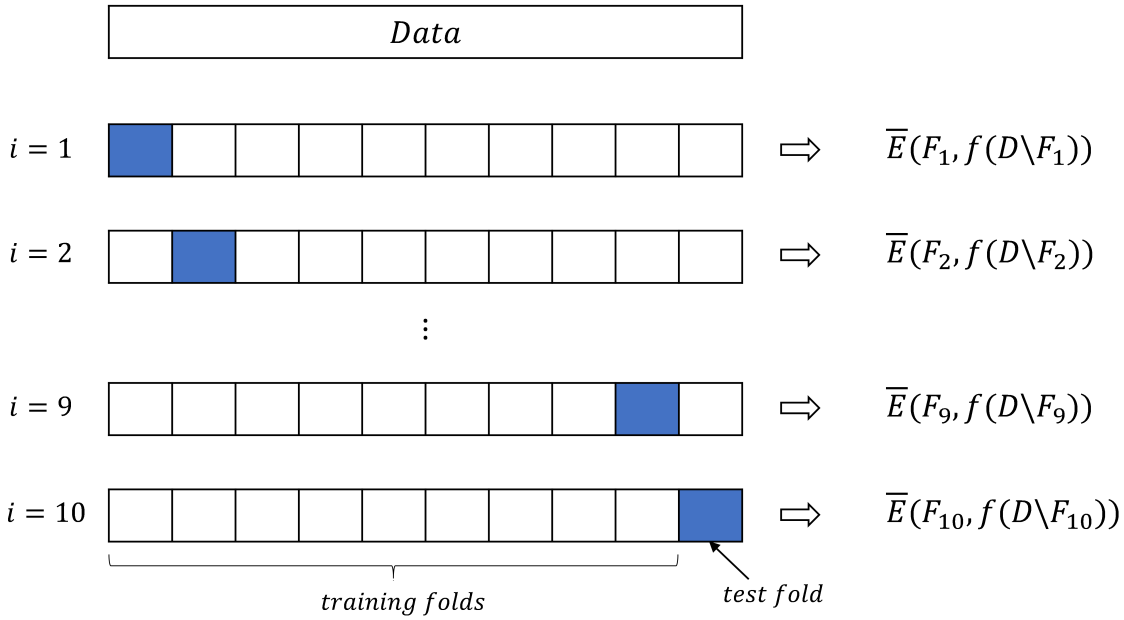

Figure 3.3: 10-Fold Cross-Validation. In each round/iteration of Cross-Validation one fold (in colour) is used for testing and the rest of the folds are merged into the training set.

K-Fold CV as the M training sets greatly overlap each other [3]. It is also highly computationally intensive, as it needs to fit M models (as many as the samples) in order to produce the error estimate.

A typical value for K that has been found to produce good results, relatively balancing out the trade-off between bias and variance, is $K = 10$ [4].

3.3 Model Selection

3.3.1 Bayes Model and Residual Error

Assuming that the conditional probability distribution of Y given X, $P(Y|X)$, is known, then the best model for a specific task would be derived as follows. The *expected test error* can be written as:

$$Err = E_{X,Y}\{L(Y, f(X, D))\} = E_X\{E_{Y|X}\{L(Y, f(X, D))\}\} \tag{3.6}$$

and the model f_B which minimizes it, is a model which minimizes the inner expectation for each point x of the input space $\mathcal{X}$:

$$f_B = \underset{y \in \mathcal{Y}}{argmin} E_{Y|X=x}\{L(Y, y)\} \tag{3.7}$$

The model f_B is referred to as the *Bayes model* and its expected generalization error $Err(f_B)$ is known as the *residual error* in supervised machine learning. This is the minimal error that can be achieved, due to random deviation (or noise) in the data.

3.3.2 Selecting the best (possible) model

Having defined the Bayes model, solving the problem of finding the best model, would be equal to the problem of estimating the conditional probability distribution $P(Y|X)$ from the set of training examples D_{train}. However, a close estimation of $P(Y|X)$ requires $|D_{train}|$ to grow exponentially with the number of input variables which renders this solution infeasible [25].

In practice, when dealing with problems of high dimensionality, one must make assumptions on the properties of the function f. In particular, a collection of candidate models of specific structure is assumed to contain the best one, and then, optimization is performed among these models, based on the learning set, in order to find the one which minimizes the expected error. This collection of models is often referred to as the *Hypothesis space* and is denoted by H. It is likely that the final selected model f is not close to the Bayes model f_B of the specific problem (i.e. $Err(f) \gg Err(f_B)$), however there may exist models in H which approximate f_B.

Definition 3.3.1 *The approximation error, defined by:*

$$Err_H = \min_{f \in H}\{Err(f)\} - Err(f_B) \tag{3.8}$$

is a measure of how well the models in H can approximate the optimal model f_B.

It is intuitive that the larger the hypothesis space H, the smaller the approximation error will be. However, when fitting multiple models on a finite training set, the estimation of the performance of the selected one will be optimistically biased (i.e. the model will be *overfitting* the data). This phenomenon is also called *the problem of multiple inductions* in machine learning and is described in detail in [12]. Loosely, when the number of trained models is large, then a more complex model with low bias and high variance (i.e. the "luckiest" one for the particular training set) will be consistently chosen and it will generalize poorly on new data.

When performing a machine learning analysis, this finite set of models that we wish to find the best performing one from, typically consists of a few different learning algorithms (e.g. polynomial regression, SVM, neural network) combined with several different combinations of values for their *hyper-parameters*. The term *hyper-parameters* refers to the set of parameters for a certain algorithm that are user defined and cannot be directly estimated from the training set. For example, the degree of a polynomial regression model or the number of hidden layers of a neural network.

3.3.3 Algorithm and Hyper-Parameter Optimization

It is important to note here that there exist conceptual differences between *model selection* in the context of machine learning and in the context of statistical modelling. In the case of statistical modelling, all the final candidate models f are produced, their performance is evaluated based on an estimator of generalization performance (e.g. Hold-Out CV, K-Fold CV), and the best one among them is then chosen and put to use. However, in the case of machine learning the term *model selection* often refers to the problem of *algorithm and hyper-parameter optimization* or *tuning* where only the combination of the learning algorithm and its hyper-parameter values that produced the best performing model (again on the basis of an estimation of their performance) are returned. The latter term is more appropriate to use in the case of retraining the selected combination of algorithm and its hyper-parameter values on the entire dataset. Retraining on all data returns a different model than the one employed for estimating the performance. However, under the assumption that the loss of a learning algorithm drops monotonically, on average, with increasing sample size, this is generally a better model to put in use.

In our practice, we employ the retraining step on the complete dataset and so we use the terms *model selection* and *algorithm and hyper-parameter optimization* (or *tuning*) without any distinction between them, to refer to the process of finding the best combination of algorithm and its hyper-parameter values.

More formally, let A represent a learning algorithm, and $\mathcal{A}$ represent a set of algorithms $\{A^{(1)}, A^{(2)}, ..., A^{(l)}\}$. Let, also, $\theta^{(i)} \in \Theta^{(i)}$ represent the vector of hyper-parameter values of the i-th algorithm $A^{(i)}$. We will denote by $A(D, \theta) = f(D, \theta)$ the model produced when applying algorithm A with hyper-parameter values θ on data D. Each hyper-parameter space $\Theta^{(i)}$ of a learning algorithm is a subset of the cross product of the domain spaces of each individual hyper-parameter of the algorithm which could be continuous or discrete. Then the problem of *Algorithm and Hyper-Parameter Tuning* becomes:

$$\{A^*, \theta^*\} = \underset{\substack{A^{(i)} \in \mathcal{A}, \theta^{(i)} \in \Theta^{(i)} \\ i=1,...,l}}{argmin} \{Err(A(D, \theta))\} \tag{3.9}$$

Finding $\{A^*, \theta^*\}$ is usually difficult, since the generalization error of each model needs to be computable. However, good enough θ can be found which approximate the performance of θ^*.

As with performance estimation, when the size of the dataset D is relatively large, one could use the *Hold-Out Cross-Validation* method (see Section 3.2.1). Instead of fitting just one model on the training set D_{train} and testing its performance on the test or holdout set D_{test}, now, a number of different models are fit on D_{train}, and the one that minimizes the error on D_{test} is chosen:

$$\{A^*, \theta^*\} = \underset{\substack{A^{(i)} \in \mathcal{A}, \theta^{(i)} \in \Theta^{(i)} \\ i=1,...,l}}{argmin} \{\bar{E}(D_{test}, A(D_{train}, \theta))\} \tag{3.10}$$

This method exhibits the same advantages and disadvantages in terms of bias and variance as when it is used for performance estimation (see Section 3.2.1).

The most commonly used method for small and medium datasets is the *K-Fold Cross-Validation* with the appropriate adjustments to perform tuning:

$$\{A^*, \theta^*\} = \underset{\substack{A^{(i)} \in \mathcal{A}, \theta^{(i)} \in \Theta^{(i)} \\ i=1,\dots,l}}{argmin} \{\frac{1}{K} \sum_{i=1}^{K} \bar{E}(F_i, A(D \backslash F_i, \theta))\} \tag{3.11}$$

Finding $\{A^*, \theta^*\}$ that satisfy even the last two equations is not an easy task since it still requires optimizing over the combinations of algorithms $A \in \mathcal{A}$ and different settings of their respective hyper-parameter values. The set $\mathcal{A}$ can be restricted to a few different algorithms. However, some of the hyper-parameters of the learning algorithms could be taking values from continuous domains or from infinite sets. Some of the more general methods for approximating a solution, that are also used in practice are *manual search*, *grid search*, *random search* and *Bayesian optimization*. In *manual search* a human analyst, based on their knowledge on the problem or using rules-of-thumb, will try some initial choices for A and θ and according to the performance of the resulting model, they will tweak them and repeat the process until some well performing values have been identified. *Grid search* refers to the process of exhaustively searching through a manually specified subset of the hyper-parameter space of a learning algorithm and is an automated procedure. *Random search* simply samples hyper-parameter settings for a predefined number of times or until some condition has been met. It has been shown to achieve same or better performance than grid search and is computationally more efficient in high dimensional hyper-parameter spaces [26]. *Bayesian optimization* is a methodology for the global optimization of noisy black-box functions. In general, Bayesian optimization first assumes a statistical model that captures the dependence of a loss function on hyper-parameter values. It then proceeds by repeatedly using this model to identify hyper-parameter settings in a way that trades off exploration (i.e. choose hyper-parameter values that their performance is uncertain) and exploitation (i.e. choose hyper-parameter values that are promising), evaluate their performance and update the original model with the new observations. Bayesian optimization has been well studied [27–30] and in practice, it seems to obtain better results with fewer trials of hyper-parameter settings than grid search and random search.

In all cases, optimizing over θ must be made cautiously; the model trained with hyper-parameter values θ should be neither too complex nor too simple. A too complex model will have adapted to the data and will have low error on the training set but higher error on the test set (i.e. it will generalize poorly). In this case, we say that the model is *overfitting*. On the contrary, when the model is too simple, it is said to *underfit* the data because it is unable to capture the true relation between the input variables X and the output Y. This kind of model will show high error on both the training and the test set. The most appropriate choice for the values θ would be those that balance the trade-off between variance and bias so that the model is of moderate

complexity.

3.4 Simultaneous Model Selection and Evaluation

In practice, when analyzing real problems, it is desirable that both the final model (i.e. combination of learning algorithm and its hyper-parameter setting) is selected and its performance is evaluated. The combination of the two tasks is not trivial. However, there exist methods that have been studied [3–7, 10, 21, 22] for their properties and effectiveness in doing so, and are widely used in the field of machine learning.

Typically, a supervised learning analysis will also consist of performing a few more steps than just optimizing for the learning algorithm and its hyper-parameter values. For instance, imputation may need to be applied if the data has missing. Binarization of categorical variables, standardization of numerical features or completely different representations of the data could also be tried. Feature selection could also be applied in the case that we need to restrict the input space of the problem or when we are more interested in identifying which of the input variables are more related to the outcome (i.e. diagnose the causes of the problem).

Most of these extra steps will also require to choose among a plethora of algorithms that also need tuning. The correct way to proceed is to incorporate all these steps in the protocol that is used for model selection and evaluation. As discussed in [3], if feature selection is applied to all the data prior to using some estimator of performance and hyper-parameters (e.g. Cross-Validation), then the assessment of the error of each model will not be performed on a completely independent test set since those samples were already used in the process of selecting the features. In general, every step in an analysis which involves "peeking" into the output variable, should be incorporated in the preferred protocol.

Essentially, the modelling procedure consists of multiple steps and we need to optimize in terms of the combinations of all the algorithms involved in each step and their respective hyper-parameters. Let c denote a sequence of modelling steps, with specified algorithms and hyper-parameter values for each step, also referred to in this work as *configuration*. For example, c could be "standardization of numerical variables, LASSO for feature selection with hyper-parameter lambda equal to 0.1, Random Forests with 1000 trees". Let $C = \{c_1, c_2, ...c_l\}$ denote a finite set of candidate configurations to be tried. Then, $f(D, c)$ is the model produced when the sequence of modelling steps c is applied to D. It is important to note that c is completely defined (i.e. there are no undefined hyper-parameters or other choices to be made on it). In this sense, optimizing over c is the same as performing a grid search for all possible combinations of algorithms and their respective hyper-parameter settings for all steps in the analysis pipeline.

All the procedures in the following Sections will be described in the context of the model selection problem being the optimization of an entire analysis pipeline.

Algorithm 1 Train-Validation-Test

Input: A training set $D = (x, y) \in X \times Y$, A finite set of learning configurations C
Output: A model $\mathcal{M}$, An estimation of performance P of model $\mathcal{M}$

1: **function:** $TVT(D, C)$
2: randomly partition D into three disjoint subsets $D_{train}, D_{validation}$ and D_{test}
3: **for** each configuration c $\in$ C **do**
4: $\quad \hat{e_c} = \bar{E}(D_{validation}, f(D_{train}, c))$
5: **end for**
6: $c^* = \underset{c \in C}{argmin}\{\hat{e_c}\}$
7: $P = \bar{E}(D_{test}, f(D_{train} \cup D_{validation}, c^*))$
8: $\mathcal{M} = f(D, c^*)$
9: **return** $\mathcal{M}, P$

3.4.1 Train-Validation-Test Protocol

A simple way of assessing the performance of the selected model would be to use the test set error estimate $\bar{E}(D_{test}, f(D_{train}, c^*))$ of the Hold-Out method (see equation 3.10). Simple yet naive, since the model was chosen due to its performance on the test set and therefore it cannot be considered independent from it. As a result, the reported performance estimate on the same test set will be optimistically biased.

To produce a reliable estimation of the performance of the model, we need to evaluate it on a separate, independent test set. To do so, we can modify the Hold-Out protocol as shown in Algorithm 1. First, the dataset is split into three disjoint sets $D_{train}, D_{validation}$ and D_{test}. Then, model selection (i.e. chose the best configuration $c \in C$) is performed on $D_{train} \cup D_{validation}$ using test sample estimates (Lines 3-6). After the best configuration c^* is identified, the unbiased expected generalization error of the model trained on $D_{train} \cup D_{validation}$ is estimated on D_{test} in Line 7. Finally, the returned model is learned on the entire set of learning examples (Line 8).

As with the simpler cases of Hold-Out for performing model selection (see Section 3.3.3) or performance estimation (see Section 3.2.1), one major drawback is that it "wastes" data. Again, a learning curve related bias is introduced as the Hold-Out CV estimates are always conditioned on less than $|D|$ samples. If the training set is relatively small, then the estimate of the expected test error would be biased upward (see Figure 3.2). Also, the arbitrary choice of splitting of the data will introduce some variance to the estimate. Ideally, we want a large enough training set so that the fitted model operates on the plateau of the learning curve, and yet there would be enough samples to ensure accurate estimates of the performance. A rule-of-thumb for splitting the data is 60% of the data instances for the training set $|D_{train}|$ and 20% for each of the validation and test sets $|D_{validation}|$ and $|D_{test}|$.

Algorithm 2 K-Fold Cross-Validation

Input: A training set $D = (x, y) \in X \times Y$, A finite set of learning configurations C, A positive integer K

Output: A model $\mathcal{M}$, An estimation of performance P of model $\mathcal{M}$

1: **function:** $KCV(D, C, K)$
2: randomly partition D into K disjoint subsets $F_i, i = 1..K$ of approximately equal size
3: **for** $i = 1$ to K **do**
4: **for** each configuration c $\in$ C **do**
5: $\hat{e_c}^{(i)} = \bar{E}(F_i, f(D \backslash F_i, c))$
6: **end for**
7: **end for**
8: $\hat{e_c} = \frac{1}{K} \sum_{i=1}^{K} \hat{e_c}^{(i)}$
9: $c^* = \underset{c \in C}{argmin}\{\hat{e_c}\}$
10: $P = \underset{c \in C}{min}\{\hat{e_c}\}$
11: $\mathcal{M} = f(D, c^*)$
12: **return** $\mathcal{M}, P$

3.4.2 K-Fold Cross-Validation

The K-Fold Cross-Validation method could also be used to simultaneously perform model selection and model assessment. The procedure is detailed in Algorithm 2. First, the data instances are randomly split into $K \ll M$ mutually exclusive subsets $F_1, F_2, ..., F_K$, also known as *folds*, of approximately equal size. Then, model selection is performed using K-Fold estimates in Lines 3-9 (i.e. find the configuration c^* which minimizes the mean error over the K test folds F_i). $\hat{e_c}^{(i)}$ is the average loss in fold F_i of the model trained on $D \backslash F_i$ with configuration c. The model returned is the one learned from the entire dataset using the best configuration c^* (Line 11) and its performance P is evaluated as the mean performance of the K different models that were constructed on different subsets of the data with configuration c^*, $f(D \backslash F_i, c^*)$.

Similarly to the case of the simple Hold-Out CV method, where we only had one test set for both selecting and evaluating the model, the K-Fold CV will not provide an accurate estimate of the expected generalization error since this same quantity was used to guide the model selection procedure. Indeed, as it has been shown in [4, 6, 8, 10, 13, 22] the K-Fold CV estimate of the performance will be biased, especially when the number of data samples is relatively small.

