Mach Learn (2018) 107:1895–1922
https://doi.org/10.1007/s10994-018-5714-4

Bootstrapping the out-of-sample predictions for efficient and accurate cross-validation

Ioannis Tsamardinos[1] · **Elissavet Greasidou**[1] · **Giorgos Borboudakis**[1]

Received: 3 August 2017 / Accepted: 21 April 2018 / Published online: 9 May 2018
© The Author(s) 2018

Abstract Cross-Validation (CV), and out-of-sample performance-estimation protocols in general, are often employed both for (a) selecting the optimal combination of algorithms and values of hyper-parameters (called a configuration) for producing the final predictive model, and (b) estimating the predictive performance of the final model. However, the cross-validated performance of the best configuration is optimistically biased. We present an efficient bootstrap method that corrects for the bias, called Bootstrap Bias Corrected CV (BBC-CV). BBC-CV's main idea is to bootstrap the whole process of selecting the best-performing configuration on the out-of-sample predictions of each configuration, without additional training of models. In comparison to the alternatives, namely the nested cross-validation (Varma and Simon in BMC Bioinform 7(1):91, 2006) and a method by Tibshirani and Tibshirani (Ann Appl Stat 822–829, 2009), BBC-CV is computationally more efficient, has smaller variance and bias, and is applicable to any metric of performance (accuracy, AUC, concordance index, mean squared error). Subsequently, we employ again the idea of bootstrapping the out-of-sample predictions to speed up the CV process. Specifically, using a bootstrap-based statistical criterion we stop training of models on new folds of inferior (with high probability) configurations. We name the method Bootstrap Bias Corrected with Dropping CV (BBCD-CV) that is both efficient and provides accurate performance estimates.

Keywords Performance estimation · Bias correction · Cross-validation · Hyper-parameter optimization

Ioannis Tsamardinos and Elissavet Greasidou have equally contributed to this work.

Editor: Hendrik Blockeel.

✉ Elissavet Greasidou
greasidouelissavet@gmail.com

Ioannis Tsamardinos
tsamard.it@gmail.com

[1] Computer Science Department, University of Crete and Gnosis Data Analysis PC, Heraklion, Greece

1 Introduction

Typically, the goals of a machine learning predictive modeling task are twofold: to return a high-performing predictive model for operational use and an estimate of its performance. The process often involves the following steps: (a) *Tuning*, where different combinations of algorithms and their hyper-parameter values (called *configurations*) are tried producing several models, their performance is estimated, and the best configuration is determined, (b) *Production* of the final model trained on all available data using the best configuration, and (c) *Performance Estimation* of the final model.

Focusing first on tuning, we note that a configuration may involve combining several algorithms for every step of the learning process such as: pre-processing, transformation of variables, imputation of missing values, feature selection, and modeling. Except for rare cases, each of these algorithms accepts a number of hyper-parameters that tune its behavior. Usually, these hyper-parameters affect the sensitivity of the algorithms to detecting patterns, the bias-variance trade-off, the trade-off between model complexity and fitting of the data, or may trade-off computational complexity for optimality of fitting. Examples include the maximum number of features to select in a feature selection algorithm, or the type of kernel to use in Support Vector Machine and Gaussian Process learning.

There exist several strategies guiding the order in which the different configurations are tried, from sophisticated ones such as Sequential Bayesian Optimization (Snoek et al. 2012; Garnett et al. 2010) to simple grid search in the space of hyper-parameter values. However, independently of the order of production of configurations, the analyst needs to estimate the performance of the average model produced by each configuration on the given task and select the best.

The estimation methods of choice for most analysts are the out-of-sample estimation protocols, where a portion of the data training instances is hidden from the training algorithm to serve as an independent test set. The performance of several models stemming from different configurations is tried on the test set, also called the hold-out set, in order to select the best performing one. This procedure is known as the *Hold-out* protocol. We will refer to such a test set as a *tuning set* to emphasize the fact that it is employed repeatedly by all configurations for the purposes of tuning the algorithms and the hyper-parameter values of the learning pipeline. We note that while there exist approaches that do not employ out-of-sample estimation, such as using the Akaike Information Criterion (AIC) (Akaike 1974) of the models, the Bayesian Information Criterion (BIC) (Schwarz 1978), and others, in this paper we focus only on out-of-sample estimation protocols.

The process of withholding a tuning set can be repeated multiple times leading to several analysis protocols variations. The simplest one is to repeatedly withhold different, randomly-chosen tuning sets and select the one with the best average performance over all tuning sets. This protocol is called the *Repeated Hold-out*.

Arguably however, the most common protocol for performance estimation for relatively low sample sizes is the *K-fold Cross-Validation* or simply Cross-Validation (CV). In CV the data training instances are partitioned to K approximately equal-sized subsets, each one serving as a tuning set and the remaining ones as training sets. The performance of each configuration is averaged over all tuning folds. The difference with the Repeated Hold-Out is that the process is repeated exactly K times and the tuning sets are enforced to be non-overlapping in samples (also referred to as instances, examples, or records). The process can be repeated with different partitions of the data to folds leading to the *Repeated CV*.

A final note on tuning regards its name. In statistics, the term *model selection* is preferred for similar purposes. The reason is that a common practice in statistical analysis is to produce several models using different configurations on all of the available data, manually examine their fitting, degrees of freedom, residuals, AIC and other metrics and then make an informed (but manual) choice of a model. In contrast, in our experience machine learning analysts estimate the performance of each configuration and select the best *configuration* to employ on all data, rather than selecting the best model. Thus, in our opinion, the term tuning is more appropriate than model selection for the latter approach.

Considering now the production of the final model to deploy operationally, the most reasonable choice is arguably to *train a single model using the best configuration found on all of the available data*. Note that each configuration may have produced several models during CV or Repeated Hold-Out for tuning purposes, each time using a subset of the data for training. However, assuming that—on average—a configuration produces better predictive models when trained with larger sample sizes (i.e., its learning curve is monotonically improving) it is reasonable to employ all data to train the final model and not waste any training examples (samples) for tuning or performance estimation. There may be exceptions of learning algorithms that do not abide to this assumption (see Krueger et al. 2015 for a discussion) but it is largely accepted and true for most predictive modeling algorithms.

The third step is to compute and return a performance estimate of the final model. *The cross-validated performance of the best configuration (estimated on the tuning sets) is an optimistically biased estimate of the performance of the final model. Thus, it should not be reported as the performance estimate.* Particularly for small sample sizes (less than a few hundred) like the ones that are typical in molecular biology and other life sciences, and when numerous configurations are tried, the optimism could be significant.

The main problem of using the estimation provided on the tuning sets is that these sets have been employed repeatedly by all configurations, out of which the analysts selected the best. Thus, equivalent statistical phenomena occur as in multiple hypothesis testing. The problem was named the multiple comparisons in induction problems and was first reported in the machine learning literature by Jensen and Cohen (2000). A simple mathematical proof of the bias 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, where $|D_{train}|$ is the size of the training sets. The sample estimate of μ_i on the tuning sets 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 $\min\{m_1, \ldots, m_n\}$, where n is the number of configurations tried. On average, the estimate on the best configuration on the tuning sets is $E(\min\{m_1, \ldots, m_n\})$ while the estimate of the true best is $\min\{\mu_1, \ldots, \mu_n\} = \min\{E(m_1), \ldots, E(m_n)\}$. The optimism (bias) is $Bias = \min\{E(m_1), \ldots, E(m_n)\} - E(\min\{m_1, \ldots, m_n\}) \geq 0$ by Jensen's inequality (1906). For metrics such as classification accuracy and Area Under the Receiver's Operating Characteristic Curve (AUC) (Fawcett 2006), where higher is better, the min is substituted with max and the inequality is reversed.

The bias of cross-validation when multiple configurations are tried has been explored empirically in Tsamardinos et al. (2015) on real datasets. For small samples (< 100) the AUC bias ranges frequently between 5 and 10%. The bias depends on several factors such as (a) the number of configurations tried, (b) the correlation between the performance of the models trained by each configuration, (c) the sample size, and (d) the difference between the performance of the true best configuration and the rest. Prior works (Varma and Simon 2006; Boulesteix and Strobl 2009; Yousefi et al. 2011) have also investigated and identified the bias of CV when tuning both on real and simulated data.

To avoid this bias *the simplest procedure is to hold-out a second, untainted set, exclusively used for estimation purposes on a single model*. This single model is of course the one produced with the best configuration as found on the tuning sets. This approach has been particularly advocated in the Artificial Neural Networks literature were the performance of the current network is estimated on a *validation* set (equivalent to a tuning set) as a stopping criterion of training (weight updates). Thus, the validation set is employed repeatedly on different networks (models), albeit only slightly different by the weight updates of one epoch. For the final performance estimation a separate, independent test set is used. Thus, in essence, the data are partitioned to train-tuning-estimation subsets: the tuning set is employed multiple times for tuning of the configuration; then, a single model produced on the union of the train and tuning data with the best configuration is tested on the estimation subset. Generalizing this protocol so that all folds serve as tuning and estimation subsets and performance is averaged on all subsets leads to the *Nested Cross-Validation* (NCV) protocol (Varma and Simon 2006). The problem with NCV is that it requires $O(K^2 \cdot C)$ models to be trained, where K the number of folds and C the number of configurations tried, resulting in large computational overheads.

The main contribution of this paper is the idea that *one can bootstrap the pooled predictions of all configurations over all tuning sets (out-of-sample predictions)* to achieve several goals. The first goal is to estimate the loss of the best configuration (i.e., remove the *bias* of cross-validating multiple configurations) without training additional models. Specifically, the (out-of-sample) predictions of all configurations are bootstrapped, i.e., selected with replacement, leading to a matrix of predictions. The configuration with the minimum loss on the bootstrapped data is selected and its loss is computed on the out-samples (not selected by the bootstrap). The procedure is repeated for a few hundred bootstraps and the average loss of the selected best configuration on the out-samples is returned. Essentially, the above procedure *bootstraps the strategy* for selecting the best configuration and computes its average loss on the samples not selected by the bootstrap.

Bootstrapping has a relatively low computational overhead and is trivially parallelized. The computational overhead for each bootstrap iteration amounts to re-sampling the sample indexes of predictions, computing the loss on the bootstrapped predictions for each configuration, and selecting the minimum. We call the method Bootstrap Bias Corrected CV (BBC-CV). BBC-CV is empirically compared against NCV, the standard for avoiding bias, and a method by Tibshirani and Tibshirani (2009) (TT from hereon) which addresses the large computational cost of NCV. BBC-CV is shown to exhibit a more accurate estimation of the *Bias* than TT and similar to that of NCV, while it requires no training of new models, and thus being as computationally efficient as TT and much faster than NCV. Bootstrapping the out-of-sample predictions can also trivially be used to compute confidence intervals for the performance estimate in addition to point estimates. In experiments on real data, we show that the confidence intervals are accurate albeit somewhat conservative (i.e. have higher coverage than expected).

The second main use of bootstrapping the out-of-sample predictions is to create a hypothesis test for the hypothesis that a configuration exhibits equal performance as the currently best configuration. The test is employed in every new fold serving for tuning during CV. When the hypothesis can be rejected based on the predictions on a limited number of folds, the configuration is eliminated or dropped from further consideration and no additional models are trained on the remaining folds. We combine the idea of dropping configurations with the BBC-CV method for bias correction, and get the Bootstrap Bias Corrected with Dropping CV (BBCD-CV). BBCD-CV results in significant computational gains, typically achieving a speed-up of 2-5 (in some cases up to the theoretical maximum which equals the number of

folds, in this case 10) over BBC-CV, while providing accurate estimates of performance and confidence intervals. Finally, we examine the role of repeating the procedure with different partitions to folds (*Repeated BBC-CV*) and show that multiple repeats improve the selection of the best configuration (tuning) and lead to better performing models. In addition, for the same number of trained models, Repeated BBC-CV leads to better performing models than NCV while having similar bias in their performance estimates. These results corroborate a growing body of work on the benefits of repeating an analysis with multiple fold partitions. Krstajic et al. (2014) and more recently, Seibold et al. (2017) have also shown that repeating the cross-validation procedure decreases the variability in the tuning process and produces a more robust result in terms of the performance of the final model. The latter work in particular, also examines the effect of repeating the CV on the selection of features.

The rest of this paper is structured as follows. In Sect. 2 we present and discuss widely established protocols for tuning and out-of-sample performance estimation. In Sect. 3.3 we discuss additional related work. We introduce our methods BBC-CV and BBCD-CV in Sects. 3 and 4, respectively, and empirically evaluate them on synthetic and real settings in Sect. 5. We conclude the paper in Sect. 6.

2 Preliminaries of out-of-sample estimation

In this section, we present the basics of out-of-sample estimation of the performance of a learning method f and introduce the notation employed in the rest of the paper. We assume the learning method is a function that accepts as input a dataset $D = \{\langle x_j, y_j \rangle\}_{j=1}^{N}$ of pairs of training vectors x and their corresponding labels y and returns another function $M(x)$ (a predictive model), so that $f(D) = M$. We can also think of D as a 2D matrix with the rows containing the examples, and the columns corresponding to features (a.k.a. variables, attributes, measured/observed quantities). It is convenient to employ the *Matlab* index notation on matrices to denote with $D(:, j)$ the j-th column of D and $D(i, :)$ the i-th row of D; similarly $D(I, j)$ is the vector of values in the j-th column from rows with indexes in vector I.

We also overload the notation and use $f(x, D)$ to denote the output (predictions) of the model M trained by f on dataset D when given as input one or multiple samples x. We denote the loss (metric of error) between the value y of a label and a corresponding prediction $\hat{y}$ as $l(y, \hat{y})$. For convenience, we can also define the loss between a *vector* of labels y and a *vector* of predictions $\hat{y}$ as the vector of losses between the corresponding labels and predictions:

$$[l(y, \hat{y})]_j = l(y_j, \hat{y_j})$$

The loss function can be either the 0-1 loss for classification (i.e, one when the label and prediction are equal and zero otherwise), the squared error $(y - \hat{y})^2$ for regression or any other metric. Some metrics of performance such as the AUC or the Concordance Index for survival analysis problems (Harrell et al. 1982) cannot be expressed using a loss function defined on single pairs $\langle y, \hat{y} \rangle$. These metrics can only be computed on a test set containing at least 2 predictions and thus, $l(y, \hat{y})$ is defined only when y and $\hat{y}$ are vectors for such metrics.

The K-fold Cross-Validation (CV) protocol is arguably the most common out-of-sample performance estimation protocol for relatively small sample sizes. It is shown in Algorithm 1. The protocol accepts a learning method f, a dataset D already partitioned into K folds F. *The model to return is computed by applying the learning method f on all available data*. To estimate the performance of this model CV employs each fold F_i in turn as an

Algorithm 1 CV$(f, D = \{F_1, \ldots, F_K\})$: Basic K-Fold Cross-Validation

Input: Learning method f, Data matrix $D = \{\langle x_j, y_j \rangle\}_{j=1}^{N}$ partitioned into about equally-sized folds F_i
Output: Model M, Performance estimation L_{CV}, out-of-sample predictions Π on all folds
1: Define $D_{\setminus i} \leftarrow D \setminus F_i$
2: // Obtain the indexes of each fold
3: $I_i \leftarrow indexes(F_i)$
4: // Final Model trained by f on all available data
5: $M \leftarrow f(D)$
6: // Performance estimation: learn from $D_{\setminus i}$, estimate on F_i
7: $L_{CV} \leftarrow \frac{1}{K} \sum_{i=1}^{K} l(y(I_i), f(F_i, D_{\setminus i}))$
8: // Out-of-sample predictions are used by bias-correction methods
9: Collect out-of-sample predictions $\Pi = [f(F_1, D_{\setminus 1}); \cdots ; f(F_K, D_{\setminus K})]$
10: **Return** $\langle M, L_{CV}, \Pi \rangle$

estimation set and trains a model M_i on the remaining data (in the algorithm denoted as $D_{\setminus i}$) using f, i.e., $M_i = f(D_{\setminus i})$. It then computes the loss of M_i on the hold-out fold F_i. The final performance estimate is the average loss over all folds. The pseudo-code in Algorithm 1 as presented, also collects and returns all out-of-sample predictions in a vector Π. This facilitates the presentation of some bias-correction methods below, who depend on them. In case no bias-correction is applied afterwards, Π can be omitted from the output arguments.

As simple and common as CV is, there are still several misconceptions about its use. First, the protocol returns $f(D)$ learned from the full dataset D, but the losses computed are on different models, namely models trained with subsets of the data. So, CV does not estimate the loss of the specific returned model. We argue that *cross-validation estimates the average loss of the models produced by f when trained on datasets of size $|D_{\setminus i}|$ on the distribution of D*. The key conceptual point is that *it is not the returned model who is being cross-validated, but the learning method f*. Non-expert analysts (and students in particular) often wonder which out of the K models produced by cross-validation excluding each fold in turn should be returned. The answer is none; the model to use operationally is the one learned by f on all of D.

A typical major assumption is that $f's$ models improve on any given task with increased sample size. This is a reasonable assumption to make, although not always necessarily true. If it does hold, then we expect that the returned loss estimate L of CV to be conservative, i.e., on average higher than the average loss of the returned model. This is because the final model f is trained on $|D|$ samples while the estimates are produced by models trained on fewer samples of size $|D_{\setminus i}|$. Exactly how conservative it will be depends on where the classifier is operating on its learning curve for this specific task, which is unknown a priori. It also depends on the number of folds K: the larger the K, the more $(K-1)/K$ approaches 100% and the bias disappears. When sample sizes are small or distributions are imbalanced (i.e., some classes are quite rare in the data), we expect most classifiers to quickly benefit from increased sample size, and thus, for CV to be more conservative.

Based on the above, one expects that leave-one-out CV (where each fold's size is 1 sample) should be the least biased. However, there is evidence that there exist situations where leave-one-out CV is not recommended. First, leave-one-out has been shown to exhibit a large estimation variance (Braga-Neto and Dougherty 2004). Second, the protocol can collapse in some situations in the sense that it can provide extremely misleading estimates in degenerate situations (see Witten et al. 2016, p. 151; Kohavi 1995 for an extreme failure of leave-one-out CV and of the 0.632 bootstrap rule). Specifically, consider the case where there are exactly 50 and 50 positive and negative examples in a dataset, respectively and the classifier employed

Algorithm 2 CVT(f, $D = \{F_1, \ldots, F_K\}$, Θ): Cross-Validation With Tuning

Input: Learning method f, Data matrix $D = \{\langle x_j, y_j \rangle\}_{j=1}^{N}$ partitioned into about equally-sized folds F_i, set of configurations Θ
Output: Model M, Performance estimation L_{CVT}, out-of-sample predictions Π on all folds for all configurations

1: **for** $i = 1$ to $C = |\Theta|$ **do**
2: // Create a closure of f (a new function) by grounding the configuration θ_i
3: $f_i \leftarrow \text{Closure}(f(\cdot, \theta_i))$
4: $\langle M_i, L_i, \Pi_i \rangle \leftarrow \mathbf{CV}(f_i, D)$
5: **end for**
6: $i^\star \leftarrow \arg\min_i L_i$
7: // Final Model trained by f on all available data using the best configuration
8: $M \leftarrow f(D, \theta_{i^\star})$
9: // Performance estimation; may be optimistic and should not be reported in general
10: $L_{CVT} \leftarrow L_{i^\star}$
11: // Out-of-sample predictions are used by bias-correction methods
12: Collect all out-of-sample predictions of all configurations in one matrix $\Pi \leftarrow [\Pi_1 \cdots \Pi_C]$
13: **Return** $\langle M, L_{CVT}, \Pi \rangle$

just takes a majority vote. In leave-one-out CV the training data always vote (predict) for the opposite class of the held-out instance and the performance estimate is 0%, instead of 50%. This problem stems from the fact that the folds may follow a quite different distribution than the distribution of the class in the original dataset: when only one example is left out, the distribution of one class in the fold is 100 and 0% for all the others. Instead, we advise to use K only as large as possible to still allow the distribution of classes in each fold to be approximately similar as in the original dataset, and impose this restriction when partitioning to folds. The latter restriction leads to what is called *stratified CV* and there is evidence that stratification leads to improved performance estimations (Tsamardinos et al. 2015).