To guarantee an unbiased estimate, the test sets (folds) on which the expected test error is evaluated should be kept out of the entire process of model selection and only be used once the best configuration c^* is selected. A protocol that follows this procedure is the *Nested K-Fold Cross-Validation* described in detail in 3.4.4.

Algorithm 3 Tibshirani and Tibshirani

Input: A training set $D = (x, y) \in X \times Y$, A finite set of learning configurations C, A positive integer K

Output: A model $\mathcal{M}$, An estimation of performance P of model $\mathcal{M}$

1: **function:** $TT(D, C, K)$
2: $\mathcal{M}, P_{KCV}, c^*, c_i^* = KCV(D, C, K)$
3: $\widehat{Bias} = \frac{1}{K}\sum_{i=1}^{K}(\hat{e}_{c^*}^{(i)} - \hat{e}_{c_i^*}^{(i)})$
4: $P = P_{KCV} + \widehat{Bias}$
5: **return** $\mathcal{M}, P$

3.4.3 Tibshirani and Tibshirani (TT) Bias Correction

The *Tibshirani and Tibshirani* (TT) protocol applies the idea of approximately estimating the bias of the minimum K-Fold CV error rate in K-Fold Cross-Validation [2]. The method is outlined in Algorithm 3. Apart from the best overall configuration c^*, the TT protocol also needs the configurations c_i^*, which minimize the expected error for each of the folds $F_i, i = 1, ..., K$ (Line 2). Then, the bias due to model selection is estimated for each fold as the average loss of the model trained with the overall best found configuration c^*, minus the average loss of the model trained with the best configuration c_i^* for that particular fold. The overall bias $\widehat{Bias}$ then, is the mean bias over the K folds (Line 3). The adjusted estimate of performance of the TT protocol is then:

$$\widehat{Err}_{TT} = \widehat{Err}_{KCV} + \frac{1}{K}\sum_{i=1}^{K}(\hat{e}_{c^*}^{(i)} - \hat{e}_{c_i^*}^{(i)}) \tag{3.12}$$

where $\hat{e}_{c^*}^{(i)}$ is the average loss in fold i of the model trained with configuration c^* and $\hat{e}_{c_i^*}^{(i)}$ is the average loss in fold i of the model trained with configuration c_i^*. Assuming that the data partitioning is the same, then *K-Fold CV* and *TT* return the same model.

The TT has minimum computational overhead as it does not require the training of any additional models other than the ones already trained by the K-Fold Cross-Validation. Tsamardinos et al. in [10], show that the method is robust providing conservative performance estimates which are statistically equivalent to those of the Nested K-Fold Cross-Validation protocol (see Section 3.4.4). They suggest the use of TT over the Nested K-Fold Cross-Validation mailny due to its lower computational complexity. However, TT is not suitable for LOO-CV or when the size of the test set is too small, and it was shown to over-estimate the bias in some settings [18].

3.4.4 Nested K-Fold Cross-Validation

Varma and Simon in [6], report a bias in error estimation when using K-Fold Cross-Validation, and suggest the use of the *Nested K-Fold Cross-Validation* (NCV) protocol as an almost unbiased estimate of the true performance. NCV introduces an outer loop to the K-Fold CV procedure which results, essentially, in the "cross-validatory assessment of the cross-validatory choice" as

Algorithm 4 Nested K-Fold Cross-Validation

Input: A training set $D = (x, y) \in X \times Y$, A finite set of learning configurations C, A positive integer K

Output: A model $\mathcal{M}$, An estimation of performance P of model $\mathcal{M}$

1: **function:** $NCV(D, C, K)$
2: randomly partition D into K disjoint subsets $F_i, i = 1..K$ of approximately equal size
3: **for** $i = 1$ to K **do**
4: $\quad c_i^* = KCV(D \backslash F_i, C, K)$
5: $\quad \hat{e_i} = \bar{E}(F_i, f(D \backslash F_i, c_i^*))$
6: **end for**
7: $P = \frac{1}{K} \sum_{i=1}^{K} \hat{e_i}$
8: $c^* = KCV(D, C, K)$
9: $\mathcal{M} = f(D, c^*)$
10: **return** $\mathcal{M}, P$

was defined by Stone [15]. Algorithm 4 details the method. First, the data samples are randomly split into $K \ll M$ mutually exclusive subsets $F_1, F_2, ..., F_K$, also known as *folds*, of approximately equal size. Then, model selection (i.e. find the best configuration c^*) is performed using *K-Fold Cross-Validation* on $D \backslash F_i$ and the expected generalization error $\hat{e_i}$ of $f(D \backslash F_i, c^*)$ (i.e. the model trained using configuration c^* on all the data but fold F_i) is evaluated on F_i, for $i = 1, ..., K$ (Lines 3-6). The (unbiased) expected generalization error of the final selected model is evaluated as the average generalization error of the models $f(D \backslash F_i, c^*)$ over the folds (Line 7). Finally, the best configuration c^* is determined by performing model selection on the entire dataset D using K-Fold Cross-Validation (Line 8), and the final model is learned on D using the chosen configuration c^*.

It is important to note that the final returned model is the same as the one returned by the K-Fold Cross-Validation protocol (assuming same splitting of the data into folds). The main drawback of the method is that it is computationally expensive since the number of models that need to be trained are $> K^2 \cdot |C|$, where C is the configurations set.

Although Varma and Simon [6] are (probably) the first to explicitly study the effectiveness of the Nested Cross-Validation protocol, the method (or similar ones) has been used as early as 2003 and 2005 [31–33] and has gained popularity since then due to its almost unbiased (slightly conservative [10]) error estimate.

Figure 3.4 illustrates the Nested K-Fold Cross-Validation protocol where $K = 5$ for both the outer and the inner loop of the procedure. Each of the folds of the outer loop serves in turn as a test set (shown in blue) for estimating the performance of the configuration returned by the 5-Fold Cross-Validation (the inner loop).

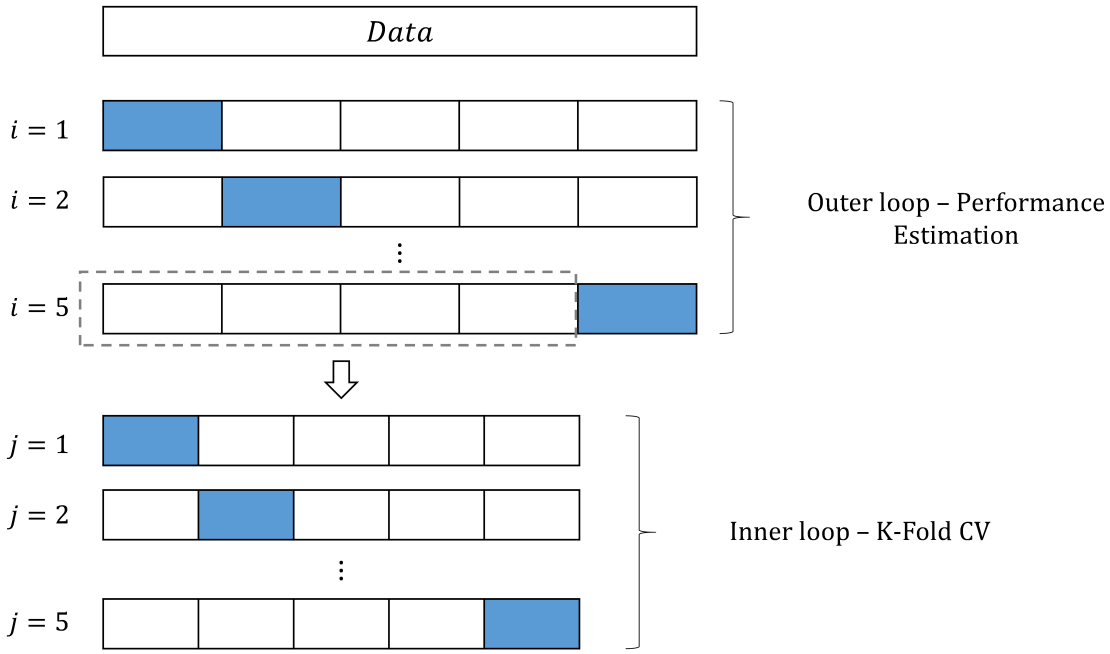

Figure 3.4: Nested K-Fold Cross-Validation with $K = 5$ for both the outer and the inner loop of the procedure.

3.5 Stratification of Folds

Most of the methods described in this Chapter, require that the dataset is split into two or more subsets. The Train-Validation-Test and the K-Fold CV protocols partition the data into three and K disjointed subsets respectively. When the dataset is large enough, randomly splitting the data will (usually) guarantee that the distribution of the output classes will be the same in each subset. However, with small sample sizes or with unbalanced class distributions it could be the case that some of the subsets' distributions are different from the distribution of the original dataset. In extreme cases, some of the subsets will contain no samples from one or more of the classes. Then, the estimation of performance for that subset will exclude some classes. *Stratified partitioning* of the dataset ensures that the resulting subsets will maintain the original distribution. Kohavi in [4] and Tsamardinos et al. in [10] recommend the use of stratification as a better option compared to random splitting, both for bias and variance.

Chapter 4

Proposed Method for Model Selection and Evaluation

In this Chapter we briefly outline the limitations of existing methods for model selection and assessment (Section 4.1). We then describe the bootstrap procedure and some of its primary applications in Section 4.2. In Section 4.3, we present our proposed method for model selection and evaluation of its performance and discuss its properties and advantages compared to the existing ones. We conclude with a method (still under evaluation) for calculating the confidence intervals of the estimated performance (Section 4.4).

4.1 Limitations of Existing Methods

In a real supervised machine learning analysis, it is important not only to find the best model, but also to evaluate its performance (i.e. how the model generalizes to unseen data). A few of the most commonly used methods for simultaneously performing the two tasks are the *Train-Validation-Test* protocol, the *Tibshirani & Tibshirani* protocol and the *Nested K-Fold Cross-Validation* protocol which are described in detail in Sections 3.4.1, 3.4.3, and 3.4.4 respectively. However, all these protocols, appear to have limitations under some circumstances which can make them impractical.

The Train-Validation-Test protocol simulates the ideal, data-rich scenario where each task (i.e. training, selection, assessment) will be carried out using different, independent and large enough (virtually infinite) sets of data samples drawn from the same distribution $P(X, Y)$. In practice, it is often not possible to draw additional data, and the set of learning instances D constitutes the only available data. When the training set is relatively small, then a *learning curve* bias is introduced in the estimate. On the other hand, the arbitrary choice of splitting of the data samples will introduce variance to the estimate (see Sections 3.2.1, 3.4.1).

The Tibshirani & Tibshirani (TT) [2] and the Nested K-Fold Cross-Validation protocols [6], both provide a conservative performance estimate [10]. TT tries to approximately estimate

the bias in the Cross-Validation error estimate without any significant computational overhead, however it is not suitable for the LOO-CV or when the size of the test set is too small. The Nested K-Fold Cross-Validation protocol provides an almost unbiased error estimate [6] but the use of it can be prohibitive due to its computational cost since the number of models that need to be trained are quadratic to the number of folds.

The WMC/WMCS method [9] is quite complex and is based on a parametric assumption which in some cases might be difficult to hold true.

The IPL method [11] outperforms the aforementioned ones in terms of performance estimation but it exhibits significant limitations; it has higher computational cost than TT and WMC/WMCS, collapses when the size of the original dataset is small and it makes the assumption that the learning curve for each classifier can be fitted well by inverse power law.

We propose a new, general method for correcting the bias in Cross-Validation procedures, which works regardless of the data analysis task (e.g. classification, regression) or the structure of the models being involved in it. It has low computational overhead with respect to the Cross-Validation procedure and it was proven, through experiments (see Chapter 6), to produce an almost unbiased expected performance estimate even when the number of training samples is small. Unlike the TT, it is also suitable for correcting the bias of both the LOO-CV and the Hold-Out CV.

4.2 The Bootstrap

The *bootstrap* was formally introduced by Efron in [34] and it is a very general resampling procedure for making statistical inferences when the usual parametric assumptions of a population are questionable or violated. The term "bootstrap" refers to the notion of "pulling oneself up by one's bootstrap" which is a metaphor for "a self-sustaining process that proceeds without external help".

To better understand bootstrapping, one must first understand the concept of a *sampling distribution*. The term *sampling distribution* refers to the probability distribution that would result from the computed values of a statistic (e.g. sample mean, sample median, sample standard deviation) on all possible samples of a given size M of a population of interest. A statistic is assumed to be fixed in the population, but its estimate would vary for different samples of the population. The bootstrap simulates this procedure in order to approximate the sampling distribution by treating the available data as the population. It repeatedly draws observations from the data to create a large number of samples known as the *bootstrap samples*. Each bootstrap sample is drawn randomly with replacement and has the same sample size as the original. The statistic of interest is then computed on each of the bootstrap samples and the resulting distribution is the estimate of the population distribution of that statistic, also known as the *bootstrap distribution* of the statistic. When resampling a dataset of size M with replacement, on average only 0.632 of the original observations will end up being included since the probability of any given observation

not being selected after M samples is $(1 - 1/M)^M \approx e^{-1} \approx 0.368$. Consequently, each of the bootstrap samples will randomly depart from the original. And since the observations in the samples vary randomly, the values of the statistic calculated for each one will be different. For more details on the bootstrap see [14].

4.2.1 Primary applications of Bootstrap

The bootstrap has many applications in statistics and machine learning. For example, one could use the bootstrap to approximate the standard error of a sample estimate of a population parameter θ, to correct its bias, or to compute the confidence intervals of θ. For all these applications the same philosophy of bootstrap is followed: replace the population with the empirical population.

Standard Error Approximation

Let us first introduce some notation. Suppose θ is a population parameter that we need to examine and let $\hat{\theta}$ be an estimator of θ on the basis of a random sample of size M from a population. The empirical standard deviation of a series of bootstrap replications of $\hat{\theta}$ can be used to approximate the standard error $SE(\hat{\theta})$:

$$SE(\hat{\theta}) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}\left(\hat{\theta}_b^* - \hat{\theta}^*\right)^2} \tag{4.1}$$

where

$$\hat{\theta}^* = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}_b^*$$

and $\hat{\theta}_b^*$ is the parameter estimation for the b-th bootstrap sample, and B is the number of total bootstrap samples.

Another resampling technique used for approximating the standard error is the *Jackknife* [35–37], with the bootstrap being a more general method and applicable in the case when the population parameter is the sample median where the Jackknife fails.

Bias Correction

For most sample statistics, the sampling distribution of $\hat{\theta}$ (i.e. a sample statistic) for large sample size M is normal with mean θ (i.e. the corresponding population parameter). However, the mean of the sampling distribution of $\hat{\theta}$ will often be different from θ with a bias equal to $Bias(\hat{\theta}) = E(\hat{\theta}) - \theta$. A bootstrap based approximation of this bias is:

$$\widehat{Bias}_{boot}(\hat{\theta}) = \frac{1}{B}\sum_{b=1}^{B}(\theta_b^* - \hat{\theta}) \tag{4.2}$$

where θ_b^* is the estimate of the b-th bootstrap sample. The corrected estimator is then

$$\hat{\theta}_c = \hat{\theta} - \widehat{Bias}_{boot}(\hat{\theta})$$

Confidence Intervals

When computing a sample statistic, it is desirable to know how well it estimates the underlying population value. *Confidence intervals*, for a given population parameter θ, address this issue by providing a sample based range of values that contain θ with a high probability. This probability is known as *confidence level* $(1 - a)$ and is usually specified to be 95% or 99%. One of the most popular and simple methods for constructing confidence intervals using bootstrap is the percentile method. First, B independent bootstrap samples of size M (the size of the original sample) are drawn. Then, the parameter θ is estimated for each of the bootstrap samples to get $(\theta_1^*, \theta_2^*, ..., \theta_B^*)$. Next, the bootstrap replications of $\hat{\theta}$ are ranked in ascending order so that $(\theta_{(1)}^* \leqslant \theta_{(2)}^* \leqslant ... \leqslant \theta_{(B)}^*)$. The lower and upper confidence bounds are the $B \cdot a/2$-th and $B \cdot (1 - a/2)$-th ordered elements respectively. For example, for $B = 1000$, which is the smallest number of replications that is usually recommended, and $a = 0.05$, the resulting bootstrap percentile confidence interval would be $[\theta_{(25)}^*, \theta_{(975)}^*]$. A confidence stated at a $1 - a$ level can be thought of as the inverse of a significance level, a. For more theoretical details on the bootstrap confidence intervals and differents methods for constructing them, as well as a comparison of them, see [38].

Bootstrap Performance Estimation

In machine learning, a bootstrap based variation of the *Repeated Hold-Out Cross-Validation* (see Section 3.2.1) protocol can be used to perform model assessment. Instead of repeatedly splitting the data D (with $|D| = M$) into two disjoint subsets D_{train} and D_{test} and averaging the results, where $|D_{train}|, |D_{test}| < M$, the bootstrap method will create multiple training sets D_{in} of size M by sampling D with replacement. These training sets, as mentioned before, will consist, on average, of the 0.632 of the original samples. The bootstrap estimate of the expected test error for B bootstrap samples is:

$$\widehat{Err}_{boot} = \frac{1}{B} \sum_{b=1}^{B} \frac{1}{M} \sum_{i=1}^{M} L(y_i, f^{(b)}(x_i, D_{in})) \tag{4.3}$$

where $f^{(b)}(x_i, D_{in})$ is the predictions for the vector of input values x_i of the model trained on the training set D_{in} of the b-th bootstrap sample. Unfortunately, this is not a good estimator of the expected error since the models $f^{(b)}$ that are used to predict the outcome of a vector of input variable values x_i, might have been trained using the data point (y_i, x_i) (i.e. the test data and the train data overlap). When $(y_i, x_i) \notin D_{in}$, then $\frac{1}{M} \sum_{i=1}^{M} L(y_i, f^{(b)}(x_i, D_{in}))$ is similar to the K-Fold Cross-Validation error of equation 3.5, otherwise it is similar to the training error of

equation 3.3. This will lead to the overfitting of the models $f^{(b)}$ and $\widehat{Err}_{boot}$ being considerably biased downward.