2.1 Cross-validation with tuning (CVT)

A typical data analysis involves several algorithms to be combined, e.g., for transforming the data, imputing the missing values, variable selection or dimensionality reduction, and modeling. There are hundreds of choices of algorithms in the literature for each type of algorithms. In addition, each algorithm typically takes several hyper-parameter values that should be tuned by the user. We assume that the learning method $f(D)$ is augmented to $f(D, \theta)$ to take as input a vector θ that determines which combination of algorithms to run and with what values of hyper-parameters. We call θ a *configuration* and refer to the process of selecting the best θ as *tuning* of the learning pipeline.

The simplest tuning procedure is to cross-validate f with a different configuration θ each time within a predetermined set of configurations Θ, choose the best performing configuration $\theta^\star$ and then train a final model on all data with $\theta^\star$. The procedure is shown in Algorithm 2. In the pseudo-code, we compute f_i as the closure of f[1] when the configuration input parameter is grounded to the specific values in θ_i. For example, if configuration θ_i is a combination of a feature selection algorithm g and modeling algorithm h with their respective hyper-parameter values a and b, taking the closure and grounding the hyper-parameters produces a function $f_i = h(g(\cdot, a), b)$, i.e., a function f_i that first applies the specified feature selection g using

[1] The term closure is used in the programmatic sense to denote a function produced by another function by binding some free parameters to specific values; see also http://gafter.blogspot.gr/2007/01/definition-of-closures.html.

hyper-parameters a and uses the result to train a model using h with hyper-parameters b. The use of the closures leads to a compact pseudo-code implementation of the method.

We now put together two observations already noted above: the performance estimate L_{CVT} of the winning configuration tends to be conservative because it is computed by models trained on only a subset of the data; at the same time, it tends to be optimistic because it is selected as the best among many tries. Which of the two trends will dominate depends on the situation and is a priori unknown. For large K and a large number of configurations tried, the training sets are almost as large as the whole dataset and the optimistic trend dominates. In general, for small sample sizes and a large number of configurations tried L_{CVT} *is optimistic and should not be reported as the performance estimate of the final model*.

2.2 The nested cross-validation (NCV) protocol

Given the potential optimistic bias of CV when tuning takes place, other protocols have been developed, such as the Nested Cross-Validation (NCV). We could not trace who introduced or coined up first the name nested cross-validation but the authors and colleagues have independently discovered it and using it since 2005 (Statnikov et al. 2005); around the same time (Varma and Simon 2006), report a bias in error estimation when using K-fold cross-validation, and suggest the use of the NCV protocol as an almost unbiased estimate of the true performance. A similar method in a bioinformatics analysis was used in 2003 (Iizuka et al. 2003). One early comment hinting of the method is in Salzberg (1997), while Witten and Frank (2005, p. 286) briefly discuss the need of treating any parameter tuning step as part of the training process when assessing performance. It is interesting to note that the earlier works on NCV appeared first in bioinformatics where the sample size of datasets is often quite low and the effects of the bias due to tuning and trying multiple configurations are more dramatic.

The idea of the NCV is evolved as follows. Since the tuning sets have been used repeatedly for selecting the best configuration, one needs a second hold-out set exclusively for estimation of one, single, final model. However, one could repeat the process with several held-out folds and average the estimates. In other words, each fold is held-out for estimation purposes each time and a CVT takes place for the remaining folds in selecting the best configuration and training on all remaining data with this best configuration to return a single model. Thus, in NCV each fold serves once for estimation and multiple times as a tuning set. Under this perspective, NCV is a generalization of a double-hold-out protocol partitioning the data to train-tuning-estimation.

Another way to view NCV is to *consider tuning as part of the learning process*. The result is a new learning function f' that returns a single model, even though internally it is using CV to select the best configuration to apply to all input data. NCV simply cross-validates f'. What is this new function f' that uses CV for tuning and returns a single model? It is actually CVT for the given learning method f and configuration set Θ. Naturally, any method that performs hyper-parameter optimization and returns a single model can be used instead of CVT as f'. The pseudo-code in Algorithm 3 clearly depicts this fact and implements NCV in essentially two lines of code using the mechanism of closures.

Counting the number of models created by NCV, let us denote with $C = |\Theta|$ the number of configurations to try. To produce the final model, NCV will run CVT on all data. This will create $K \times C$ models for tuning and once the best configuration is picked, one more model will be produced leading to $K \times C + 1$ models for final model production. To produce the estimate, the whole process is cross-validated each time excluding one fold, thus leaving $K-1$ folds for the inner cross-validation loop (the loop inside f'). Overall, this leads to $K \times ((K-1) \times C + 1)$

Algorithm 3 NCV$(f, D = \{F_1, \ldots, F_K\}, \Theta)$: Nested Cross-Validation

Input: Learning method f, Data matrix $D = \{\langle x_j, y_j \rangle\}_{j=1}^{N}$ partitioned into about equally-sized folds F_i, set of configurations Θ
Output: Model M, Performance estimation L_{NCV}, out-of-sample predictions Π on all folds for all configurations
1: // Create closure by grounding the f and the Θ input parameters of **CVT**
2: $f' \leftarrow$ **CVT**$(f, \cdot, \Theta)$
3: // Notice: final Model is trained by f' on all available data; final estimate is provided by basic CV (no tuning) since f' returns a single model each time
4: $\langle M, L_{NCV}, \Pi \rangle \leftarrow$ **CV**(f', D)
5: **Return** $\langle M, L_{NCV} \rangle$

models trained for estimation. The total count is exactly $K^2 \times C + K + 1$ models, which is of course computationally expensive as it depends quadratically on the number of folds K.

2.3 The Tibshirani and Tibshirani protocol

To reduce the computational overhead of NCV, Tibshirani and Tibshirani (2009) introduced a new method for estimating and correcting for the bias of CVT without training additional models. We refer to this method as the TT and it is the first work of its kind, inspiring this work.

The main idea of the TT method is to consider, in a sense, each fold a different dataset and serving as an independent example to estimate how much the process of selecting the best configuration out of many incurs optimism. It compares the loss of the final, selected configuration with the one selected in a given fold as an estimate of the bias of the selection process. Let I_k denote the indexes of the samples (rows) of the k-th fold F_k. Furthermore, let j denote the index of the best performing configuration (column of Π), as computed by CVT. The bias *TTBias* estimated by the TT method is computed as:

$$TTBias = \frac{1}{K} \sum_{k=1}^{K} (l(y(I_k), \Pi(I_k, j)) - \min_i l(y(I_k), \Pi(I_k, i)))$$

Note that, the average of the first terms $l(y(I_k), \Pi(I_k, j))$ in the sum is the average loss of the best configuration computed by CVT, L_{CVT}. Thus, *TTBias* can be rewritten as:

$$TTBias = L_{CVT} - \frac{1}{K} \sum_{k=1}^{K} \min_i l(y(I_k), \Pi(I_k, i))$$

The final performance estimate is:

$$L_{TT} = L_{CVT} + TTBias$$

The pseudo-code is presented in Algorithm 4 where it is clear that the TT does not train new models, employs the out-of-sample predictions of all models and corresponding configurations, and returns the same final model as both the CVT and the NCV. It is also clear that when the same configuration is selected on each fold as the final configuration, the bias estimate is zero.

Observe that the bias estimate of TT obeys $0 \leq TTBias \leq L_{CVT}$. Thus, the final estimate L_{TT} is always between L_{CVT} and $2 \times L_{CVT}$. However, the main disadvantage of TT is that there are non-contrived cases where it over-corrects the loss. As an example of the former, consider the extreme case of classification, 0-1 loss and leave-one-out CV where each fold

Algorithm 4 TT(f, $D = \{F_1, \ldots, F_K\}$, Θ): Cross-Validation with Tuning, Bias removal using the TT method

Input: Learning method f, Data matrix $D = \{\langle x_j, y_j \rangle\}_{j=1}^{N}$ partitioned into about equally-sized folds F_i, set of configurations Θ
Output: Model M, Performance estimation L_{TT}
1: // Notice: the final Model is the same as in CVT
2: $\langle M, L_{CVT}, \Pi \rangle \leftarrow$ **CVT**(f, D, Θ)
3: **for** $k = 1$ to K **do**
4: // Compute bias estimate for fold k
5: $TTBias_k \leftarrow l(y(I_k), \Pi(I_k, j)) - \min_i l(y(I_k), \Pi(I_k, i))$
6: **end for**
7: $TTBias \leftarrow \frac{1}{K} \sum_{k=1}^{K} TTBias_k$
8: $L_{TT} \leftarrow L_{CVT} + TTBias$
9: **Return** $\langle M, L_{TT} \rangle$

contains a single instance. Then it is likely, especially if many configurations have been tried, that there always is a configuration that correctly predicts the held-out sample in each fold. Thus, in this scenario the bias estimate will be exactly equal to the loss of the selected configuration and so $L_{TT} = 2L_{CVT}$. If for example in a multi-class classification problem, the selected configuration has an estimated 0–1 loss of 70%, the TT method will adjust it to return 140% loss estimate! Such problems are very likely to be observed with few samples and if many configurations are tried. For reliable estimation of the bias, the TT requires relatively large folds, but it is exactly the analyses with overall small sample size that need the bias estimation the most. For the same reason, it is less reliable for performance metrics such as the AUC or the concordance index (in survival analysis) that require several predictions to compute; thus, estimating these metrics in small folds is totally unreliable.