An improvement on that is the *Leave-One-Out bootstrap* error estimate, which mimics the K-Fold Cross-Validation one:

$$\widehat{Err}_{LOOboot} = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{|\mathcal{B}^{-i}|}\sum_{b\in\mathcal{B}^{-i}} L(y_i, f^{(b)}(x_i, D_{in})) \tag{4.4}$$

where $\mathcal{B}^{-i}$ is the set of bootstrap samples that do not contain the pair (y_i, x_i). This solves the problem of overfitting, however $\widehat{Err}_{LOOboot}$ is conditioned on less than M observations (see *learning curve bias* in Section 3.2.1) and will be upward biased (i.e. overestimated).

To solve the bias problem, Efron [21] proposed the *0.632 bootstrap estimator*:

$$\widehat{Err}_{0.632boot} = 0.368\widehat{Err}_{train} + 0.632\widehat{Err}_{LOOboot} \tag{4.5}$$

where $\widehat{Err}_{train}$ is the *training error* or *resubstitution estimate* of equation 3.3. Intuitively, the idea is to reduce the bias of the LOO-bootstrap estimate by pulling it toward the training error. However, when $\widehat{Err}_{train}$ is close to zero (i.e. in highly overfitted situations), the estimator will be downward biased [21].

In our knowledge, the bootstrap estimator for selecting a model or assessing its performance is not widely used in the field of machine learning. Kohavi in [4], compares the estimate of K-Fold Cross-Validation and that of the 0.632 bootstrap on a variety of real-world data with different characteristics and finds that "the bootstrap has low variance, but extremely large bias on some problems". He concludes his research by recommending using stratified 10-Fold Cross-Validation. It is also important to note here that the bootstrap method is highly computationally expensive due to the large number of models that need to be trained.

4.3 Bootstrap Bias Correction

In this Section, we present a new, general method for correcting the bias in Cross-Validation procedures, which works regardless of the data analysis task (e.g. classification, regression) or the structure of the models being involved in it. The method takes advantage of the bias correction properties of the bootstrap [14], briefly described in Section 4.2.1. It shares some common ground with the 0.632 bootstrap and with the Nested K-Fold CV but is less expensive than both of them. It has low computational overhead with respect to the Cross-Validation procedure which results from repeatedly sampling the different models' predictions.

The *bootstrap bias correction* (BBC) for Cross-Validation is a simple method for correcting the bias of all Cross-Validation procedures (e.g. K-Fold CV, LOO-CV). It only requires that the predicted values of the models from which we want to choose the best one and assess its performance are known. The method is outlined in Algorithm 5 and is paired with a CV procedure

Algorithm 5 Bootstrap Bias Correction for CV

Input: A training set $D = (x, y) \in X \times Y$, A finite set of learning configurations C, A positive integer K, A positive integer B

Output: A model $\mathcal{M}$, An estimation of performance P_{boot} of model $\mathcal{M}$, The bias $\widehat{Bias}$ of the CV procedure estimate

1: **function:** $BBC(D, C, K, B)$
2: $\mathcal{M}, P, Preds = CV(D, C, K)$
3: draw B bootstrap samples from D of size $|D|$ with replacement: $(D_{in}^{(1)}, D_{in}^{(2)}, ..., D_{in}^{(B)})$
4: **for** $b = 1$ to B **do**
5: $\quad c_b^* = \underset{c \in C}{argmin}\{\bar{E}(y_{in}^{(b)}, Preds(x_{in}^{(b)}, c))\}$ # where $\bar{E}(y, x) = \frac{1}{M}\sum_{i=1}^{M} L(y_i, x_i)$
6: $\quad \hat{e}_b = \bar{E}(y_{out}^{(b)}, Preds(x_{out}^{(b)}, c))$
7: **end for**
8: $P_{boot} = \frac{1}{B}\sum_{b=1}^{B} \hat{e}_b$
9: $\widehat{Bias} = P - P_{boot}$
10: **return** $\mathcal{M}, P_{boot}, \widehat{Bias}$

for model selection whose performance corrects. Essentially, Algorithm 5 outlines a complete method for model selection and evaluation of its performance.

The algorithm begins by calling the K-Fold Cross-Validation procedure of Algorithm 2 (Line 2) which returns the final selected model $\mathcal{M}$, its (biased) performance P as estimated by the procedure, and an array of predictions $Preds$. We assume that $Preds$ is a 2-dimensional array of size $|D| \times |C|$, whose columns correspond to the models that were produced using the set of configurations C (see Section 3.4) in the CV procedure and its rows correspond to the predicted values of these models for each sample x_i. In order for the K-Fold CV procedure to agree with our bias correction method, we use a slightly different way to compute the CV performance estimate; instead of performance P being calculated as the average performance of the models that were trained with the best found configuration c^* over the K folds (see Algorithm 2), now, we pool together all the predictions for each configuration and the performance P_{pool} is just the average performance over all predictions for the best configuration c^*. When the number of samples in each fold is the same, and for some metrics (e.g. accuracy), $P = P_{pool}$. However, when the folds are of different size, or the AUC metric is optimized, $P_{pool} \simeq P$.

Next, B bootstrap samples of size $|D|$ (i.e. the number of data points) are drawn with replacement (Line 3). We use $D_{in}^{(b)}$ to denote the data points that are contained in the b-th bootstrap sample (around 0.632 of the original samples) and $D_{out}^{(b)}$ to denote the set of the ones that are not. Subsequently, we use $x_{in}^{(b)}$ and $y_{in}^{(b)}$ to denote the set of input variable values and the set of their respective output values of the samples that are included in the b-th bootstrap.

Then, for each bootstrap sample $D_{in}^{(b)}$ we find the configuration c_b^* that minimizes the loss function L and we estimate its performance on $D_{out}^{(b)}$ (Lines 4-7). $Preds(x_{in}^{(b)}, c)$ represents the predictions of the model(-s) trained with configuration c for the set of samples $x_{in}^{(b)}$. Essentially,

c and $x_{in}^{(b)}$ act as indices to the predictions matrix $Preds$. We use this notation to point out that there is no need for another model to be trained or even for predictions to be produced. The BBC method only needs the array of predictions already produced by the respective Cross-Validation procedure. The estimated performance of the best model is then the average of the performances of the best selected models for each bootstrap sample (Line 8). The model returned by the method is the one that has the best performance averaged over all pooled predictions. In the case of having accuracy as the metric of performance and assuming same split of the data in folds, it is the same as the ones returned by K-Fold Cross-Validation (averaging over folds), TT and the Nested K-Fold Cross-Validation, but with different estimation of the performance (Line 10).

In conclusion, the BBC method provides an almost unbiased estimate of the performance of the final selected model. It is faster than the Nested K-Fold Cross-Validation protocol and it is designed to work with all kinds of CV procedures and also with small sample size (where TT fails).

4.4 Computing Confidence Intervals

When assessing the performance of a model, it is desirable to know how well the estimate reflects the true performance. *Confidence intervals* provides us with a sample based range of values that contain the true performance with high probability. One of the most popular and simple methods for constructing confidence intervals using bootstrap is the percentile method explained in Section 4.2.1.

In the BBC procedure B bootstrap samples are independently drawn from the predictions' array (essentially), and for each of them the best configuration is found and its performance is estimated. In order to use the percentile method for the construction of confidence intervals, the only extra work that needs to be done is to rank the bootstrap replications of performance of the BBC method in ascending order. Then, the lower and upper confidence bounds are the $B \cdot a/2$-th and $B \cdot (1 - a/2)$-th ordered elements respectively, where $(1 - a)$ is the confidence level.

We are still evaluating how well fit this method is for this purpose, but it seems to provide good results in our experiments. We also plan to test other, more reliable methods for calculating confidence intervals such as the BC_a method [39].

Chapter 5

Bootstrap-Based Dropping of Under-Performing Configurations

In this Chapter we introduce a method, based on the bootstrap, for eliminating under-performing configurations early in the K-Fold Cross-Validation procedure in order to speed it up. Section 5.2 presents the way the method is incorporated within K-Fold Cross Validation and describes the approach to identifying under-performing configurations.

5.1 Dropping of Under-Performing Configurations

K-Fold Cross Validation has become a de-facto standard in machine learning for model selection and evaluation and it is commonly paired with grid search. When the space of hyper-parameter values is large, K-Fold Cross Validation can become computationally expensive and thus prohibitive to be applied in full.

We present a procedure, based on bootstrap testing, for dropping under-performing configurations early within a Cross-Validation procedure in order to speed up its execution time. We call our method *Bootstrap-based Early Dropping* (hereafter *BED*), and we have studied its behaviour through extensive experimentation both on simulated and real data (see Chapter 6).

To the best of our knowledge, the only work that focuses on the speeding up of the learning process by specifically eliminating, early in the CV procedure, under-performing configurations is the one by Krueger et al. [20]. Their method is called *Fast Cross-Validation via Sequential Testing* (CVST), and it uses nonparametric testing together with sequential analysis to choose the best performing configuration on the basis of linearly increasing subsets of data.

5.2 Bootstrap-Based Dropping of Under-Performing Configurations

BED is incorporated within the K-Fold Cross Validation procedure and it is outlined in Algorithm 6. We use $f(\cdot, D_{train}, c)$ to denote the model trained on dataset D_{train} with configuration c, and $f(D_{test}, D_{train}, c)$ to denote the output (i.e. the predictions) of that model when applied on D_{test}.

The K-Fold CV with BED procedure starts with every configuration $c \in C$ being active, and discards (removes from C) under-performing configurations at each iteration i of the procedure, based on their performance on the last i iterations. More specifically, at each iteration i, the following steps are executed. First, for each configuration $c \in C$ a model is trained on $D \backslash \{F_i\}$, and their predictions for the test fold F_i are stored in a 2-dimensional array $Preds_i$ (Lines 4-6). Consequently, each column of $Preds_i$ corresponds to a configuration in C and each of its rows corresponds to a data sample in F_i. Then, the union of the arrays $Preds_k$ for $k = 1, ..., i$ is stored in $Preds$; that is all the produced predictions up to iteration i (Line 7). Again, the columns of $Preds$ correspond to the configurations in C, but its rows contain the predictions for all the test folds F_k for $k = 1, ..., i$. Next, the array of predictions $Preds$, along with the corresponding true output (labels) y, the set C, the desirable number of bootstraps B, and a threshold t are passed to the BED procedure that determines which of the configurations in C will be eliminated before the next iteration. Finally, the configurations that were found to be under-performing are removed from the set of configurations C in Line 10.

After all K iterations of the K-Fold CV procedure have been executed, the final, best configuration is chosen among the ones that are left in the set C, and its performance is estimated (Lines 12-17). In Algorithm 6 this is performed by finding the configuration c^* which minimizes the mean loss over all predictions. This way, the BBC procedure for bias correction (see Section 4.3) can be directly applied. One could also calculate the mean loss over the predictions of each fold F_i independently, and then average the results, as is usually the case for K-Fold CV (see Section 3.4.2).

The *BED* procedure is detailed in Algorithm 7. It takes as input a 2-dimensional array of predictions $Preds$ of size $M \times N$ and the corresponding true labels y (a vector of M values), a set of configurations $C = \{c_1, c_2, ..., c_N\}$, the number B of bootstraps to perform, and a threshold t. First, the configurations in C are sorted in ascending order based on their mean error on the predictions in $Preds$ in order to get $C^{(*)} = \{c^{(1)}, c^{(2)}, ..., c^{(N)}\}$. Consequently, configuration $c^{(1)}$ is the currently best performing one. Next, it constructs B bootstrap samples (i.e. multiple instances of $Preds$) by sampling the rows of $Preds$ with replacement. Then, for each configuration $c^{(i)} \in C^{(*)}, i = 2, ..., N$, we calculate the probability of it being statistically different (worse) than $c^{(1)}$ (the best one) in terms of performance. This is equal to the proportion of bootstraps for

Algorithm 6 K-Fold Cross-Validation with Bootstrap Dropping

Input: A training set $D = (x, y) \in X \times Y$, A finite set of learning configurations C, A positive integer K, A positive integer B, A threshold t

Output: A model $\mathcal{M}$, An estimation of performance P of model $\mathcal{M}$, A configuration c^*

function: $BD(D, C, K, B, a)$
randomly partition D into K disjoint subsets $F_i, i = 1..K$ of approximately equal size
for $i = 1$ to K **do**
  for each configuration c ∈ C **do**
    $Preds_i = f(F_i, D \backslash \{F_i\}, c)$
  end for
  $Preds = \bigcup_{k=1}^{i} Preds_k$
  $y = y_i \in \bigcup_{k=1}^{i} F_k$
  $C_{drop} = BED(y, Preds, C, B, t)$
  $C = C \backslash C_{drop}$
end for
for each configuration c ∈ C **do**
  $\hat{e}_c = \frac{1}{M} \sum_{i=1}^{M} L(y_i, Preds(x_i, c))$
end for
$c^* = \underset{c \in C}{argmin}\{\hat{e}_c\}$
$P = \underset{c \in C}{min}\{\hat{e}_c\}$
$\mathcal{M} = f(\cdot, D, c^*)$
return $\mathcal{M}, P, c^*$

which $c^{(1)}$ had better performance than the i-th configuration:

$$P(\hat{e}_{c^{(1)}} < \hat{e}_{c^{(i)}}) = \frac{1}{B} \sum_{b=1}^{B} 1(\hat{e}^{(b)}_{c^{(1)}} < \hat{e}^{(b)}_{c^{(i)}}) \tag{5.1}$$

where $1(condition)$ is equal to 1 if *condition* is true, and 0 otherwise. This is performed for all the configurations $c^{(i)}$ and the ones that satisfy:

$$P(\hat{e}_{c^{(1)}} < \hat{e}_{c^{(i)}}) \geqslant t \tag{5.2}$$

are those that will be eliminated.

For the number of bootstraps B we suggest that 1000 is a good enough value to ensure an almost accurate estimate of equation 5.1. t, is a threshold on the probability defined in equation 5.1. Essentially, it determines the number of bootstraps $\widehat{B}$ out of B, that $c^{(1)}$ needs to perform better than another configuration $c^{(i)}$, in order for $c^{(i)}$ to be dropped. For example, for $B = 1000$ and $t = 0.95$, $\widehat{B} = 950$. The higher the value of t is, the more conservative the dropping procedure becomes (the more unlikely it becomes to drop a configuration).

We empirically show that BED significantly speeds up the running time of the K-Fold Cross-Validation procedure since the number of the models that are trained throughout K-Fold CV are reduced at least by 50%. We cannot explicitly measure the performance of the method in terms

Algorithm 7 Bootstrap-based Early Dropping

Input: A vector of output values y, An array of predictions $Preds$ of size $M \times N$, A set of configurations $C = \{c_1, c_2, ...c_N\}$, A positive integer B, A threshold a

Output: A set C_{drop} of under-performing configurations

1: **function:** $BED(y, Preds, C, B, t)$
2: sort the configurations in C by their mean error in ascending order to get $C^{(*)} = \{c^{(1)}, c^{(2)}, ..., c^{(N)}\}$ # $c^{(1)}$ has the best performance
3: construct B bootstrap samples from $Preds$ of size $M \times N$ with replacement $(Preds^{(1)}, Preds^{(2)}, ..., Preds^{(B)})$ # by sampling rows of $Preds$
4: $C_{drop} = \varnothing$
5: **for** $i = 2$ to N **do** # for each $c^{(i)}, i = 2, ..., N$
6: **for** $b = 1$ to B **do** # for each bootstrap sample $Preds^{(b)}$
7: $\hat{e}^{(b)}_{c^{(1)}} = \frac{1}{M} \sum_{k=1}^{M} L(y_k^{(b)}, Preds^{(b)}(x_k, c^{(1)})$
8: $\hat{e}^{(b)}_{c^{(i)}} = \frac{1}{M} \sum_{k=1}^{M} L(y_k^{(b)}, Preds^{(b)}(x_k, c^{(i)})$
9: **end for**
10: $s = \frac{1}{B} \sum_{b=1}^{B} 1(\hat{e}^{(b)}_{c^{(1)}} < \hat{e}^{(b)}_{c^{(i)}})$ # 1(condition = false/true) = 0/1
11: **if** $s \geqslant t$ **then**
12: $C_{drop} = C_{drop} \bigcup \{c^{(i)}\}$
13: **end if**
14: **end for**
15: **return** C_{drop}

of running time, since we had to ran the experiments on a few different machines with different characteristics.

We have not compared BED to CVST in practice, and therefore we cannot make claims that concern the running time and model selection properties of the two methods. However, there are some points that make our method more appealing to use than CVST. The CVST method uses linearly increasing subsets of data as training sets. If the sample size of the original data is small, then CVST will probably not be applicable. Our method uses the bootstrap as a statistical test in order to detect differences in performance between configurations which is general and can be used independently of the learning task. The CVST method employs a different statistical test for each task (e.g. the Friedman and the Cochran's Q tests for regression and classification respectively). Finally, the CVST method has a total number of four parameters that need to be set by the user in contrast to BED which only has two: the number of bootstraps B and the threshold t.

5.2.1 Discussion

A few comments on BED. It is a heuristic procedure mainly with focus on computational efficiency, not statistical theoretical properties. Ideally, the null hypothesis to test for each configuration c would be the hypothesis that c will be selected as the best configuration at the end of the KCV procedure, given a finite number of folds remain to be considered. If this null hypothesis

is rejected for a given c, it should be dropped. Each of these hypotheses for a given c has to be tested in the context of all other configurations that participate in the KCV procedure. In contrast, the heuristic procedure we provide essentially tests each configuration $c^{(i)}$ in isolation. For example, it could be the case during bootstrapping, configuration $c^{(i)}$ exhibits a significant probability of a better loss than $c^{(1)}$ (not dropped by our procedure), but it could be that in all of these cases, it is always dominated by some other configuration $c^{(j)}$. Thus, the actual probability of being selected as best in the end maybe smaller than the percentage of times it appears better than $c^{(1)}$.

In addition, our procedure does not consider the uncertainty (variance) of the selection of the current best method $c^{(1)}$. Perhaps, a double bootstrap procedure would be more appropriate in this case [40] but any such improvements would have to also minimize the computational overhead to be worthwhile in practice.

The computational cost is overcome by the strong theoretical properties of the bootstrap. Our simulation based preliminary results have showed that McNemar's test [41] did not work well for small sample sizes. The obvious way would be to bootstrap the McNemar's test statistic. The big advantage of this general bootstrap method is that it can be employed with any type of classification and regression task without the necessity of imposing any extra (possibly unrealistic) assumptions and inherited by the need of performing appropriate tests.

A faster, again bootstrap based, early dropping method would be via ordering of the cumulatively aggregated performances of the models at each fold and performing the aforementioned bootstrap dropping at each step. The speed-up factor is considerable, nevertheless, the proportion of computational time of the CV protocol spent in this procedure is rather small and makes no difference in the overall time.

Chapter 6

Experiments and Evaluation

In this Chapter we evaluate the Bootstrap Bias Correction Method (BBC) and compare it against the K-Fold Cross-Validation (KCV), the Tibshirani and Tibshirani (TT), and the Nested K-Fold Cross-Validation (NCV) estimates of performance. We also evaluate the Bootstrap-based Early Dropping method (BED) for eliminating under-performing configurations along the way of Cross-Validation. In Section 6.1 the simulation studies are presented. The experimental evaluation using real data sets is presented in Section 6.2.

6.1 Simulation Studies

In order to assess the quality and investigate the properties of our bias correction method (BBC), and compare it to other estimates of performance, we conducted extensive simulation studies. We also test BED, our method for eliminating under-performing configurations within Cross-Validation.

Without loss of generality we examined the case of binary classification, where the target variable Y takes on only two discrete values (e.g. $Y \in \{0, 1\}$). We studied a variety of settings for different combinations of values for the sample size M, the number of models N, and the true performance P_{true}. For the classification task we used the percentage of correctly classified samples (i.e. classification accuracy) as the measure of performance. M and N take values from $\{50, 100, 200, 300, 500, 1000\}$ and $\{50, 100, 200, 300, 500, 1000, 2000\}$, respectively. We assume that the true classification accuracy, P_{true}, follows a beta distribution $Be(a, b)$ with the mean μ taking values from $\{0.6, 0.7, 0.8, 0.9\}$. For each P_{true}, 5 pairs of parameters were used, controlling the variance of the beta distribution, ranging from high to low. Figure 6.1 shows the Beta distributions from which the simulated predictions were produced.

For each setting an $M \times N$ array of predictions is constructed, with the given classification accuracy, where the rows correspond to data instances/samples and the columns correspond to the models inferred from different configurations (i.e. different combinations of algorithms and hyper-parameter values for each step of the learning procedure).

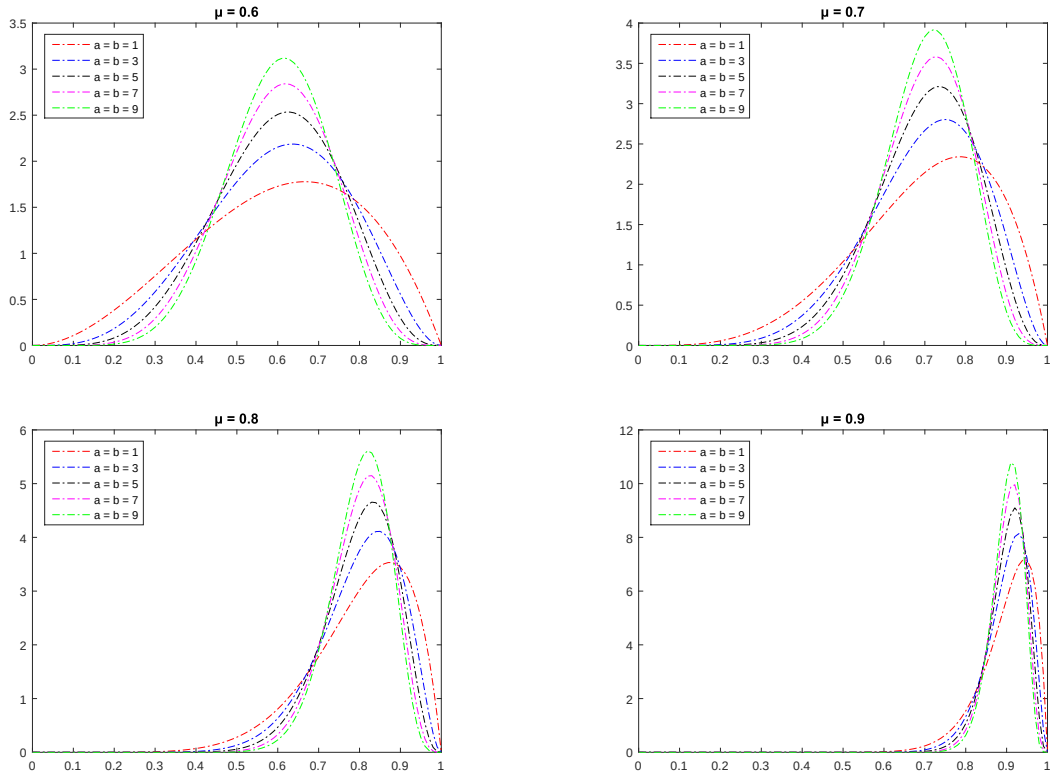

Figure 6.1: Density of the Be(a, b) distribution for the parameters used in the simulation studies. The parameters are such that μ = a/(a + b) = (0.6, 0.7, 0.8, 0.9).

The number of bootstraps for the BBC method was set to **500**. The number of bootstraps B and the threshold t for the BED dropping method were set to **1000** and **0.99** respectively. In all cases, the results were averaged over 500 repetitions.

We compared the K-Fold Cross Validation (KCV), Bootstrap Bias Correction (BBC), Nested K-Fold Cross-Validation (NCV), Tibshirani and Tibshirani (TT), Bootstrap-based Early Dropping (BED), and the BED with BBC methods in terms of bias of the estimated performance (Section 6.1.1). Since we use classification accuracy as the measure of performance and every fold of K-Fold Cross Validation contains the same number of data instances, the KCV, BBC, NCV and TT protocols all select the same model. For the dropping method BED, which might select a different model than the aforementioned methods, we also compute the model selection error (Section 6.1.2).

6.1.1 Bias Correction Estimation

The bias of the estimation is computed as $\hat{Bias} = \hat{P} - P_{true}$, where $\hat{P}$ and P_{true} denote the estimated and the true performance of the selected model, respectively. The target value of $\hat{Bias}$ is 0, indicating the absence of bias. A positive bias indicates a lower P_{true} than the one estimated by the corresponding performance estimation method and implies the method is optimistic.

Figures 6.2-6.5 show the average bias, over 500 repetitions of the simulations for the different

settings, for the estimates of each of the methods examined for a true classification accuracy P_{true} equal to 0.6, 0.7, 0.8, and 0.9 respectively. For each P_{true}, only the most challenging cases are presented. These are the cases where the simulated data are produced from a Beta distribution with low variance (see Figure 6.1).

It is clear that the K-Fold Cross Validation (KCV) estimate of performance is optimistically biased. The smaller the sample size, the proportionally higher the bias is. The reverse holds for the number of models; the greater the number of models, the higher the bias is (in an almost equal way up to a certain sample size). For higher rates of true performance the bias is lower even for the small samples. For example, in Figure 6.2 (60% true accuracy), the bias of KCV for sample size equal to 50 and number of models equal to 2000 is more than 8%, whereas in Figure 6.5 (90% true accuracy) it is a little less than 6%.

The TT estimate of performance seems to be unsuitable for small sample sizes. Its bias greatly varies with the number of models; it either corrects the bias by only a small amount or it overestimates it. For larger sample sizes, it systematically over-corrects the bias.

BBC always provides slightly conservative estimates of performance (i.e. it slightly over-corrects the bias of KCV). As the sample size increases, the bias tends to zero. Compared to TT, it seems to be a lot more suitable for smaller sample sizes and produces less biased estimates. For higher rates of true classification accuracy the bias gets even smaller.

NCV exhibits the smallest bias which slightly varies (being positive or negative) for sample size $\leqslant 200$. NCV, however, is a lot more computationally expensive than BBC and TT since the number of models that need to be trained depends quadratically to the number of folds K.

BED exhibits similar results to those of KCV. BED with BBC (BED-BBC) produces the best results, in terms of bias, performance and computational cost. For sample size $\geqslant 100$ it has the smallest bias in absolute value (together with NCV), and it approaches zero at a much faster rate than BBC.

All in all, NCV, BBC and BED-BBC provide the best results. NCV has the smallest bias but is computationally expensive. BBC and BED-BBC, systematically, have negative bias (i.e. they are more conservative in terms of performance estimation) with the latter approaching zero in a much faster pace.

6.1.2 Model Selection Error

In order to evaluate the model selection properties of KCV and BED, we compute the following quantity which we call *model selection error* $P_{true}^{best} - P_{true}^{selected}$. P_{true}^{best} denotes the true performance of the best model and $P_{true}^{selected}$ denotes the true performance of the selected model.

Figure 6.6 shows the model selection error for KCV and BED, which are the methods that, in the case of the simulations, select different models. We notice that BED has the same or slightly greater (no more than 0.005 points of accuracy) model selection error than KCV which means that they select models of the same or similar performance.

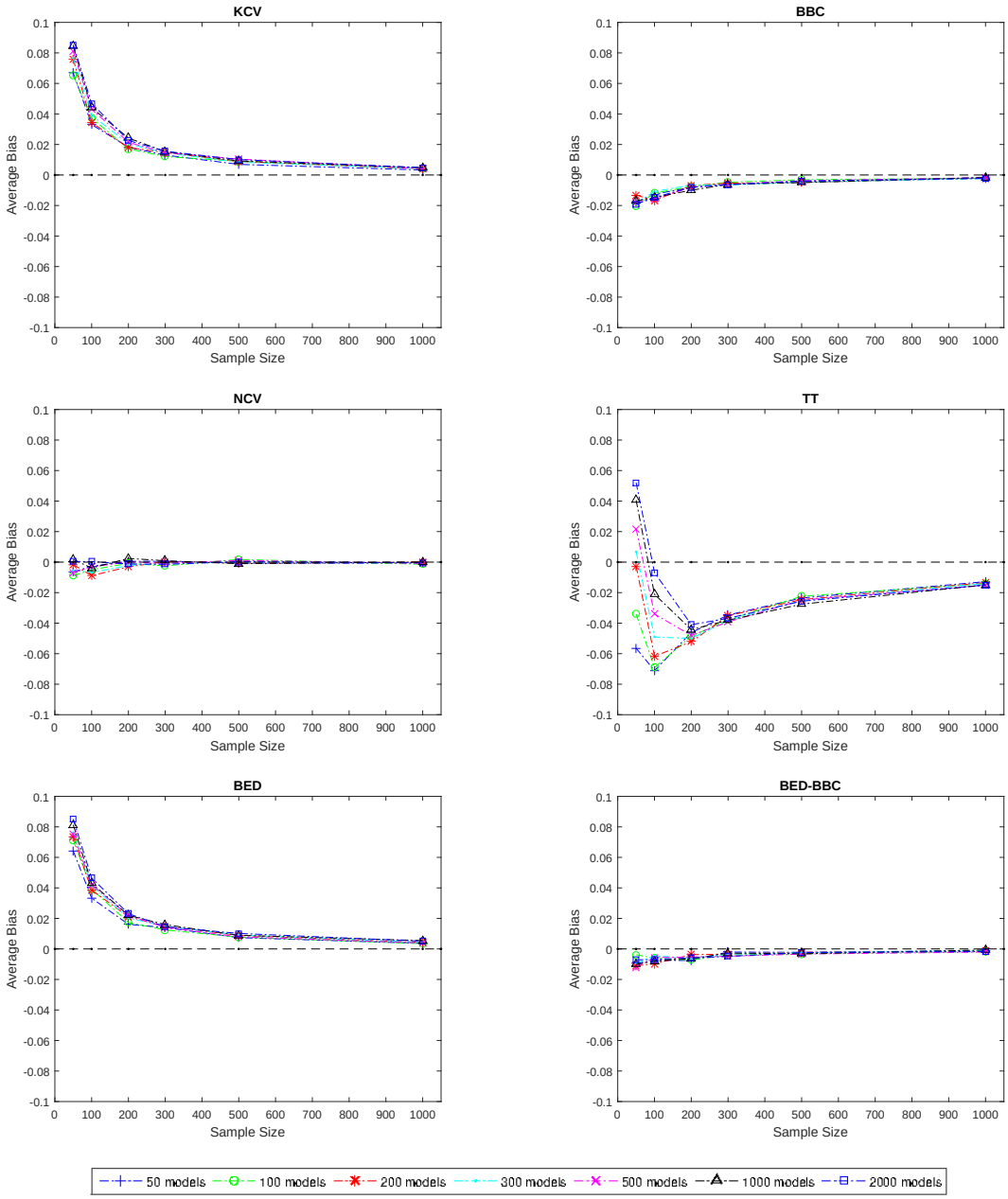

Figure 6.2: Average performance bias for the estimates of KCV, BBC, NCV, TT, BED, and BED-BBC for 60% true classification accuracy. KCV and BED are clearly optimistic for sample size $\leqslant 300$. BBC is slightly conservative. TT's bias greatly varies for sample size $\leqslant 100$ with the number of models and overcorrects for sample size $\geqslant 200$. NCV and BED-BBC exhibit the smallest bias, especially for sample size $\leqslant 100$.

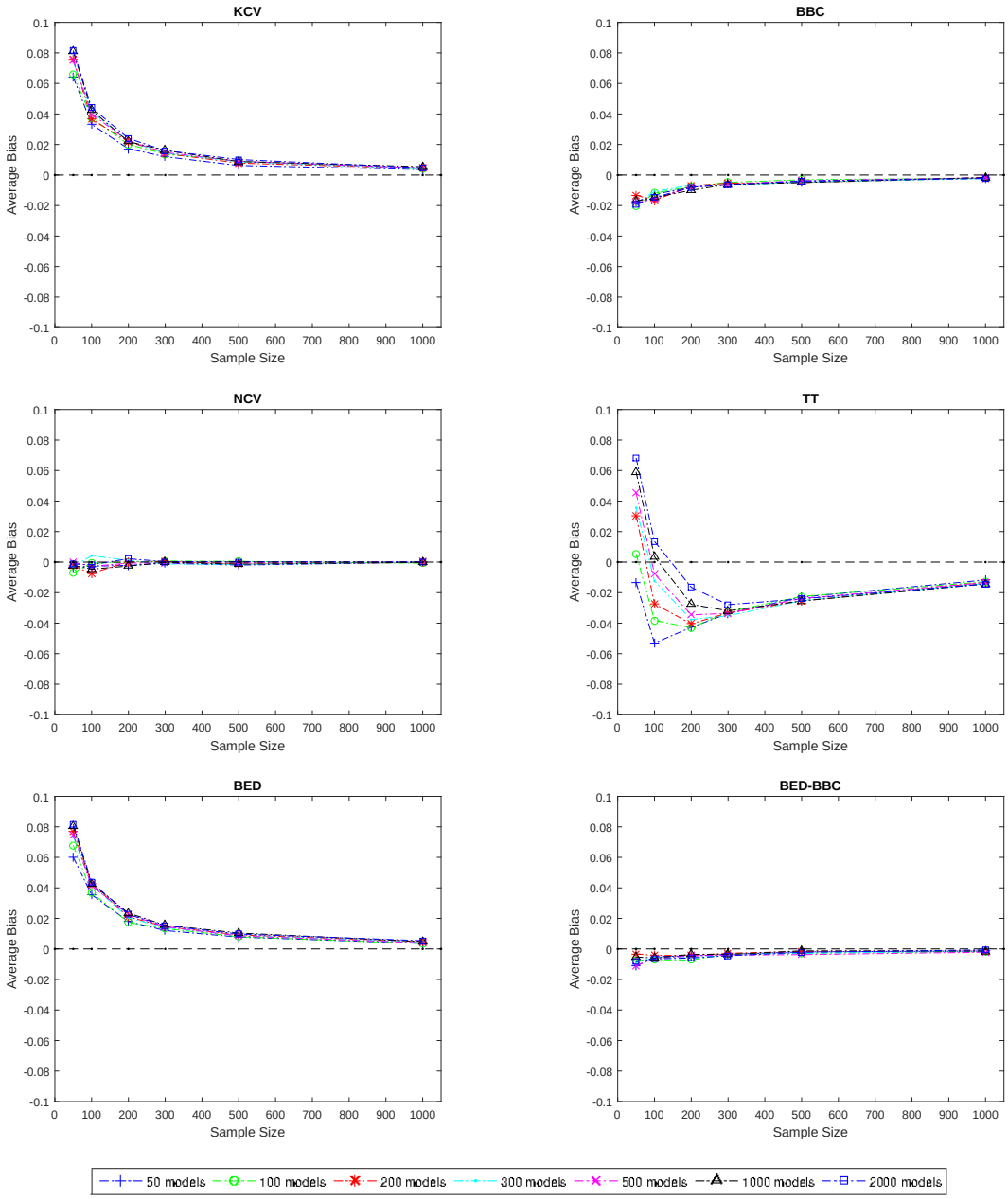

Figure 6.3: Average performance bias for the estimates of KCV, BBC, NCV, TT, BED, and BED-BBC for 70% true classification accuracy. KCV and BED are clearly optimistic for sample size $\leqslant 300$. BBC is slightly conservative. TT's bias greatly varies for sample size $\leqslant 100$ with the number of models and overcorrects for sample size $\geqslant 200$. NCV and BED-BBC exhibit the smallest bias, especially for sample size $\leqslant 100$.