3 The Bootstrap Bias Corrected Cross-Validation (BBC-CV)

The bootstrap (Efron and Tibshirani 1993) has been developed and applied extensively to estimate in a non-parametric way the (unknown) distribution of a statistic b_o computed for a population (dataset). The main idea of the bootstrap is to sample with replacement from the given dataset multiple times (e.g., 500), each time computing the statistic $b_i, i = 1, \ldots, B$ on the resampled dataset. The empirical distribution of b_i, under certain broad conditions approaches the unknown distribution of b_o. Numerous variants have appeared for different statistical tasks and problems (see Davison and Hinkley 1997).

In machine learning, for estimation purposes the idea of bootstrapping datasets has been proposed as an alternative to the CV. Specifically, to produce a performance estimate for a method f multiple training sets are produced by bootstrap (uniform re-sampling with replacement of rows of the dataset), a model is trained and its performance is estimated on the out-of-sample examples. On average, random re-sampling with replacement results in 63.2% of the original samples included in each bootstrap dataset and the rest serving as out-of-sample test sets. The protocol has been compared to the CV in Kohavi (1995) concluding that the CV is preferable.

The setting we explore in this paper is different than what described above since we examine the case where one is also tuning. A direct application of the bootstrap idea in such settings would be to substitute CVT (instead of CV) with a bootstrap version where not one but all configurations are tried on numerous bootstrap datasets, the best is selected, and its

performance is estimated as the average loss on the out-of-sample predictions. This protocol would require the training of $B \times C$ models, where B is the number of bootstraps. Typical applications of the bootstrap require B to be in the range of a few hundreds to thousands, which is unacceptably high in this context. Setting B to equal the number of folds would obviously result to the same number of trained models for both the bootstrap and the CV. However, Kohavi (1995) experiments provide evidence that the bootstrap can have "extremely large" bias on some problems even with 100 bootstrapped datasets (iterations).

Before proceeding with the proposed method, let us define a new important function $\mathbf{css}(\Pi, y)$ standing for *configuration selection strategy*, where Π is a matrix of out-of-sample predictions and y is a vector of the corresponding true labels. Recall that Π contains N rows and C columns, where N is the sample size and C is the number of configurations so that $[\Pi]_{ij}$ denotes the out-of-sample prediction of on the i-th sample of the j-th configuration. The function **css** returns the index of the best-performing configuration according to some criterion. The simplest criterion, also employed in this paper, is to select the configuration with the minimum average loss:

$$\mathbf{css}(\Pi, y) = \arg\min_j l(y, \Pi(:, j))$$

where we again employ the Matlab index notation $\Pi(:, j)$ to denote the vector in column j of matrix Π, i.e., all pooled out-of-sample predictions of configuration j. However, by explicitly writing the selection as a new function, one can easily implement other selection criteria that consider, not only the out-of-sample loss, but also the complexity of the models produced by each configuration.

We propose the Bootstrap Bias Corrected CV method (BBC-CV), for efficient and accurate performance estimation. The pseudo-code is shown in Algorithm 5. BBC-CV uses the out-of-sample predictions Π returned by CVT. It creates B bootstrapped matrices $\Pi^b, b = 1, \ldots, B$ and the corresponding vectors of true labels y^b by sampling N rows of Π with replacement. Let $\Pi^{\setminus b}, b = 1, \ldots, B$ denote the matrices containing the samples in Π and not in Π^b (denoted as $\Pi \setminus \Pi^b$), and $y^{\setminus b}$ their corresponding vectors of true labels. For each bootstrap iteration b, BBC-CV: (a) applies the configuration selection strategy $\text{css}(\Pi^b, y^b)$ to select the best-performing configuration i, and (b) computes the loss L_b of configuration i as $L_b = l(y^{\setminus b}, \Pi(:,\text{i})^{\setminus b})$. Finally, the estimated loss L_{BBC} is computed as the average of L_b over all bootstrap iterations.

BBC-CV differs from the existing methods in two key points: (a) the data that are being bootstrapped are in the matrix Π of the pooled out-of-sample predictions computed by CVT (instead of the actual data in D), and (b) the method applied on each bootstrap sample is the configuration selection strategy **css** (not the learning method f). Thus, performance estimation can be applied with minimal computational overhead, as *no new models need to be trained*.

A few comments on the BBC-CV method now follow. First, notice that if a single configuration is always selected as best, the method will return the bootstrapped mean loss (on the excluded samples) of this configuration instead of the mean loss on the original predictions. The first asymptotically approaches the second as the number of bootstrap iterations increase and they will coincide. A single configuration may always selected for two reasons: either only one configuration was cross-validated or one configuration dominates all others with respect to the selection criterion. In both these cases the BBC-CV estimate will approach the CVT estimate.

Second, BBC-CV simultaneously considers a bootstrap sample from all predictions of all configurations, not only the predictions pertaining to a single fold each time. Thus, unlike

Algorithm 5 BBC-CV(f, $D = \{F_1, \ldots, F_K\}$, Θ): Cross-Validation with Tuning, Bias removal using the BBC method

Input: Learning method f, Data matrix $D = \{\langle x_j, y_j \rangle\}_{j=1}^{N}$ partitioned into approximately equally-sized folds F_i, set of configurations Θ
Output: Model M, Performance estimation L_{BBC}, 95% confidence interval $[lb, ub]$
1: // Notice: the final Model is the same as in CVT
2: $\langle M, L_{CVT}, \Pi \rangle \leftarrow$ **CVT**(f, D, Θ)
3: **for** $b = 1$ to B **do**
4: $\quad \Pi^b \leftarrow$ sample with replacement N rows of Π
5: $\quad \Pi^{\setminus b} \leftarrow \Pi \setminus \Pi^b$ // get samples in Π and not in Π^b
6: $\quad$ // Apply the configuration selection method on the bootstrapped out-of-sample predictions
7: $\quad j \leftarrow$ **ccs**(Π^b, y^b)
8: $\quad$ // Estimate the error of the selected configuration on predictions not selected by this bootstrap
9: $\quad L_b \leftarrow l(y^{\setminus b}, \Pi(:, j)^{\setminus b})$
10: **end for**
11: $L_{BBC} = \frac{1}{B} \sum_{b=1}^{B} L_b$
12: // Compute 95% confidence interval; $L_{(k)}$ denotes the k-th value of L_b's in ascending order
13: $[lb, ub] = [L_{(0.025 \cdot B)}, L_{(0.975 \cdot B)}]$
14: **Return** $\langle M, L_{BBC}, [lb, ub] \rangle$

TT, it is robust even when folds contain only one or just a few samples. For the same reason, it is also robust when the performance metric is the AUC (or a similar metric) and requires multiple predictions to be computed reliably. There is one caveat however, with the use of BCC-CV and the AUC metric: *because BBC-CV pools together predictions from different folds, and thus different models (although produced with the same configuration), the predictions in terms of scores have to be comparable (in the same scale) for use with the AUC*.[2] Finally, we note that we presented BBC in the context of K-fold CV, but the main idea of bootstrapping the pooled out-of-sample predictions of each configuration can be applied to other protocols. One such protocol is the hold-out where essentially there is only one fold. Similarly, it may happen that an implementation of K-fold CV, to save computational time decides to terminate only after a few folds have been employed, e.g., because the confidence intervals of performance are tight enough and there is no need to continue. We call the latter the *incomplete CV protocol*. Again, even though predictions are not available for all samples, BBC-CV can be applied to the predictions of any folds that have been employed for tuning.

3.1 Computing confidence intervals with the bootstrap

The idea of bootstrapping the out-of-sample predictions can not only correct for the bias, but also trivially be applied to provide confidence intervals of the loss. $1 - \alpha$ (commonly 95%) confidence intervals for a statistic L_0 are provided by the bootstrap procedure by computing the population of bootstrap estimates of the statistics L_1,...,L_B and considering an interval $[lL, uL]$ that contains p percentage of the population (Efron and Tibshirani 1993). The parameter 1 - α is called the confidence level of the interval. The simplest approach to compute such intervals is to consider the ordered statistics $L_{(1)}$,...,$L_{(B)}$, where $L_{(i)}$ denotes the i -th value of L's in ascending order, and take the interval $[L_{(\alpha/2 \cdot B)}, L_{((1-\alpha/2) \cdot B)}]$, excluding a probability

[2] As an example consider the following scenario: a configuration that employs a feature selection method, then a linear Support Vector Machine. Now consider the case where in one training set the feature selection returns 4 features and another where it returns 5. The SVM models built on these feature sets rank predictions according to the distance to the separating hyper-plane. This distance is computed in a 4 and a 5 dimensional space for the two models respectively. This makes the distances computed from different models incomparable with each other and the computation of AUC from the pooled predictions problematic.

mass of $\alpha/2$ on each side of extreme values. For example, when $\alpha = 0.05$ and B = 1000 we obtain $[lL,uL] = [L_{(25)}, L_{(975)}]$. Other variants are possible and could be applied, although outside the scope of this paper. For more theoretical details on the bootstrap confidence intervals and different methods for constructing them, as well as a comparison of them, see Efron and Tibshirani (1993).