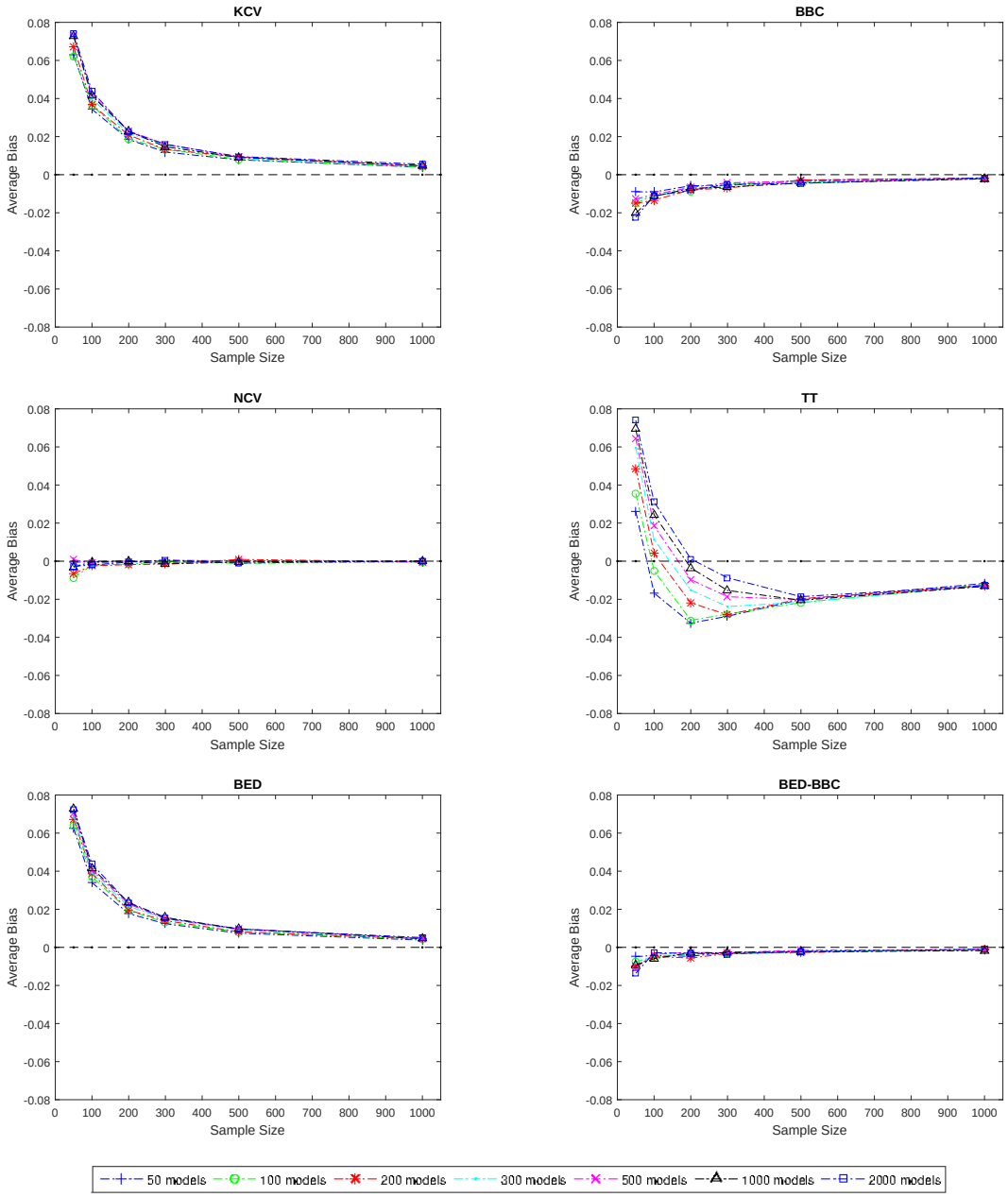

Figure 6.4: Average performance bias for the estimates of KCV, BBC, NCV, TT, BED, and BED-BBC for 80% true classification accuracy. KCV and BED are clearly optimistic for sample size $\leqslant 300$. BBC is slightly conservative. TT's bias varies with the number of models and overcorrects for sample size $\geqslant 500$. NCV and BED-BBC exhibit the smallest bias, especially for sample size $\leqslant 100$.

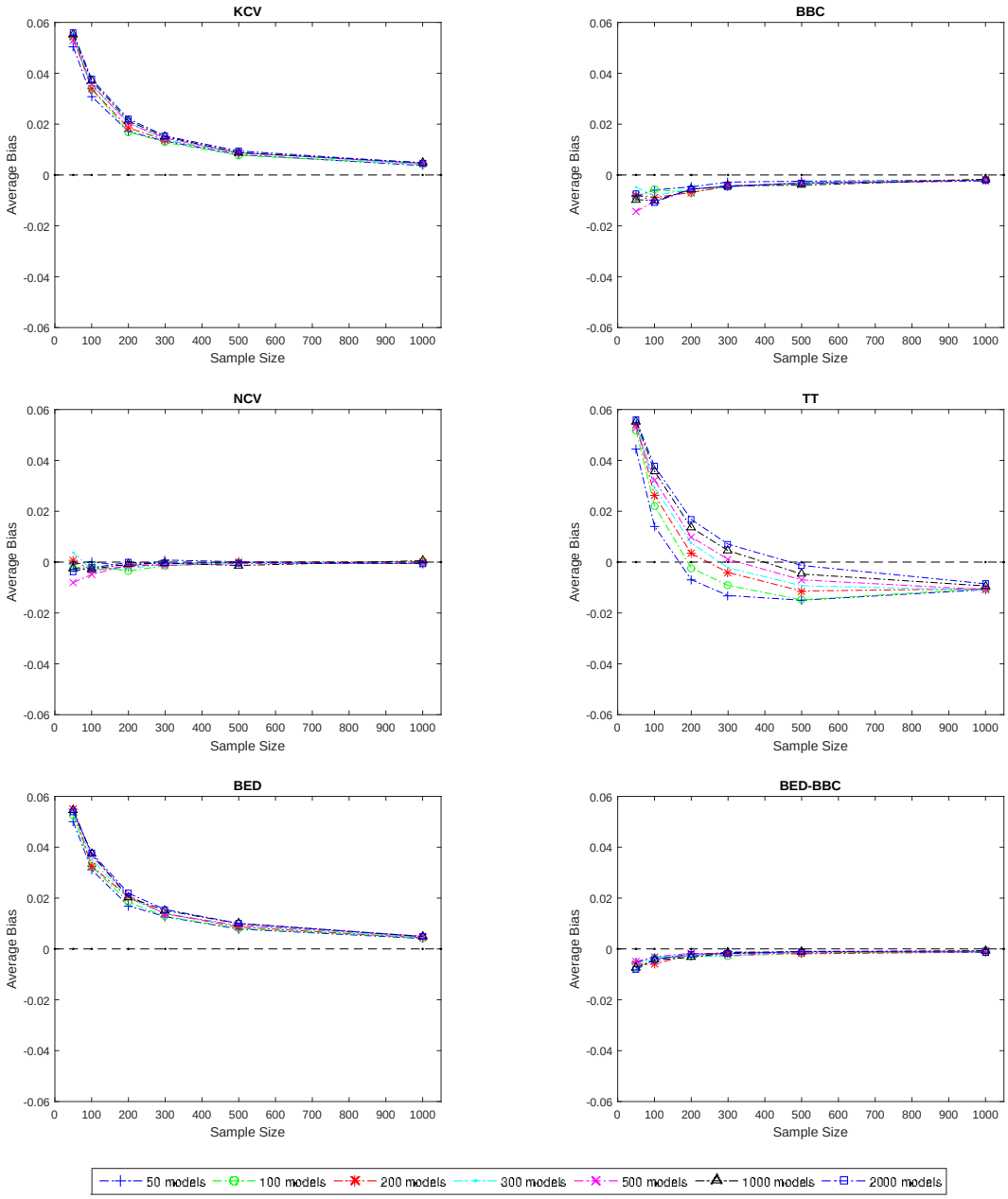

Figure 6.5: Average performance bias for the estimates of KCV, BBC, NCV, TT, BED, and BED-BBC for 90% true classification accuracy. KCV and BED are clearly optimistic for sample size $\leqslant$ 500. BBC is slightly conservative. TT's bias varies with the number of models. NCV and BED-BBC exhibit the smallest bias, especially for sample size $\leqslant$ 100.

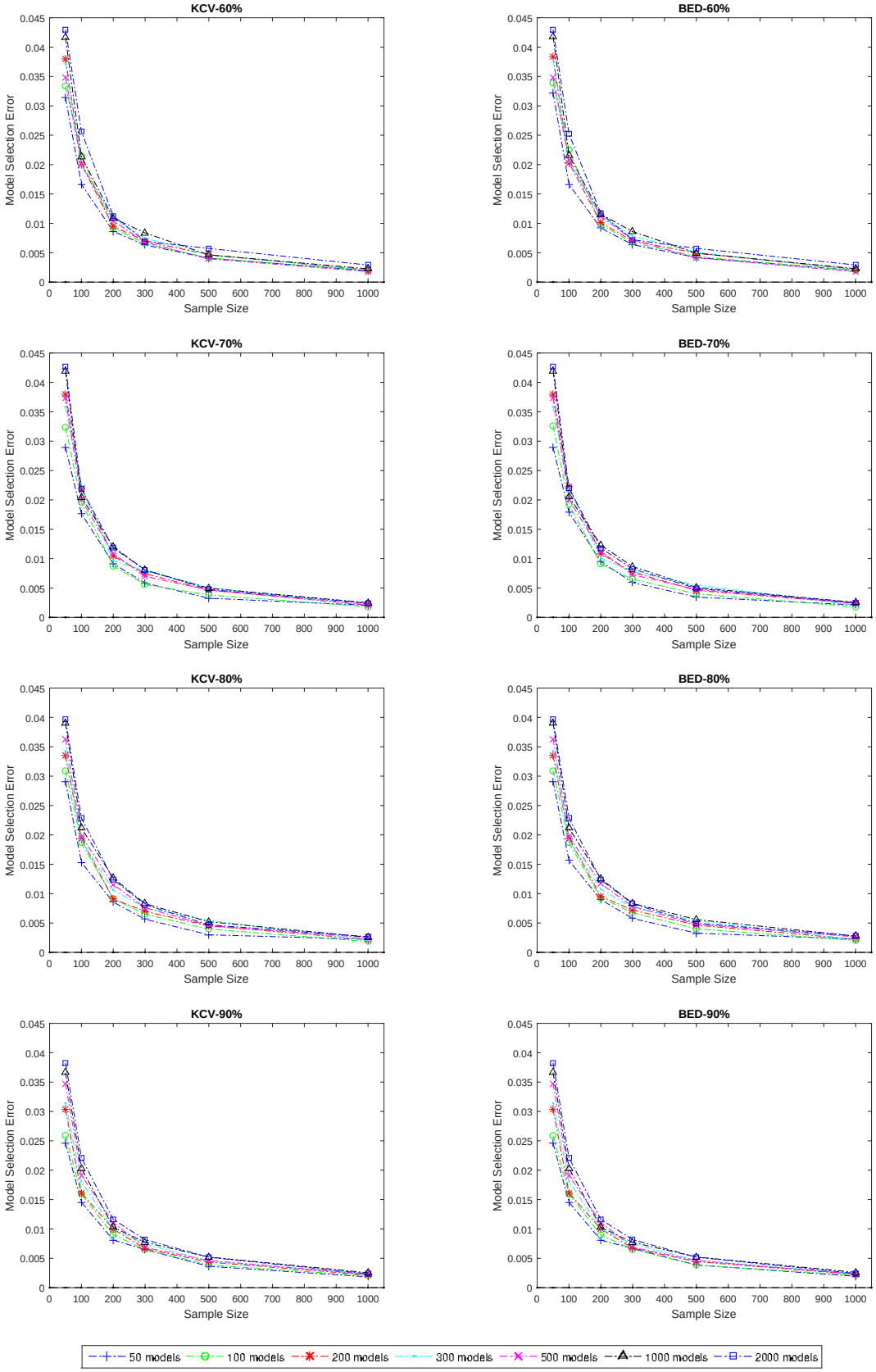

Figure 6.6: Model selection error for KCV and BED for true classification accuracy $\in \{60, 70, 80, 90\}\%$. BED has the same or slightly greater (no more than 0.005 points of accuracy) model selection error than KCV. The error decreases with higher rates of true classification accuracy.

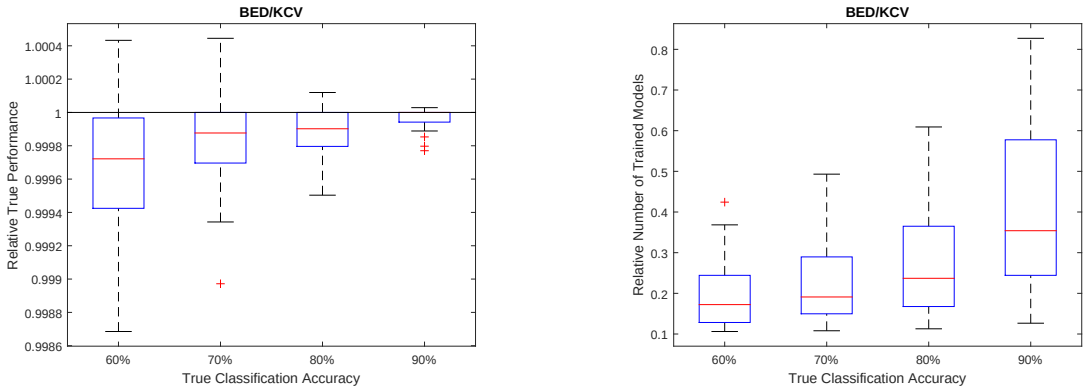

Figure 6.7: Boxplots of the relative true performance (left) and the relative number of trained models (right) for true classification accuracy $\in \{60, 70, 80, 90\}\%$ for all sample sizes ($\{50, 100, 200, 300, 500, 1000\}$) and number of configurations ($\{50, 100, 200, 300, 500, 1000, 2000\}$) for the BED and KCV methods. There is a negligible to no effect on performance when using the BED method. However, the number of models that are trained is greatly reduced.

6.1.3 Relative Performance and Number of Trained Models

Apart from the model selection error, we also compared the KCV and BED (BED-BBC) methods in terms of the true performance of the models that they select. We also evaluated the speed up factor that we get with the BED method relatively to KCV in terms of the total number of trained models. Figure 6.7 shows the boxplots of the relative average true performance of the final selected models of the BED and KCV methods (on the left), and the relative average number of trained models for the two methods (on the right) for true classification accuracy $P_{true} \in \{60, 70, 80, 90\}\%$, for all sample sizes ($\{50, 100, 200, 300, 500, 1000\}$) and number of configurations ($\{50, 100, 200, 300, 500, 1000, 2000\}$). We notice that there is a negligible to no effect on performance when using the BED method. However, the number of models that are trained is greatly reduced. The results vary with true classification accuracy. For $P_{true} = 90\%$ more models are trained (less are dropped) and the loss in performance is lower in comparison to $P_{true} = 60\%$, which is expected since in the former case the majority of models are "good" predictors.

6.2 Experiments on Real Datasets

6.2.1 An Automated Pipeline for Supervised Machine Learning

In order to evaluate the performance of both the bootstrap-based bias correction (BBC) and the bootstrap-based early dropping method (BED), we built an automated tool for supervised machine learning. The tool will automatically perform a complete analysis of a given dataset, offering numerous options to the user in regards to learning tasks, data preprocessing and feature selection methods, learning algorithms, metrics to use for optimization and performance estimation protocols. Its modular architecture allows for the easy incorporation of additional methods for all the aforementioned categories.

Data Preprocessing

Data preprocessing methods are methods that change the feature values and that are always applied whenever they can. Such methods include imputation of missing values, standardization of numerical input variables and binarization of categorical input variables (i.e. the levels of a categorical variable are coded as a collection of binary variables). As suggested in [42], in the case that the training algorithm is the ϵ-SVM, we also scale the values of the target variable, in order to bound the effective range of the values of the hyper-parameter ϵ, making it thus easier to choose values for ϵ.

Feature Selection

Feature selection (FS), also known as variable selection, is the process of identifying the most salient features for learning, allowing thus the learning algorithm to focus on those aspects of the data most useful for analysis and future prediction.

By removing as much irrelevant and redundant information (features) as possible, the dimensionality of the data can be significantly reduced thus allowing the learning algorithms to operate more effectively and faster. Feature selection can also improve the predictive performance of learned models which will usually be simpler and more easily understood and interpreted.

The automated tool, for the time being, offers the options of using all the features for training (no FS method applied) and performing feature selection using the SES [43] and the LASSO [44] algorithms. Feature selection with LASSO works by fitting the LASSO to the data and choosing features corresponding to non-zero model coefficients.

We have also implemented group-SES which is a variation of the SES algorithm to allow predefined groups of variables to be selected into or out of a model together. This is useful in the case that there exist categorical variables in the data. Group-SES can ensure that all the binary variables encoding a categorical variable are either included or excluded from the training procedure.

The aforementioned feature selection methods fall into the the category of *filters*. Filter type methods select variables regardless of the learning algorithm. They are, essentially, a preprocessing step to the learning procedure. These methods are particularly effective in computation time since they do not require re-execution on a specific dataset for different learning algorithms. The results can be cached and used repeatedly.

Model Selection and Assessment

The automated tool offers a variety of methods for model selection and assessment such as Hold-Out Cross-Validation, K-Fold Cross-Validation (KCV), Nested K-Fold Cross-Validation, KCV with dropping (BED), KCV with the Tibshirani and Tibshirani bias correction method, KCV with the bootstrap-based bias correction method (BBC), BED with BBC, as well as the stratified and repeated versions of all the aforementioned methods.