3.2 BCC-CV with repeats

When sample size is small, the variance of the estimation of the performance is large, even if there is no bias. This is confirmed in Tsamardinos et al. (2015), Krstajic et al. (2014) and Seibold et al. (2017) empirically on several real datasets. A component of the variance of estimation stems from the specific random partitioning to folds. To reduce this component it is advisable to repeat the estimation protocol multiple times with several fold partitions, leading to the Repeated Cross-Validation protocol and variants.

Applying the BBC-CV method with multiple repeats is possible with the following minimal changes in the implementation: We now consider the matrix Π of the out-of-sample predictions of the models to be three dimensional with $[\Pi]_{ijk}$ to denote the out-of-sample prediction (i.e, when the example was held-out during training) on the i-th example, of the j-th configuration, in the k-th repeat. Note that *predictions for the same instance x_i in different repeats are correlated*: they all tend to be precise for easy-to-predict instances and tend to be wrong for outliers that do not fit the assumptions of the configuration correctly. Thus, predictions on the same instance for different repeats have to all be included in a bootstrap sample or none at all. In other words, as in Algorithm 5, what is resampled with replacement to create the bootstrap data are the indexes of the instances. Other than that, the key idea remains the same as in Algorithm 5.

3.3 Related work

There are two additional major works that deal with performance estimation when tuning (model selection) is included. The first one by Bernau et al. (2013) introduced a variant of a bias correction method as a smooth analytic alternative to NCV, called WMCS. Similarly to BBC-CV, WMCS estimates the optimism of trying multiple configurations and removes it from the performance estimate (called *shrinking* in the authors' terminology, Step 2c, page 697). However, to compute the optimism WMCS relies on parametric modeling of the joint distribution of the means of all configurations and numerical integration of this distribution (Eq. 13). Specifically, WMCS assumes the joint distribution to be a multivariate normal. This is an unrealistic assumption for classification problems as the authors acknowledge: "Of course, the normality assumption can not hold exactly since the considered errors are averages of binary variables" (ranging from zero to one and not from minus to plus infinity). But, there are other reasons why the distribution may not be normal or any trivially modeled distribution: consider the case where two algorithms are tried, each instantiated over a range of hyper-parameter values. This gives rise to two groups of configurations with correlated mean performances. The distribution of the means of all configurations will then create a bimodal distribution. BBC-CV avoids directly modeling the distribution of the configuration means, and any related parametric assumptions, as well as the use of numerical integration by the use of bootstrapping; it directly computes the optimism without modeling in detail the joint of the means of configurations. Subsequent independent work by Ding et al. (2014) also reports empirical problems with the WMCS method, and specifically that it provides context-dependent estimates: optimistic for small sample sizes $N \leq 40$ and conservative

for larger N. Finally, WMCS requires a parametric assumption for modeling the means of configurations that depends on the performance metric (accuracy, mean squared error, coefficient of determination, etc.). Thus, it cannot directly be applied as presented to all types of analysis tasks and performance metrics, unlike the BBC-CV.

Ding et al. (2014) proposed a resampling-based inverse power law (IPL) method for bias correction and compared its performance to those of TT, NCV, and WMC/WMCS on both simulated and real datasets. The error rate of each classifier is estimated 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. Additionally, if the sample size of the original dataset is small, the method will provide unstable estimates. Lastly, the IPL method has higher computational cost compared to TT and the WMC/WMCS methods.

4 Bootstrap Bias Corrected with Dropping Cross-Validation (BBCD-CV)

In this section, we present a second use of the idea to bootstrap the pooled out-of-sample predictions of each configuration. Specifically, we explore the idea to use the current out-of-sample predictions of each configurations to determine the inferior configurations and early-stop further computations employing them for training modes.

4.1 The BBCD-CV protocol

We now explain in detail how the pooled out-of-sample predictions of each configuration can be employed as part of a statistical criterion that determines whether a configuration's performance is with high probability inferior than the performance of the current best configuration. If this is indeed the case, the dominated configuration can be early dropped from further consideration, in the sense that no additional models on subsequent folds will be trained under this configuration. If a relatively strict threshold is employed for the probability then the procedure will typically select the optimal configuration at the end of the CVT and thus, the prediction performance of the final model will not be affected. The Early Dropping scheme can lead to substantial computational savings as numerous configurations can be dropped after just a few folds before completing the full K-fold CV on them.

Specifically, let θ be the index of a given configuration and θ_o the index of the current best configuration and denote with $l_N(\theta)$ and $l_N(\theta_o)$ the true average loss of these configurations when trained on datasets from the distribution of the problem at hand of size N. Since all models are produced by the same dataset size stemming from excluding a single fold, we can actually drop the subscript N. The probability $P(l(\theta) > l(\theta_o))$ is estimated for every such θ still under consideration at the end of each fold i.e., as soon as new out-of-sample predictions are accrued for each configuration.

To perform this test, the current, pooled, out-of-sample predictions of all configurations still under consideration Π are employed to identify the best current configuration $\theta_o = \mathbf{css}(\Pi, y)$. Subsequently, Π's rows are bootstrapped to create matrices $\Pi^1, \ldots, \Pi^B$ and corresponding label matrices $y^1, \ldots, y^B$. From the population of these bootstrapped matrices the probability p_θ of a given configuration θ to exhibit a worse performance than θ_o is estimated as the percentage of times its loss is higher than that of θ's, i.e., $\hat{p}_\theta = \frac{1}{B}\#\{l(y^b, \Pi^b(:$

$, \theta)) > l(y^b, \Pi^b(:, \theta_o)), b = 1, \ldots, B\}$. If $\hat{p}_\theta > \alpha$ for some significance threshold (e.g., $\alpha = 0.99$), configuration θ is dropped.

A few comments on the procedure above. It is a heuristic procedure mainly with focus on computational efficiency, not statistical theoretical properties. Ideally, the null hypothesis to test for each configuration θ would be the hypothesis that θ will be selected as the best configuration at the end of the CVT procedure, given a finite number of folds remain to be considered. If this null hypothesis is rejected for a given θ, θ should be dropped. Each of these hypotheses for a given θ has to be tested in the context of all other configurations that participate in the CVT procedure. In contrast, the heuristic procedure we provide essentially tests each hypothesis H_θ in isolation. For example, it could be the case during bootstrapping, configuration θ exhibits a significant probability of a better loss than θ_o (not dropped by our procedure), but it could be that in all of these cases, it is always dominated by some other configuration θ'. Thus, the actual probability of being selected as best in the end maybe smaller than the percentage of times it appears better than θ_o.

In addition, our procedure does not consider the uncertainty (variance) of the selection of the current best method θ_o. Perhaps, a double bootstrap procedure would be more appropriate in this case (Nankervis 2005) but any such improvements would have to also minimize the computational overhead to be worthwhile in practice.

4.2 Related work

The idea of accelerating the learning process by specifically eliminating under-performing configurations from a finite set, early within the cross-validation procedure, was introduced as early as 1994 by Maron and Moore with *Hoeffding Races* (Maron and Moore 1994). At each iteration of leave-one-out CV (i.e. after the evaluation of a new test point) the algorithm employs the Hoeffding inequality for the construction of confidence intervals around the current error rate estimate of each configuration. Configurations whose intervals do not overlap with those of the best-performing one are eliminated (dropped) from further consideration. The procedure is repeated until the confidence intervals have shrunk enough so that a definite overall best configuration can be identified. However, several test point evaluations may be required before a configuration can clearly be declared the winner.

Following a similar approach, Zheng and Bilenko (2013) applied the concept of early elimination of suboptimal configurations to K-fold CV. They improve on the method by Maron and Moore by incorporating paired hypothesis tests for the comparison of configurations for both discrete and continuous hyper-parameter spaces. At each iteration of CV, all current configurations are tested pairwise and those which are inferior are dropped. Then, power analysis is used to determine the number of new fold evaluations for each remaining configuration given an acceptable false negative rate.

Krueger et al. (2015) introduced the so-called *Fast Cross-Validation via Sequential Testing* (CVST) which uses nonparametric testing together with sequential analysis in order to choose the best performing configuration on the basis of linearly increasing subsets of data. At each step, the Friedman test (1937) or the Cochran's Q test (1950) (for regression and classification tasks respectively) are employed in order to detect statistically significant differences between configurations' performances. Then, the under-performing configurations are further tested through sequential analysis to determine which of them will be discharged. Finally, an early stopping criterion is employed to further speed up the CV process. The winning configuration is the one that has the best average ranking, based on performance, in the last few iterations specified in advance. The disadvantage of CVST is that it initially operates on smaller subsets, thus risking the early elimination of good-performing models when the original dataset is

already small. In comparison to the statistical tests used in Zheng and Bilenko (2013) and Krueger et al. (2015), the bootstrap is a general test, applicable to any type of learning task and measure of performance, and is suitable even for relatively small sample sizes. Finally, BBCD-CV requires that only the value of the significance threshold α is pre-specified while the methods in Zheng and Bilenko (2013) and Krueger et al. (2015) have a number of hyper-parameters to be specified in advance.

5 Empirical evaluation

We empirically evaluate the efficiency and investigate the properties of BBC-CV and BBCD-CV, on both controlled settings and real problems. In particular, we focus on the bias of the performance estimates of the protocols, and on computational time. We compare the results to those of three standard approaches: CVT, TT and NCV. We also examine the tuning (configuration selection) properties of BBC-CV, BBCD-CV and BBC-CV with repeats, as well as the confidence intervals that these methods construct. WMCS and IPL are not included in this empirical comparison, for a variety of reasons, including the need for parametric, metric-specific assumptions (WMCS) and increased computational complexity (IPL) (see Sect. 3, subsection Related Work); in addition, both methods are complex to implement. As the main advantage of the proposed methods are on a conceptual level (simplicity of the approach and broad applicability to almost any type of performance metric and outcome of interest), such empirical evaluation would probably not be very informative.