Currently, the tool provides the options of conducting either a classification or a regression analysis. For both tasks, the learning algorithms involved in the analysis include Random Forests as implemented in Matlab 2015b, SVMs as implemented in the libsvm library [45], LASSO, Decision Trees and Logistic Regression as implemented in Matlab 2015b. For the classification task the metrics of performance that can be optimized are accuracy, balanced accuracy, AUC, precision, recall and the F1 measure. For regression the metrics include the R-squared score, mean squared error (MSE), and mean absolute error (MAE).

6.2.2 Experimental Set-Up

The experimental set-up is similar to the one used by Tsamardinos et al. in [10].

Datasets

The datasets that were used for the experiments are from the first round of the ChaLearn AutoML challenge [46]. The organizers of the challenge mention about the datasets that "the domains of application are very diverse and are drawn from: biology and medicine, ecology, energy and sustainability management, image, text, audio, speech, video and other sensor data processing, internet social media management and advertising, market analysis and financial prediction". Table 6.1 summarizes the datasets' characteristics. The ratio of the positive and negative classes for all the datasets is 50:50.

Each dataset D was spit into two subsets, D_{pool} which consisted of 30% of the data instances/points of D and $D_{holdout}$ which consisted of the remaining 70% of the data in D. D_{pool} was used to sample (without replacement) 10 subsets for each of the sample sizes in $\{20, 40, 60, 80, 100, 500\}$ leading in the creation of $5 \times 10 \times 6 = 300$ sub-datasets in total. $D_{holdout}$ was used to estimate the true performance of the final, selected model of each of the methods tested.

Table 6.1: Datasets Used; $|D_{pool}|$ refers to the portion of the datasets (30%) from which the sub-datasets were sampled and $|D_{holdout}|$ to the portion (70%) from which the true performance is estimated.

Name	#Samples	#Variables	$\|D_{pool}\|$	$\|D_{holdout}\|$
christine	5418	1636	1625	3793
jasmine	2984	144	895	2089
philippine	5832	308	1749	4082
madeline	3140	259	942	2198
sylvine	5124	20	1537	3587

Model Selection and Performance Assessment

The set of configurations C (i.e. the search grid) is constructed, by the automated tool, based on the learning task and the characteristics of the dataset to be analyzed. For the purpose of the experiments, and for the classification task that we examined, we restricted the search grid to a few hundreds of configurations, in order to be able to test all the different methods (especially NCV which is computationally expensive). The preprocessing methods were used when they could be applied. For feature selection we only included the SES algorithm and we also tested the case of no feature selection. The learning algorithms that we examined are Random Forests, SVMs, and LASSO.

The hyper-parameters that were tested for SES are $(alpha, k) \in \{0.05, 0.01\} \times \{2, 3\}$. For Random Forests the space of hyper-parameters was $(numTrees, minLeafSize, numVarToSample) \in \{1000\} \times \{1, 3, 5\} \times \{(0.5, 1, 1.5, 2) * sqrt(numVar)\}$, where $numVar$ is the number of variables of the dataset. We tested SVMs with linear, polynomial and RBF kernels. For their hyper-parameters we examined, wherever applicable, all the combinations of $degree \in \{2, 3\}$, $gamma \in \{0.01, 0.1, 1, 10, 100\}$ and $cost \in \{0.01, 0.1, 1, 10, 100\}$. Finally, LASSO was tested with all the combinations of $alpha = \{0.001, 0.5, 1.0\}$ and 10 values for $lambda$ which are created independently for each dataset using the glmnet library [47]. Overall, the number of configurations in C for each dataset is equal to 610.

For the assessment of the performance of the models inferred from the configurations in C, we used the Area Under the Receiver's Operating Characteristic Curve (AUC) [48] which is independent of the prior class distribution. The ROC curve is the curve that illustrates the performance for a binary classification problem, when a threshold is varied on the predictions. It is the curve of sensitivity, also known as recall or true positive rate, plotted against 1-specificity which is also known as false positive rate.

6.2.3 Bias and variance estimation

We performed model selection and evaluation using KCV, NCV, TT, BBC, and BED for each of the 300 created sub-datasets. For the KCV protocol we used K = 10 and we applied the same

split of the folds for all the other methods. It is important to note again, that when the BBC method is used we first pool the out-of-sample predictions of KCV and then the performance of each model is computed as the average performance of all the out-of-sample predictions. Usually, the performance for each model is computed as the average performance for each fold and then over all folds. In the case of some metrics, such as classification accuracy, the model that will be selected in each case will be the same. However, in the case of using AUC as a metric of performance , it is possible that the two methods (pooling and averaging over all folds) result in different orderings of the models and consequently, the models that the two methods select could be different. For that reason, we have two versions of KCV, the *KCV-pooling* which selects the same model as BBC, and the *KCV-average over folds* which selects the same model as NCV and TT. BED uses the pooling of the predictions approach. For NCV, K was set to 9 for the inner KCV loop, and 10 for the outer.

The bias of the estimations is computed as $\hat{Bias} = \hat{P} - P_{true}$, where $\hat{P}$ and P_{true} denote the estimated and the true performance of the selected model, respectively. The target value of $\hat{Bias}$ is 0, indicating the absence of bias. A positive bias indicates a lower P_{true} than the one estimated by the corresponding performance estimation method and implies the method is optimistic. For each protocol, original dataset, and sample size, we compute the average bias and its standard deviation over the 10 sub-samplings.

6.2.4 Results and Discussion

Bootstrap Bias Correction

Figures 6.8 and 6.9, present the average performance bias and the standard deviation of the performance bias, respectively, for KCV-pooling, BBC, KCV-average over folds, NCV and TT. KCV-pooling and KCV-average over folds, overestimate performance especially for small sample sizes ($\leqslant 100$). BBC and NCV, both correct the bias of the corresponding KCV variation in a mostly conservative way. TT is optimistic for sample size equal to 20 and over-corrects the bias for sample size $\geqslant 60$ for most of the datasets compared to BBC and NCV. The two variants of KCV have the smallest stds. BBC, NCV and TT have similar stds, although results vary with dataset.

Figure 6.10 presents the average performance of the different protocols as well as the true performance of the models they select (performance evaluated on the holdout set). The protocols of each column select the same model. Figure 6.11 shows the corresponding std of performance for all methods.

The second column of Table 6.2 shows the percentage of times, over all sub-datasets, for all sample sizes that KCV-pooling and BBC select the same model as KCV-average over folds, NCV and TT. Although the two variants of KCV mainly select different models, the true performances of the models they select (see Figure 6.10) show minor difference in points of AUC and also similar stds of performance (see Figure 6.11).

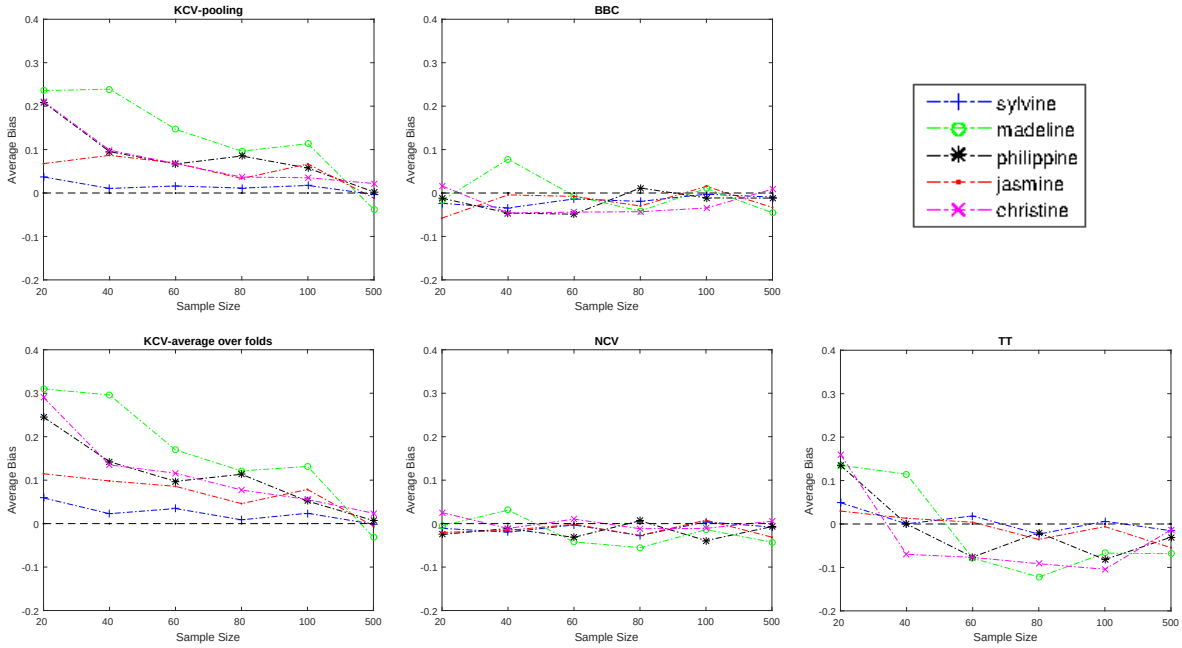

Figure 6.8: Average performance bias for the estimates of KCV-pooling, KCV-average over folds, BBC, NCV and TT. KCV-pooling exhibits lower bias than KCV-average over folds. BBC and NCV, both correct the bias of the corresponding version of KCV in a conservative way, although results vary with dataset. TT over-corrects compared to BBC and NCV and its bias is higher for sample sizes equal to 40.

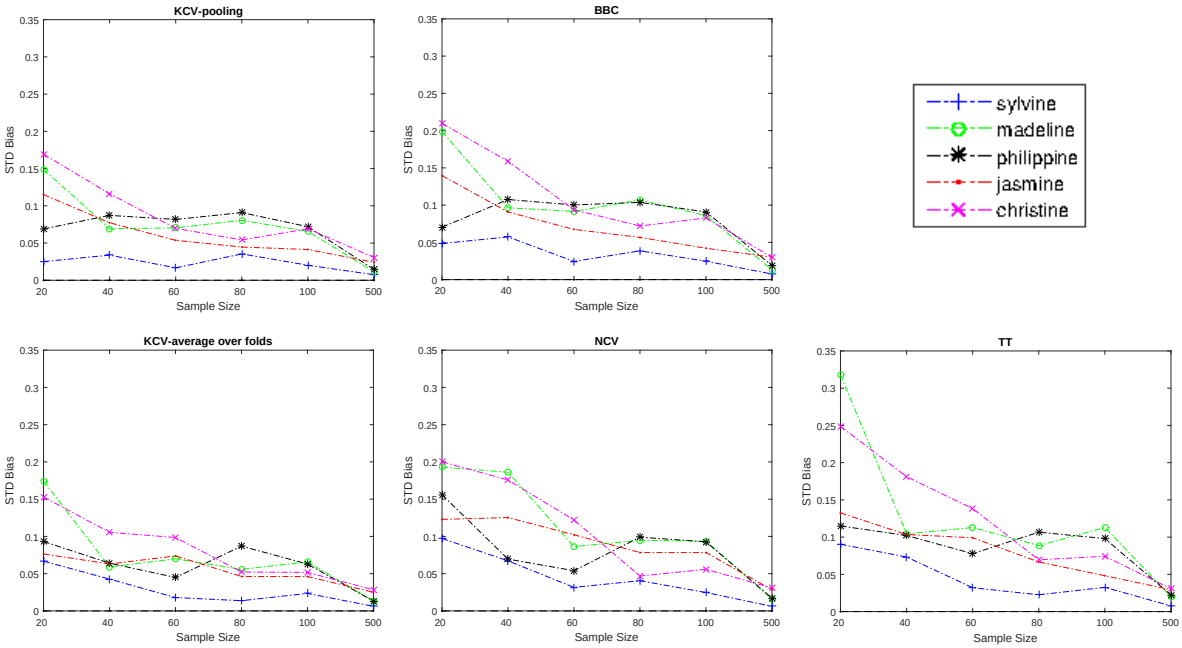

Figure 6.9: Standard deviation of bias for the estimates of KCV-pooling, KCV-average over folds, BBC, NCV and TT. KCV has the smallest variance but it overestimates performance. BBC, NCV and TT exhibit similar stds, although results vary with dataset.

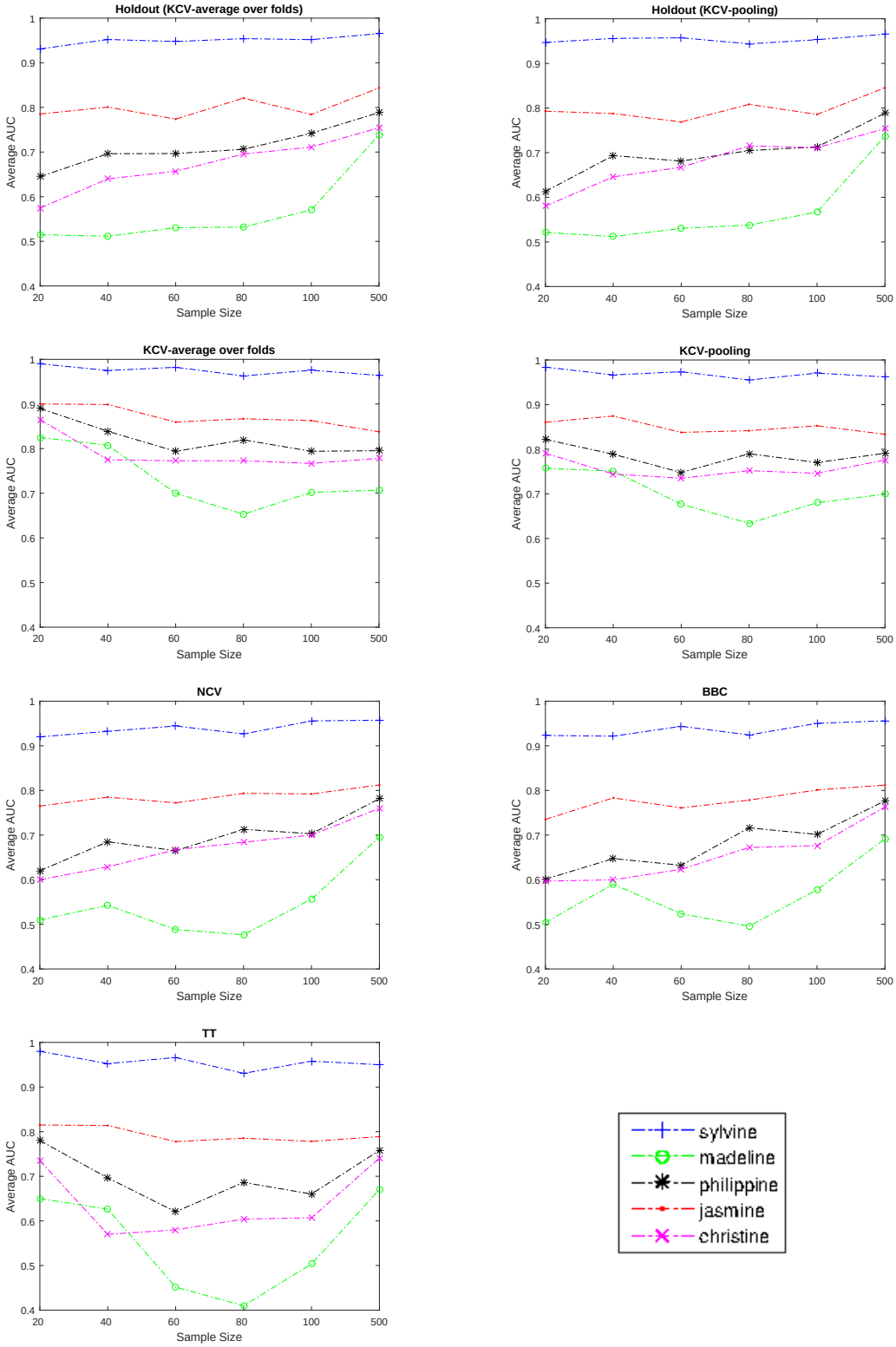

Figure 6.10: Average performance for the estimates of KCV-pooling, KCV-average over folds, BBC, NCV and TT and the true performance of the models that they select (Holdout). The methods of each column select the same model. KCV-pooling and KCV-average over folds over-estimate the performance. BBC and NCV are slightly conservative and TT mainly over-estimates for small sample sizes and over-corrects for larger ones.