5.1 Simulation studies

Extensive simulation studies were conducted in order to validate BBC-CV and BBCD-CV, and assess their performance. We focus on binary classification tasks and use classification accuracy as the measure of performance, as it is easier to simulate models with a prespecified accuracy. We examine multiple settings for varying sample size $N \in \{20, 40, 60, 80, 100, 500, 1000\}$, number of candidate configurations $C \in \{50, 100, 200, 300, 500, 1000, 2000\}$, and true performances P of the candidate configurations drawn from different Beta distributions $Be(a, b)$ with $(a, b) \in \{(9, 6), (14, 6), (24, 6), (54, 6)\}$. These betas provide configurations with mean performance values $\mu \in \{0.6, 0.7, 0.8, 0.9\}$ and variances of these performances of 0.015, 0.01, 0.0052, 0.0015. These choices result in a total of 196 different experimental settings. We chose distributions with small variances since these are the most challenging cases where the models have quite similar performances.

For each setting, we generate a simulated matrix of out-of-sample predictions Π. First, a true performance value P_j, $j = 1, \ldots, C$, sampled from the same beta distribution, is assigned to each configuration c_j. Then, the sample predictions for each c_j are produced as $\Pi_{ij} = \mathbb{1}(r_i < P_j), i = 1, \ldots, N$, where r_i are random numbers sampled uniformly from $(0, 1)$, and $\mathbb{1}(condition)$ denotes the unit (indicator) function. Notice that there is no need to simulate the actual training of the models, just the predictions of these models so that they obtain a prespecified predictive accuracy.

Then, the BBC-CV, BBCD-CV, CVT, TT, and NCV protocols for tuning and performance assessment of the returned model are applied. We set the number of bootstraps $B = 1000$ for the BBC-CV method, and for the BBCD-CV we set $B = 1000$ and the dropping threshold to $a = 0.99$. We applied the same split of the data into $K = 10$ folds for all the protocols. Consequently, all of them, with the possible exception of the BBCD-CV, select and return

the same predictive model with different estimations of its performance. The internal cross-validation loop of the NCV uses $K = 9$ folds. The whole procedure was repeated 500 times for each setting, leading to a total of 98,000 generated matrices of predictions, on which the protocols were applied. The results presented are the averages over the 500 repetitions. The code, in Matlab, implementing the simulation studies can be downloaded from https://github.com/mensxmachina/BBC-CV.

5.1.1 Bias estimation

The bias of the estimation is computed as $\widehat{Bias} = \hat{P} - P$, where $\hat{P}$ and P denote the estimated and the true performance of the selected configuration, respectively. A positive bias indicates a lower true performance than the one estimated by the corresponding performance estimation protocol and implies that the protocol is optimistic (i.e. overestimates the performance), whereas a negative bias indicates that the estimated performance is conservative. Ideally, the estimated bias should be 0, although a slightly conservative estimate is also acceptable in practice.

Figure 1 shows the average estimated bias for models with average true classification accuracy $\mu = 0.6$, over 500 repetitions, of the protocols under comparison. Each panel corresponds to a different protocol (specified in the title) and shows the bias of its performance estimate relatively to the sample size (horizontal axis) and the number of configurations tested (different plotted lines). We omit results for the rest of the tested values of μ as they are similar.

The CVT estimate of performance is optimistically biased in all settings with the bias being as high as 0.17 points of classification accuracy. We notice that the smaller the sample size, the more CVT overestimates the performance of the final model. However, as sample size increases, the bias of CVT tends to 0. Finally, we note that the bias of the estimate also grows as the number of models under comparison becomes greater, although the effect is relatively small in this experiment. The behaviour of TT greatly varies for small sample sizes (≤ 100), and is highly sensitive to the number of configurations. On average, the protocol is optimistic (not correcting for the bias of the CVT estimate) for sample size $N \in \{20, 40\}$, and over-corrects, for $N \in \{60, 80, 100\}$. For larger sample size (≥ 500), TT is systematically conservative, over-correcting the bias of CVT. NCV provides an almost unbiased estimation of performance, across all sample sizes. However, recall that it is computationally expensive since the number of models that need to be trained depends quadratically on the number of folds K.

BBC-CV provides conservative estimates, having low bias which quickly tends to zero as sample size increases. Compared to TT, it is better fitting for small sample sizes and produces more accurate estimates overall. In comparison to NCV, BBC-CV is somewhat more conservative with a difference in the bias of 0.013 points of accuracy on average, and 0.034 in the worst case (for $N = 20$); on the other hand however, BBC-CV is more computationally efficient. BBCD-CV displays similar behaviour to BBC-CV, having lower bias which approaches zero faster. It is on par with NCV, having 0.005 points of accuracy higher bias on average, and 0.018 in the worst case. As we show later on, BBCD-CV is up to one order of magnitude faster than CVT, and consequently two orders of magnitude faster than NCV.

In summary, the proposed BBC-CV and BBCD-CV methods produce almost unbiased performance estimates, and perform only slightly worse in small sample settings than the computationally expensive NCV. As expected, CVT is overly optimistic, and thus should not be used for performance estimation purposes. Finally, the use of TT is discouraged, as (a) its performance estimate varies a lot for different sample sizes and numbers of configurations,

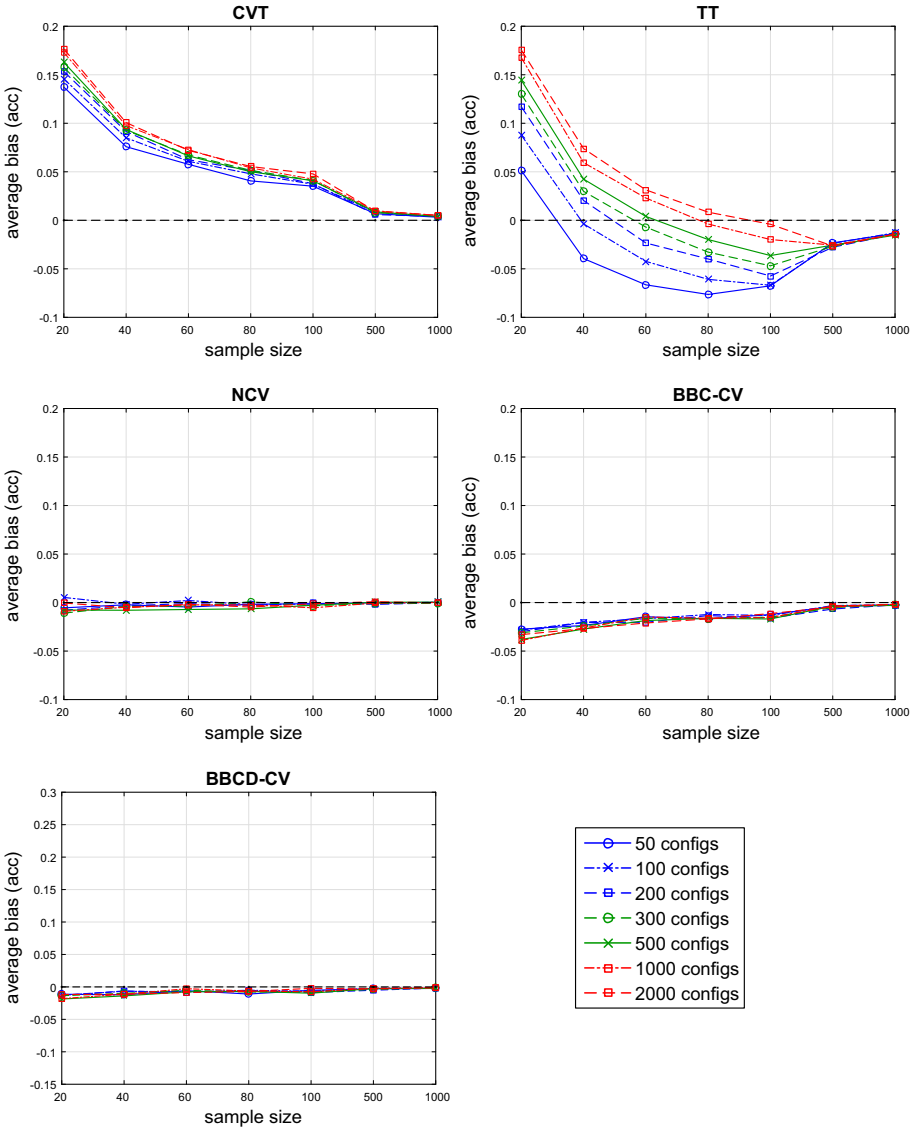

Fig. 1 Average (over 500 repeats) estimated bias of accuracy estimation of the CVT, TT, NCV, BBC-CV and BBCD-CV. The average true classification accuracy of all configurations is 60%. CVT over-estimates performance in all settings. TT's behaviour varies for sample size $N < 500$ and is conservative for $N \geq 500$. NCV provides almost unbiased estimates of performance, while BBC-CV is more conservative with a difference in the bias of 0.013 points of accuracy on average. BBCD-CV is on par with NCV

and (b) it overestimates performance for small sample sizes, which are the cases where bias correction is needed the most.

Table 1 Datasets' characteristics. pr/nr denotes the ratio of positive to negative examples in a dataset. $|D_{pool}|$ 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 of a model is estimated

Name	#Samples	#Variables	pr/nr	$\|D_{pool}\|$	$\|D_{holdout}\|$	Source
christine	5418	1636	1	1625	3793	Guyon et al. (2015)
jasmine	2984	144	1	895	2089	Guyon et al. (2015)
philippine	5832	308	1	1749	4082	Guyon et al. (2015)
madeline	3140	259	1.01	942	2198	Guyon et al. (2015)
sylvine	5124	20	1	1537	3587	Guyon et al. (2015)
gisette	7000	5000	1	2100	4900	Guyon et al. (2004)
madelon	2600	500	1	781	1819	Guyon et al. (2004)
dexter	600	20000	1	180	420	Guyon et al. (2004)
gina	3468	970	1.03	1041	2427	Guyon et al. (2006)

5.2 Real datasets

After examining the behaviour of BBC-CV and BBCD-CV on controlled settings, we investigate their performance on real datasets. Again we focus on the binary classification task but now we use the AUC as the metric of performance, as it is independent of the class distribution. All of the datasets included in the experiments come from popular data science challenges [NIPS 2003 (Guyon et al. 2004); WCCI 2006 (Guyon et al. 2006); ChaLearn AutoML (Guyon et al. 2015)]. Table 1 summarizes their characteristics. The domains of application of the ChaLearn AutoML challenge's datasets are not known, however the organizers claim that they are diverse and were chosen to span different scientific and industrial fields. *gisette* (Guyon et al. 2004) and *gina* (Guyon et al. 2006) are handwritten digit recognition problems, *dexter* (Guyon et al. 2004) is a text classification problem, and *madelon* (Guyon et al. 2004) is an artificially constructed dataset characterized by having no single feature that is informative by itself.

The experimental set-up is similar to the one used by Tsamardinos et al. (2015). Each original dataset D was split into two stratified subsets; D_{pool} which consisted of 30% of the total samples in D, and $D_{holdout}$ which consisted of the remaining 70% of the samples. For each original dataset with the exception of *dexter*, D_{pool} was used to sample (without replacement) 20 sub-datasets for each sample size $N \in \{20, 40, 60, 80, 100, 500\}$. For the *dexter* dataset we sampled 20 sub-datasets for each $N \in \{20, 40, 60, 80, 100\}$. We created a total of $8 \times 20 \times 6 + 20 \times 5 = 1060$ sub-datasets. $D_{holdout}$ was used to estimate the true performance of the final, selected model of each of the protocols tested.

The set Θ (i.e. the search grid) explored consists of 610 configurations. These resulted from various combinations of preprocessing, feature selection, and learning methods and different values for their hyper-parameters. The preprocessing methods included imputation, binarization (of categorical variables) and standardization (of continuous variables) and were used when they could be applied. For feature selection we used the SES algorithm (Lagani et al. 2017) with $alpha \in \{0.05, 0.01\}$, and $k \in \{2, 3\}$ and we also examined the case of no feature selection (i.e., a total of 5 cases/choices). The learning algorithms considered were Random Forests (Breiman 2001), SVMs (Cortes and Vapnik 1995), and LASSO (Tibshirani 1996). For Random Forests the hyper-parameters and values tried are $numTrees = 1000$, $minLeafSize \in \{1, 3, 5\}$ and $numVarToSample \in \{(0.5, 1, 1.5, 2) * \sqrt{numVar}\}$, where $numVar$

is the number of variables of the dataset. We tested SVMs with linear, polynomial and radial basis function (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 *alpha* $\in \{0.001, 0.5, 1.0\}$ (*alpha* $= 1$ represents lasso regression, other values represent elastic net optimization, and *alpha* close to 0 approaches ridge regression) and 10 different values for *lambda* which were created independently for each dataset using the glmnet library (Friedman et al. 2010).

We performed tuning and performance estimation of the final model using CVT, TT, NCV, BBC-CV, BBCD-CV, and BBC-CV with 10 repeats (denoted as BBC-CV^{10}) for each of the 1060 created sub-datasets, leading to more than 135 million trained models. We set $B = 1000$ for the BBC-CV method, and $B = 1000$, $a = 0.99$ for the BBCD-CV method. We applied the same split of the data into $K = 10$ stratified folds for all the protocols. The inner cross-validation loop of NCV uses $K = 9$ folds. For each protocol, original dataset D, and sample size N, the results are averaged over the 20 randomly sampled sub-datasets.

To compute the AUC (and similar metrics like the concordance index) during CV-like protocols one could pool all predictions first and then compute the AUC on the pooled set of predictions. Alternatively, one could compute the AUC on each fold and average on all folds (see also Sect. 3). The final selection of the best configuration and estimation of performance may be different depending the method. However, in preliminary experiments (Greasidou 2017) we found that the two methods perform similarly in terms of model performance and bias of estimation. Notice that the pooling method cannot be applied to the TT method since the latter depends on estimates of performance in each fold individually. In the experiments that follow, all other methods using pooling to compute AUC except the TT and NCV (as it is standard in the literature).

5.2.1 Bias estimation

The bias of estimation is computed as in the simulation studies, i.e., $\widehat{Bias} = \hat{P} - P$, where $\hat{P}$ and P denote the estimated and the true performance of the selected configuration, respectively. In Fig. 2 we examine the average bias of the CVT, TT, NCV, BBC-CV, and BBCD-CV estimates of performance, on all datasets, relative to sample size. We notice that the results are in agreement with those of the simulation studies. In particular, CVT is optimistically biased for sample size $N \leq 100$ and its bias tends to zero as N increases. TT over-estimates performance for $N = 20$, its bias varies with datasets for $N = 40$, and it over-corrects the bias of CVT for $N \geq 60$. TT exhibits the worst results among all protocols except CVT.

Both NCV and BBC-CV have low bias (in absolute value) regardless of sample size, though results vary with the dataset. BBC-CV is mainly conservative with the exception of the *madeline* dataset for $N = 40$ and the *madelon* dataset for $N \in \{60, 80, 100\}$. NCV is slightly optimistic for the *dexter* and *madeline* datasets for $N = 40$ with a bias of 0.033 and 0.031 points of AUC respectively. BBCD-CV has, on average, greater bias than BBC-CV for $N \leq 100$. For $N = 500$, its bias shrinks and becomes identical to that of BBC-CV and NCV.

5.2.2 Relative performance and speed up of BBCD-CV

We have shown that for large sample sizes ($N = 500$) BBCD-CV provides accurate estimates of performance of the model it returns, comparable to those of BBC-CV and NCV. How well does this model perform though? In this section, we evaluate the effectiveness of BBCD-CV

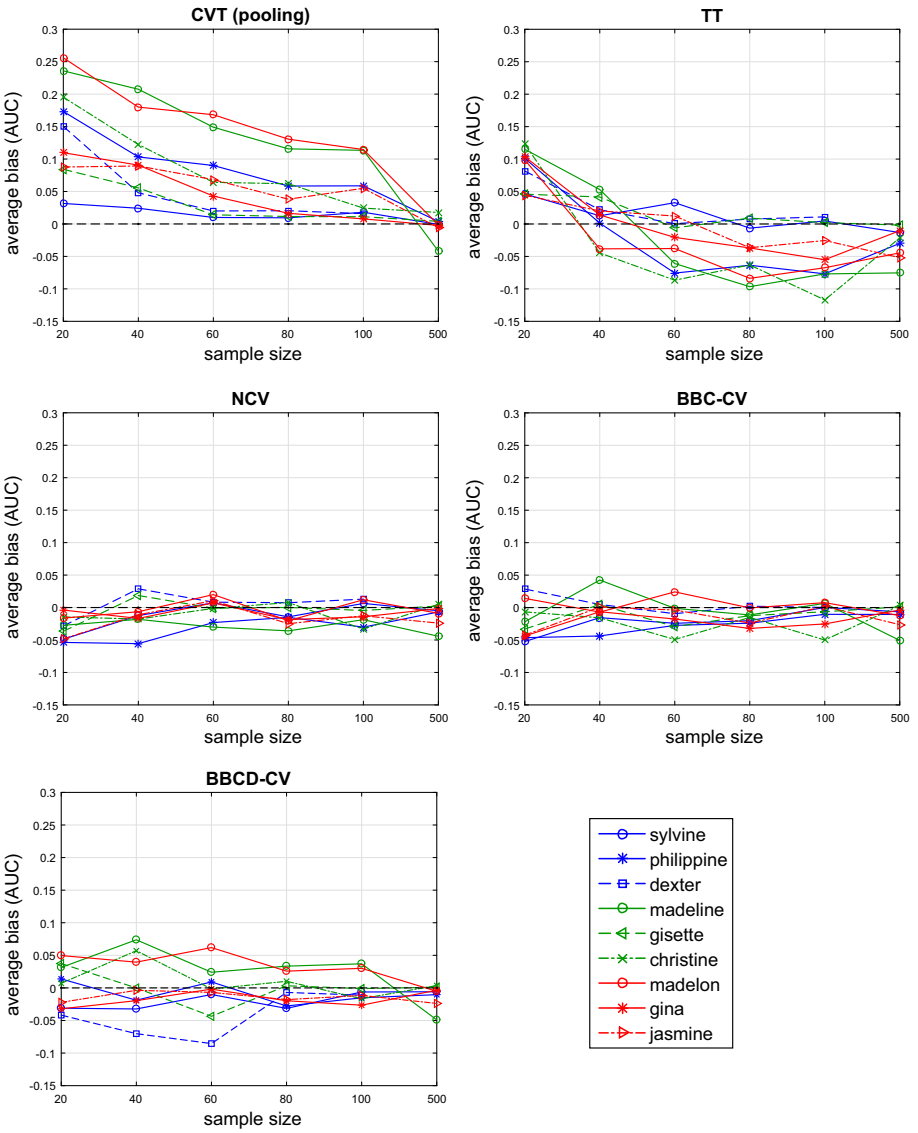

Fig. 2 Average estimated bias (over 20 sub-datasets for each original dataset) of the CVT, TT, NCV, BBC-CV and BBCD-CV estimates of performance. CVT is optimistically biased for sample size $N \leq 100$. TT's bias varies with sample size and dataset, and it is mainly over-conservative for $N \geq 80$. NCV and BBC-CV, both have low bias though results vary with dataset. BBCD-CV has, on average, greater bias than BBC-CV for $N \leq 100$ and identical for $N = 500$

in terms of its tuning (configuration selection) properties, and its efficiency in reducing the computational cost of CVT.