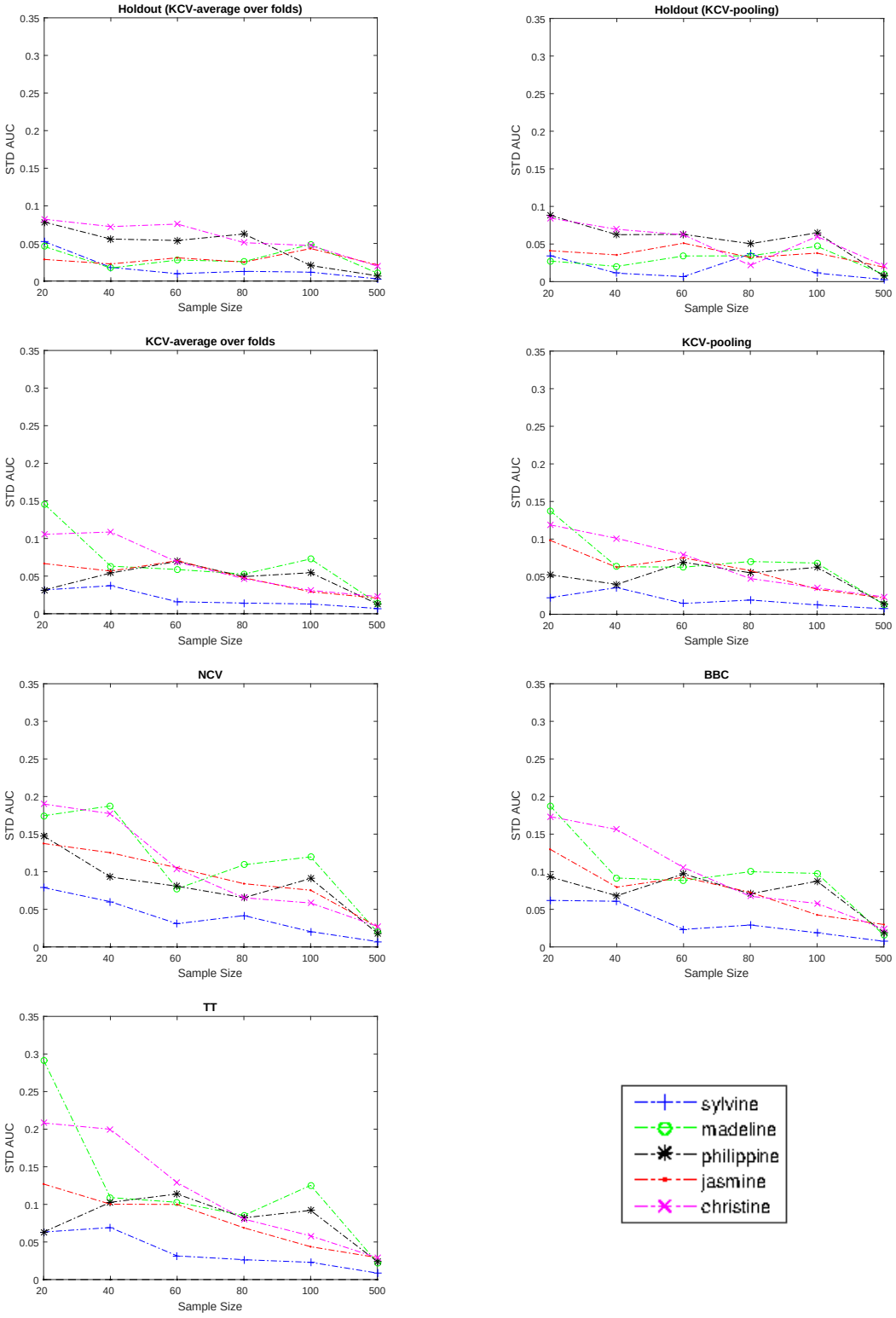

Figure 6.11: Standard deviation of performance for the estimates of KCV-pooling, KCV-average over folds, BBC, NCV and TT and the true performance of the models that they select (Holdout). The methods of each column select the same model. Holdout and KCV have the lower stds. BBC, NCV and TT have similar stds.

We could conclude that the two variants of KCV exhibit similar results and that BBC and NCV are the most promising methods for performance estimation. However, BBC is a lot less computationally expensive than NCV with the latter having to train about K times the number of models that BBC does. If computational time is an issue, BBC is clearly the optimal choice. TT also has low computational overhead with respect to KCV, however it appears to be unsuitable for small sample sizes and over-corrects for larger sample sizes.

Dropping of Under-Performing Configurations

Figures 6.12 and 6.13 show the bias and std of performance, respectively, of BED and BED-BBC for $B = 1000$ and $t \in \{0.90, 0.95, 0.99\}$. Both methods exhibit lower bias than KCV. BED-BBC corrects the bias of BED by only a small amount. It is mainly conservative with the exception of the *madeline* and *christine* datasets for small sample sizes ($\leqslant 40$). The aforementioned datasets have similar behaviour with BBC and NCV (positive bias). However, BED and BED-BBC have mainly higher stds than KCV, BBC and NCV.

In terms of the true performance of the models that are selected, BED shows insignificant loses compared to KCV. Figure 6.16 shows the boxplots of the relative true performance of the final models selected by the BED and KCV methods (on the left column), and the relative average number of trained models for the two methods (on the right column), for the *sylvine*, *madeline*, *philippine*, *jasmine*, and *christine* datasets and all sample sizes ($\{20, 40, 60, 80, 100, 500\}$). Indeed, we see that the maximum loss of performance is 6%. For the *madeline*, *philippine*, and *jasmine* datasets we also notice an increase in performance. However, for the *jasmine* and *christine* datasets there exist outliers that have greater loss in performance. This raises the need for the experiments to be replicated on a larger number of sub-datasets ($\geqslant 50$). The number of models that are being trained during BED varies with the value of the threshold t. The lower t is, the lower the number of trained models. For $t = 0.90$, at most 22% of the 6100 models that are trained throughout KCV, are actually trained. For $t = 0.99$, this percentage is at most 51% (shown also in Table 6.2).

It is clear that there is a huge gain in computational time, even with the strictest value for the threshold t. The loss in performance is negligible especially for larger values of sample size which is when the dropping method will be the most useful.

Columns 5-7 of Table 6.2 show in detail the relative average number of trained models for the BED and KCV methods for all datasets, sample sizes and values of threshold t.

BED and BED-BBC seem to be suitable for the computationally costly larger values of sample size, since there is no or insignificant loss in performance and the number of models that are trained are greatly reduced, resulting in a much faster method than KCV.

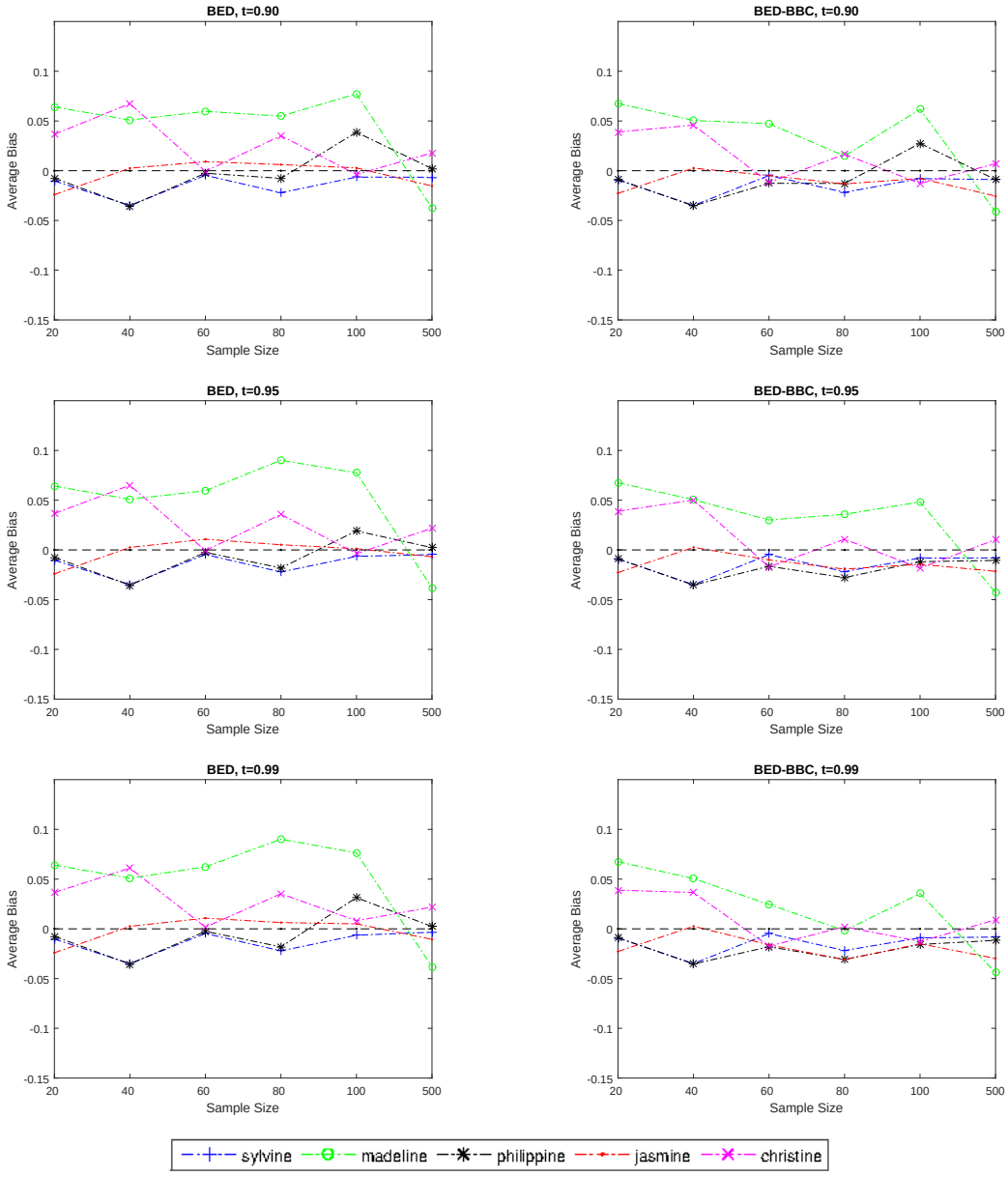

Figure 6.12: Average performance bias for the estimates of BED and BED-BBC for $B = 1000$ and different values of the threshold t. They all exhibit similar results, with BED-BBC with $t = 0.99$ having the lowest bias. BBC has a minor effect on the correction of the bias of BED for the datasets.

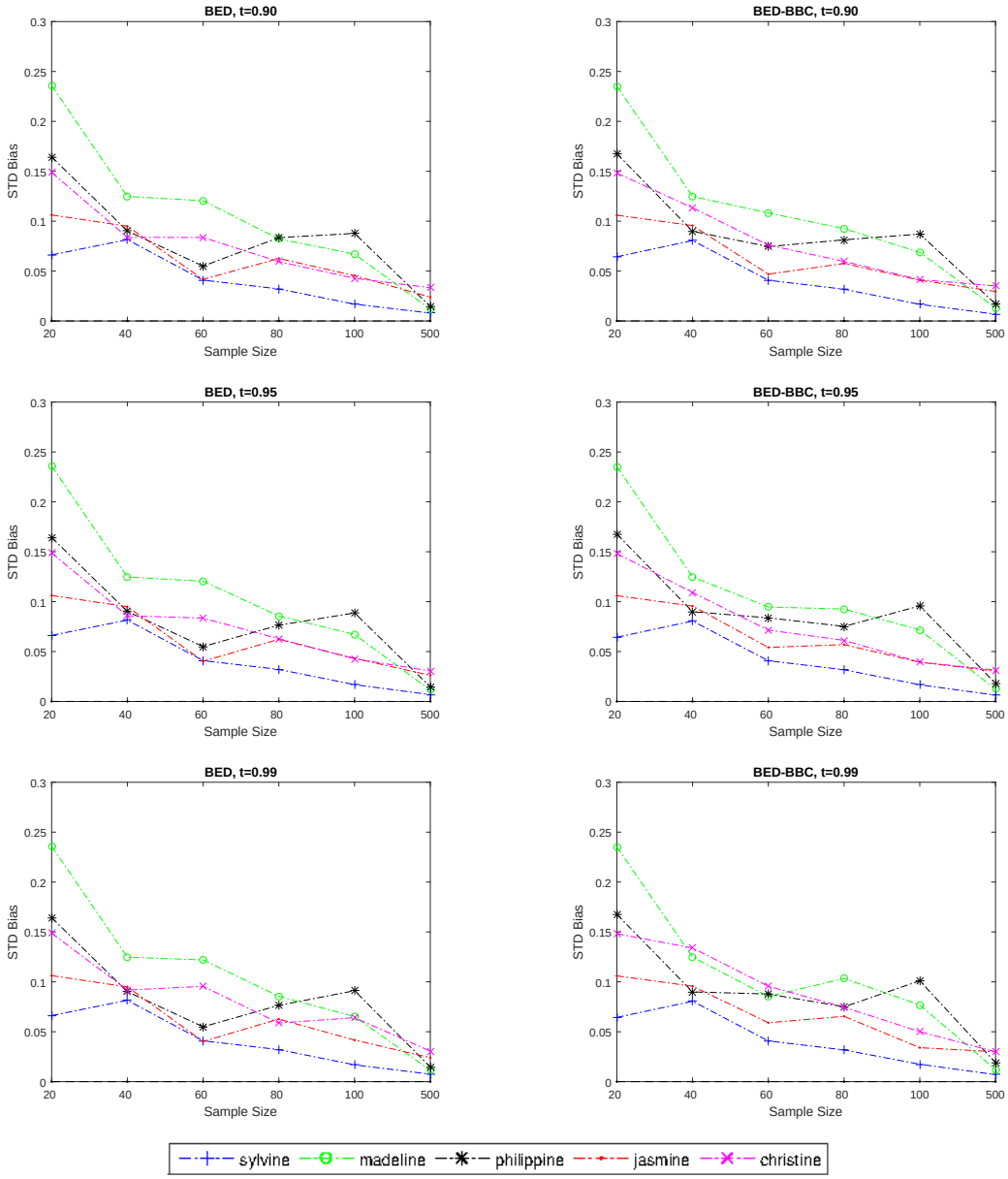

Figure 6.13: Standard deviation of bias for the estimates of BED and BED-BBC for $B = 1000$ and different values of the threshold t. They all exhibit similar results.

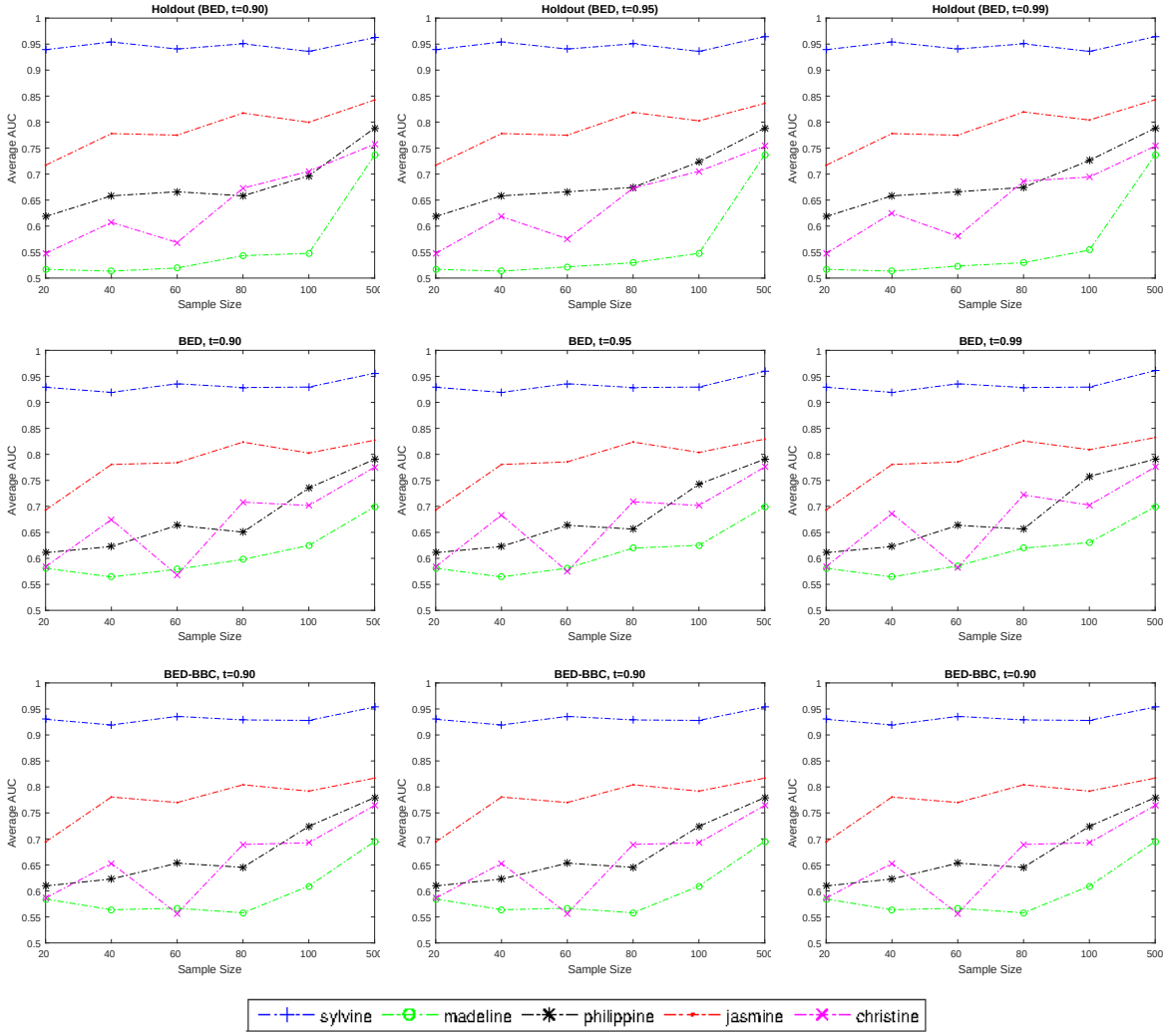

Figure 6.14: Average performance for the estimates of BED and BED-BBC for different thresholds t and the true performance of the models that they select (Holdout). The methods of each column select the same model. BED and BED-BBC are slightly conservative.

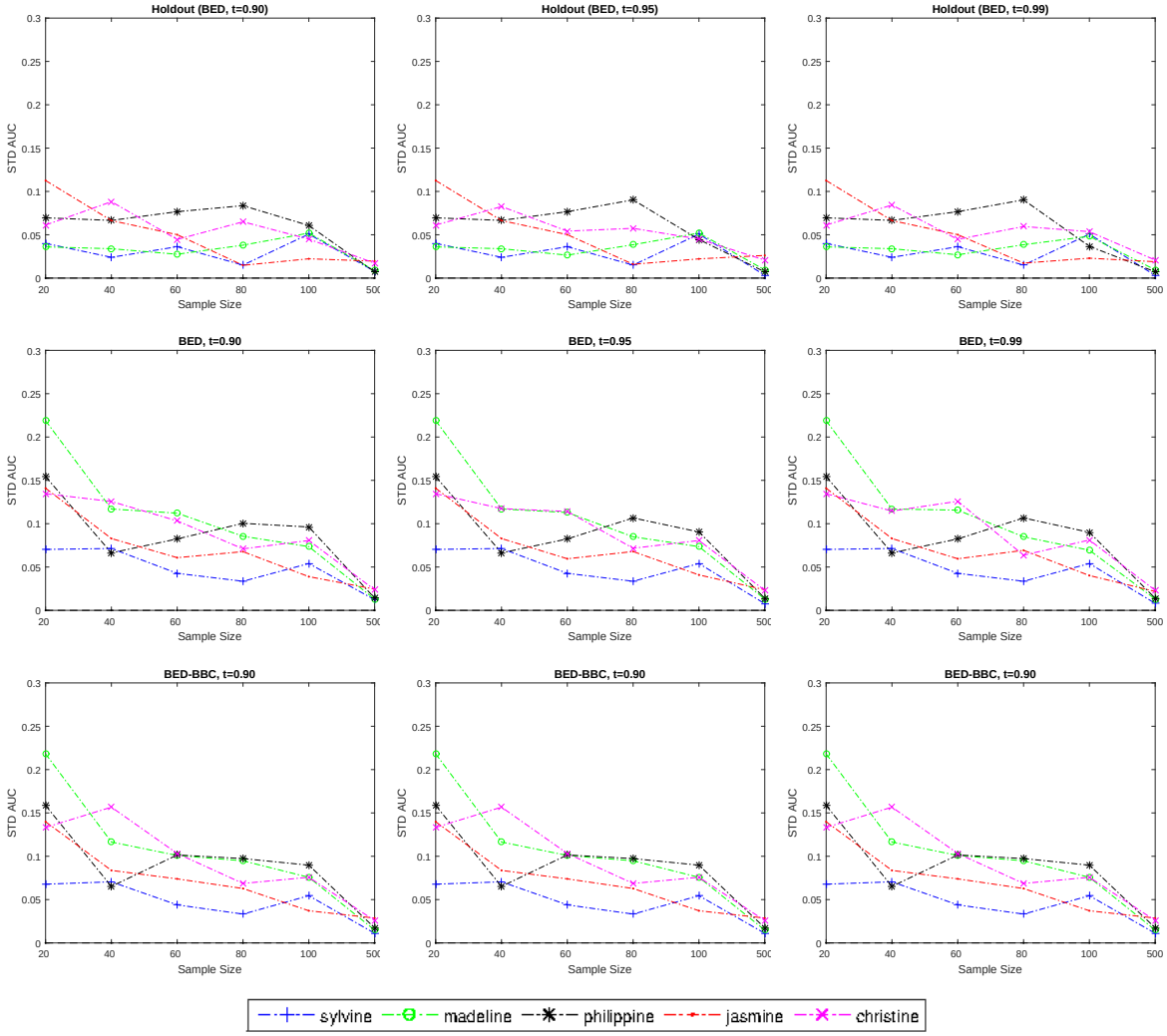

Figure 6.15: Standard deviation of performance for the estimates of BED and BED-BBC for different thresholds t and the true performance of the models that they select (Holdout). The methods of each column select the same model. They all have similar stds.

Table 6.2: Under Model selection: the percentage of the times, over the 10 sub-datasets, that the model selected by KCV-pooling and BBC is the same as the one selected by NCV (KCV-average over folds and TT), and BED for different thresholds t. Under Number of trained models: the number of models trained by BED relatively to KCV.

	Model selection				Number of trained models		
	NCV	BED			BED		
		$t = 0.90$	$t = 0.95$	$t = 0.99$	$t = 0.90$	$t = 0.95$	$t = 0.99$
20							
sylvine	20%	20%	20%	20%	10%	10%	10%
madeline	20%	10%	10%	10%	10%	10%	10%
philippine	20%	0%	0%	0%	10%	10%	10%
jasmine	10%	0%	0%	0%	10%	10%	10%
christine	20%	0%	0%	0%	10%	10%	10%
40							
sylvine	30%	0%	0%	0%	10%	10%	10%
madeline	20%	0%	0%	0%	10%	10%	10%
philippine	30%	0%	0%	0%	10%	10%	10%
jasmine	20%	0%	0%	0%	10%	10%	10%
christine	30%	20%	20%	30%	16%	18%	20%
60							
sylvine	0%	0%	0%	0%	10%	10%	10%
madeline	30%	20%	20%	30%	12%	17%	29%
philippine	0%	0%	0%	0%	11%	13%	12%
jasmine	30%	20%	30%	30%	15%	21%	31%
christine	20%	0%	0%	20%	11%	15%	20%
80							
sylvine	30%	0%	0%	0%	10%	10%	10%
madeline	40%	20%	80%	80%	15%	23%	51%
philippine	30%	10%	30%	30%	11%	12%	14%
jasmine	10%	60%	60%	70%	18%	26%	43%
christine	10%	30%	40%	50%	17%	21%	29%
100							
sylvine	40%	20%	20%	20%	13%	13%	15%
madeline	20%	10%	10%	20%	14%	16%	24%
philippine	40%	50%	60%	90%	18%	23%	39%
jasmine	20%	20%	30%	40%	16%	20%	32%
christine	30%	50%	50%	60%	13%	15%	25%
500							
sylvine	40%	40%	70%	80%	19%	25%	35%
madeline	90%	90%	100%	100%	13%	15%	23%
philippine	40%	90%	100%	100%	20%	26%	36%
jasmine	30%	40%	60%	90%	19%	29%	51%
christine	60%	90%	100%	100%	22%	29%	40%

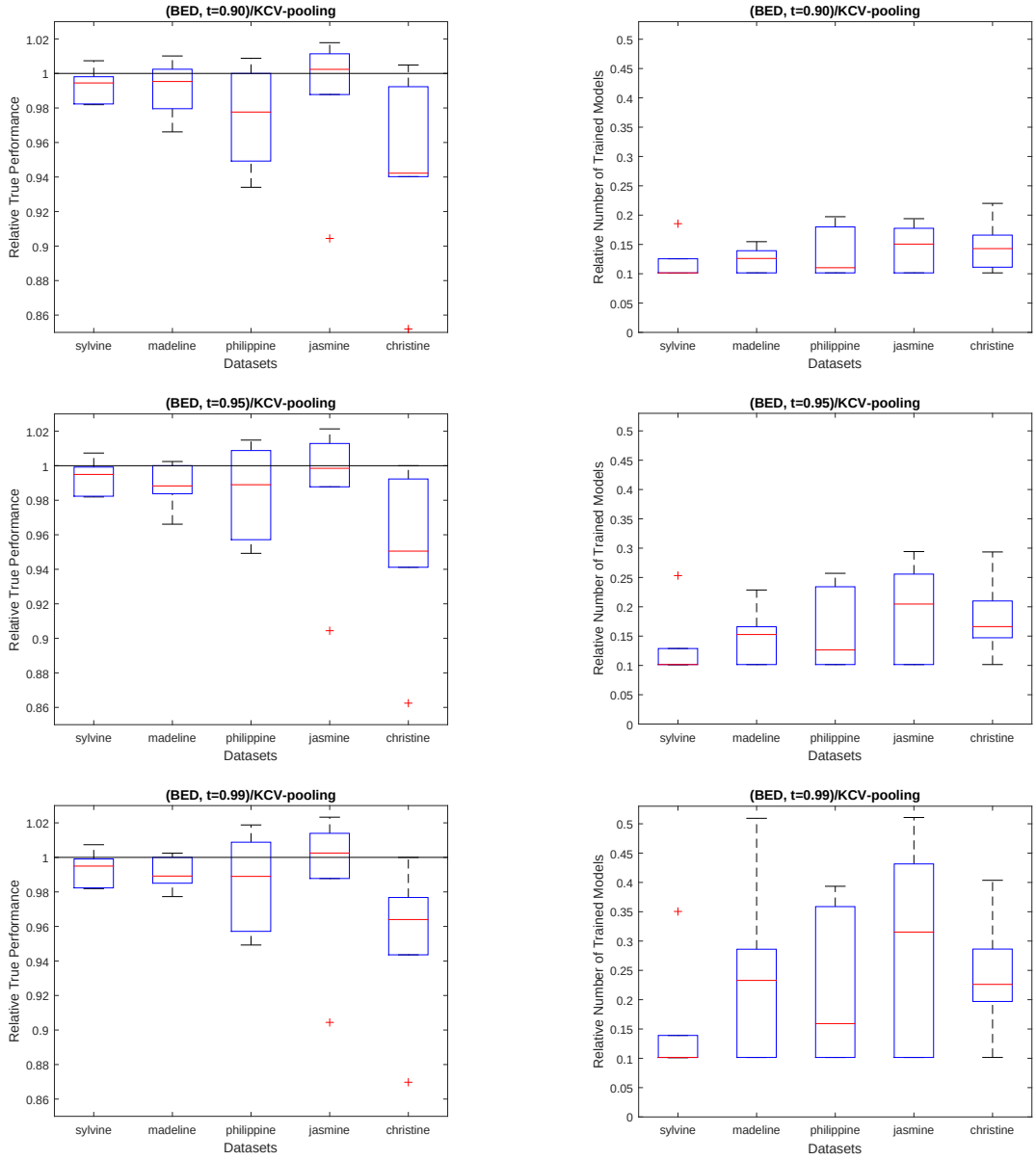

Figure 6.16: Boxplots of the relative true performance (left column), and the relative number of trained models (right column) for the *sylvine*, *madeline*, *philippine*, *jasmine*, and *christine* datasets for all sample sizes ($\{20, 40, 60, 80, 100, 500\}$) for the BED and KCV methods. There is a negligible to no effect on performance when using the BED method, However, the number of models that are trained is greatly reduced.

Chapter 7

Conclusion

7.1 Contribution

We have presented a bootstrap-based bias correction (BBC) method for correcting the bias of the Cross-Validation estimate of performance. It is a general method which works regardless of the data analysis task (e.g. classification, regression) or the structure of the models being involved in it. It has low computational overhead with respect to the Cross-Validation procedure and produces almost unbiased expected performance estimates even when the number of training samples is small. Compared to other methods for performance assessment, BBC was proven to overcome their limitations, thus making it the most appealing method for use.

We also presented a method for eliminating under-performing configurations within the Cross-Validation procedure (BED) in order to speed it up with a negligible to no effect on performance. The method also uses the bootstrap as a hypothesis test for the hypothesis that a configuration exhibits equal performance as the currently best one. The test is employed in every new iteration of Cross-Validation. When the hypothesis can be rejected based on the predictions in only a few available folds, the configuration is dropped (eliminated) from further consideration and no additional models are trained on remaining folds.

The combination of the the two methods yield the BED-BBC procedure which produces reliable estimates of performance with low computational cost.

7.2 Future Work

The BBC and BED methods need to be thoroughly examined for other data analysis tasks (e.g. multiclass classification, regression) and for different number of configurations. The main drawback of the experimental set-up is the low number (10) of sub-datasets that we create for each dataset and sample size. Although the results consort with the simulations and results from other publications, we still need to repeat the experiments with a greater number of sub-datasets ($\geqslant 50$).

We, also, plan to evaluate the use of bootstrap for constructing confidence intervals using the percentile method and the more sophisticated BC_α method. In addition, we will be investigating the effects of repeats on the BBC and BED-BBC methods in terms of performance estimation bias and model selection properties (i.e. test BBC and BED with Repeated Cross-Validation).

Finally, we need to empirically compare BED to the CVST method [20] and also explore different approaches for dropping under-performing configurations within Cross Validation.

Bibliography

[1] V. Vapnik, *The Nature of Statistical Learning Theory.* Springer-Verlag New York, Inc., 1995.

[2] R. Tibshirani and R. Tibshirani, "A bias correction for the minimum error rate in cross-validation," *The Annals of Applied Statistics*, vol. 3, pp. 822–829, 2009.

[3] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer, 2009.

[4] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection." Morgan Kaufmann, 1995, pp. 1137–1143.

[5] Y. Bengio and Y. Grandvalet, "No unbiased estimator of the variance of k-fold cross-validation," *The Journal of Machine Learning Research*, vol. 5, pp. 1089–1105, 2004.

[6] S. Varma and R. Simon, "Bias in error estimation when using cross-validation for model selection," *BMC bioinformatics*, vol. 7, 2006.

[7] R. B. Rao and G. Fung, "On the dangers of cross-validation. an experimental evaluation," *International Conference on Data Mining*, pp. 588–596, 2008.

[8] G. Cawley and N. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *The Journal of Machine Learning Research*, vol. 11, pp. 2079–2107, 2010.

[9] C. Bernau, T. Augustin, and A. Boulesteix, "Correcting the optimal resampling-based error rate by estimating the error rate of wrapper algorithms," *Biometrics*, vol. 69, no. 3, pp. 693–702, 2013.

[10] I. Tsamardinos, A. Rakhshani, and V. Lagani, "Performance-estimation properties of cross-validation-based protocols with simultaneous hyper-parameter optimization," in *Artificial Intelligence: Methods and Applications.* Springer, 2014, pp. 1–14.

[11] Y. Ding, S. Tang, S. Liao, J. Jia, S. Oesterreich, Y. Lin, and G. Tseng, "Bias correction for selecting the minimal-error classifier from many machine learning models," *Bioinformatics*, vol. 30, no. 22, pp. 3152–3158, 2014.

[12] D. Jensen and P. Cohen, "Multiple comparisons in induction algorithms," *Machine Learning*, vol. 38, pp. 309–338, 2000.

[13] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees.* Taylor & Francis, 1984.

[14] B. Efron and R. Tibshirani, *An introduction to the bootstrap.* Chapman & Hall, 1993.

[15] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 36, no. 2, pp. 111–147, 1974.

[16] D. Allen, "The relationship between variable selection and data agumentation and a method for prediction," *Technometrics*, vol. 16, no. 1, pp. 125–127, 1974.

[17] S. Geisser, "The predictive sample reuse method with applications," *Journal of the American Statistical Association*, vol. 70, no. 350, pp. 320–328, 1975.

[18] C. Bernau, T. Augustin, and A.-L. Boulesteix, "Correcting the optimally selected resampling-based error rate: A smooth analytical alternative to nested cross-validation," 2011.

[19] D. Krstajic, L. Buturovic, D. Leahy, and S. Thomas, "Cross-validation pitfalls when selecting and assessing regression and classification models," *Journal of Cheminformatics*, vol. 6, no. 1, 2014.

[20] T. Krueger, D. Panknin, and M. Braun, "Fast cross-validation via sequential testing," *Journal of Machine Learning Research*, vol. 16, pp. 1103–1155, 2015.

[21] B. Efron, "Estimating the error rate of a prediction rule: Improvement on cross-validation," *Journal of the American Statistical Association*, vol. 78, no. 382, pp. 316–331, 1983.

[22] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statistics Surveys*, vol. 4, pp. 40–79, 2010.

[23] I. Guyon, "A scaling law for the validation-set training-set size ratio," *AT & T Bell Laboratories*, pp. 1–11, 1997.

[24] R. Graham, D. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. Addison-Wesley Longman Publishing Co., Inc., 1994.

[25] P. Geurts, "Contributions to decision tree induction: bias/variance tradeoff and time series classification," Ph.D. dissertation, University of Liège, Belgium, 2002.

[26] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[27] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *25th Annual Conference on Neural Information Processing Systems (NIPS 2011)*, vol. 24. Neural Information Processing Systems Foundation, 2011.

[28] F. Hutter, H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization*. Springer Berlin Heidelberg, 2011, pp. 507–523.

[29] C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown, "Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms," *CoRR*, 2012.

[30] J. Snoek, H. Larochelle, and R. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, pp. 2951–2959, 2012.

[31] N. Iizuka, M. Oka, H. Yamada-Okabe, M. Nishida, Y. Maeda, N. Mori, T. Takao, T. Tamesa, A. Tangoku, H. Tabuchi, K. Hamada, H. Nakayama, H. Ishitsuka, T. Miyamoto, A. Hirabayashi, S. Uchimura, and Y. Hamamoto, "Oligonucleotide microarray for prediction of early intrahepatic recurrence of hepatocellular carcinoma after curative resection," *The Lancet*, vol. 361, pp. 923–929, 2003.

[32] A. Statnikov, C. Aliferis, I. Tsamardinos, D. Hardin, and S. Levy, "A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis," *Bioinformatics*, vol. 21, no. 5, pp. 631–643, 2005.

[33] A. Statnikov, I. Tsamardinos, C. Aliferis, and Y. Dosbayev, "Gems: A system for automated cancer diagnosis and biomarker discovery from microarray gene expression data," *International Journal of Medical Informatics*, vol. 74, no. 5, pp. 491–503, 2005.

[34] B. Efron, "Bootstrap methods: Another look at the jackknife," *The Annals of Statistics*, vol. 7, no. 1, pp. 1–26, 1978.

[35] M. Quenouille, "Problems in plane sampling," *The Annals of Mathematical Statistics*, vol. 20, no. 3, pp. 355–375, 1949.

[36] ——, "Notes on bias in estimation," *Biometrika*, vol. 43, no. 3/4, pp. 353–360, 1956.

[37] J. Tukey, "Bias and confidence in not-quite large samples," *The Annals of Mathematical Statistics*, vol. 29, no. 2, pp. 614–623, 1958.

[38] P. Hall, "Theoretical comparison of bootstrap confidence intervals," *The Annals of Statistics*, vol. 16, no. 3, pp. 927–953, 1988.

[39] B. Efron, "Better bootstrap confidence intervals," *Journal of the American statistical Association*, vol. 82, no. 397, pp. 171–185, 1987.

[40] J. C. Nankervis, "Computational algorithms for double bootstrap confidence intervals," *Computational statistics & data analysis*, vol. 49, no. 2, pp. 461–475, 2005.

[41] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, no. 2, pp. 153–157, 1947.

[42] C.-C. Chang and C.-J. Lin, "Training nu-support vector regression: theory and algorithms," *Neural Computation*, vol. 14, no. 8, pp. 1959–1978, 2002.

[43] 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). Heraklion*, 2012.

[44] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[45] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011.

[46] I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, T. K. Ho, N. Macià, B. Ray, M. Saeed, A. Statnikov, and E. Viegas, "Design of the 2015 chalearn automl challenge," in *Proc. of IJCNN*, 2015.

[47] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of Statistical Software*, vol. 33, no. 1, pp. 1–22, 2010.

[48] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.