Figure 3 shows the relative average true performance of the models returned by the BBCD-CV and CVT protocols, plotted against sample size. We remind here that for each of the 20 sub-datasets of sample size $N \in \{20, 40, 60, 80, 100, 500\}$ sampled from D_{pool}, the true

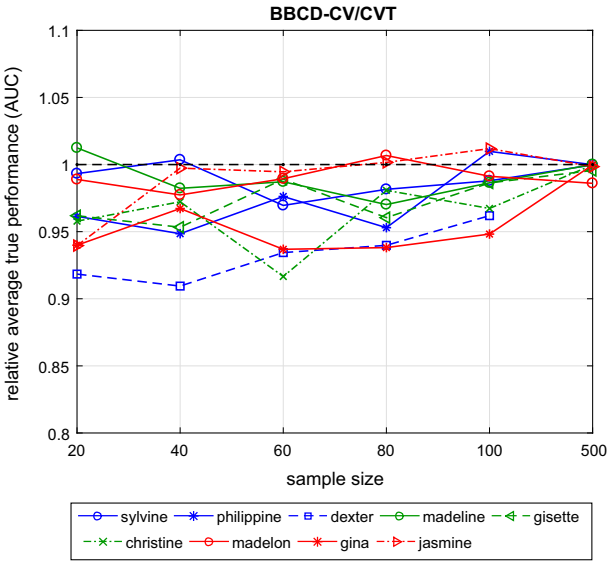

Fig. 3 Relative average true performance of the models returned by the BBCD-CV and CVT. For $N \leq 100$ the loss in performance varies greatly with dataset, however, for $N = 500$ there is negligible to no loss in performance. If N if fairly large, BBCD-CV will accelerate the CVT procedure without sacrificing the quality of the resulting model or the accuracy of its performance estimate

performance of the returned model is estimated on the $D_{holdout}$ set. We notice that, for $N \leq 100$ the loss in performance varies greatly with dataset and is quite significant; up to 9.05% in the worst case (*dexter* dataset, $N = 40$). For $N = 500$, however, there is negligible to no loss in performance. Specifically, for the *sylvine, philippine, madeline, christine* and *gina* datasets there is no loss in performance when applying BBCD-CV, while there is 0.44 and 0.15% loss for the *gisette* and *jasmine* datasets, respectively. *madelon* exhibits the higher average loss of 1.4%. We expect the difference in performance between BBCD-CV and CVT to shrink even further with larger sample sizes.

We investigated the reason of the performance loss of BBCD-CV for low sample sizes ($N \leq 100$). We observed that, in most cases the majority of configurations ($> 95\%$) were dropped very early within the CV procedure (in the first couple of iterations). With 10-fold CV, the number of out-of-sample predictions with $N \leq 100$ samples ranges from 2 to 10, which are not sufficient for the bootstrap test to reliably identify under-performing configurations. This observation leads to some practical considerations and recommendations. For small sample sizes, we recommend to start dropping configurations with BBCD-CV only after an adequate number of out-of-sample predictions become available. An exact number is hard to determine, as it depends on many factors, such as the analyzed dataset and the set of configurations tested. Given that with $N = 500$ BBCD-CV incurs almost no loss in performance, we recommend a minimum of 50 out-of-sample predictions to start dropping configurations, although a smaller number may suffice. For example, with $N = 100$, this would mean that dropping starts after the fifth iteration. Finally, we note that dropping is mostly useful with larger sample sizes (i.e. for computationally costly scenarios), which are also the cases where BBCD-CV is on par with BBC-CV and NCV, in terms of tuning and performance estimation.

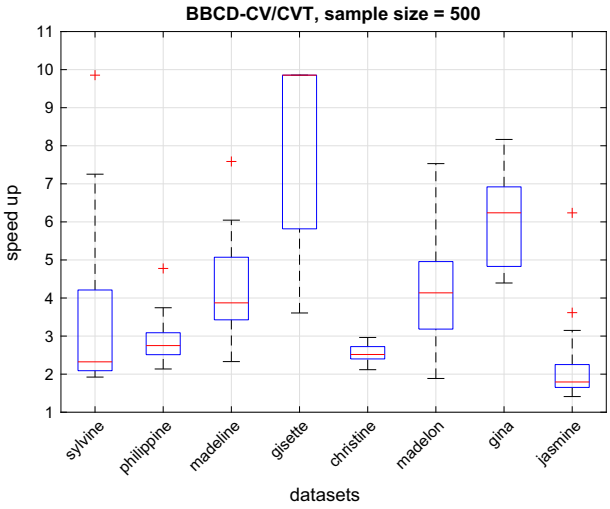

Fig. 4 The speed-up of BBCD-CV over CVT is shown for sample size $N = 500$. It is computed as the ratio of models trained by CVT over BBCD-CV. Typically, BBCD-CV achieves a speed-up of 2-5, up to 10 for the *gisette* dataset. Overall, using BBCD-CV results in a significant speed boost, without sacrificing model quality or performance estimation

Next, we compare the computational cost of BBCD-CV to CVT, in terms of total number of models trained. The results for $N = 500$ are shown in Fig. 4. We only focused on the $N = 500$ case, as it is the only case where both protocols produce models of comparable performance. We observe that a speed-up of 2 to 5 is typically achieved by BBCD-CV. For the *gisette* dataset, the speed-up is very close to the theoretical maximum of this experimental setup; the maximum is achieved when almost all configurations are dropped after the first fold and a speed up of K, the number of folds, is achieved. Overall, if sample size is sufficiently large, using dropping is recommended to speed-up CVT without a loss of performance.

Finally, we would like to note that we have also run experiments for $\alpha \in \{0.90, 0.95\}$ which are included in the Master's thesis of one of the authors (see Greasidou 2017). In terms of tuning, the results (accuracy of the final model selected) were not significantly different when compared to $\alpha = 0.99$, however, the number of trained models for some datasets and sample sizes was larger for larger α. We chose to only present the results for $\alpha = 0.99$ in this work since this is the value we suggest using in the general case (in favor of being conservative and trying a larger number of configurations versus being computationally efficient).

5.2.3 Multiple repeats

We repeated the previous experiments, running BBC-CV with 10 repeats of partitioning to different folds (called BBC-CV^{10} hereafter). First, we compare the true performance of the models returned by BBC-CV and BBC-CV^{10}, as well as the bias of the estimation. Ideally, using multiple repeats should result in a better performing model, as the variance of the performance estimation (used by CVT for tuning) due to a specific choice of split for the data is reduced when multiple splits are considered. This comes at a cost of increased computational overhead, which in case of 10 repeats is similar to that of the NCV protocol. To determine which of the approaches is preferable, we also compare the performance of the final models produced by BBC-CV^{10} and NCV.

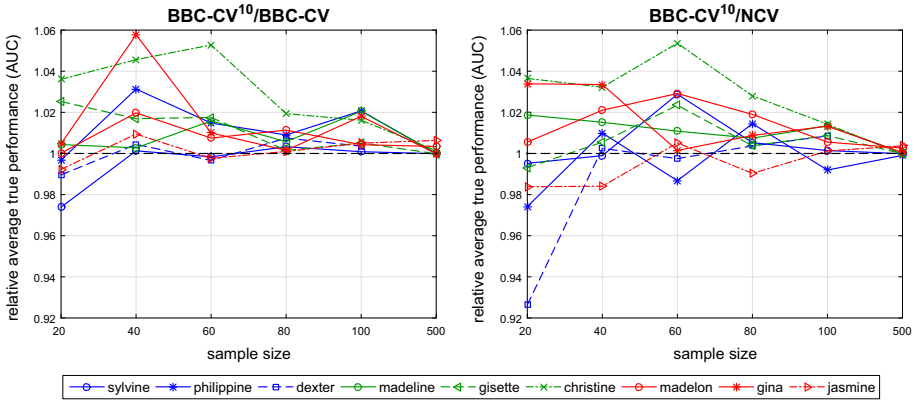

Fig. 5 Relative average true performance of BBC-CV^{10} to BBC-CV (left), and of BBC-CV^{10} to NCV (right). Multiple repeats increase the performance of the returned models, maintaining the accuracy of the performance estimation. If computational time is not a limitation, it is preferable to use BBC-CV^{10} over NCV

Figure 5 (left) shows the relative average true performance of BBC-CV^{10} to BBC-CV with increasing sample size N. We notice that, for $N = 20$ the results vary with dataset, however, for $N \geq 40$, BBC-CV^{10} systematically returns an equally good or (in most cases) better performing model than the one that BBC-CV returns. In terms of the bias of the performance estimates of the two methods, we have found them to be similar.

Similarly, Fig. 5 (right) shows the comparison between BBC-CV^{10} and NCV. We see again that for sample size $N = 20$ the relative average true performance of the returned models vary with dataset. BBC-CV^{10} outperforms NCV for $N \geq 40$ except for the *philippine* and *jasmine* datasets for which results vary with sample size. Thus, if computational time is not a limiting factor, it is still beneficial to use BBC-CV with multiple repeats instead of NCV.

To summarize, we have shown that using multiple repeats increases the quality of the resulting models as well as maintaining the accuracy of the performance estimation. We note that the number 10 was chosen mainly to compare BBC-CV to NCV with $K = 10$ folds on equal grounds (same number of trained models). If time permits, we recommend using as many repeats as possible, especially for low sample sizes. For larger sample sizes, usually one or a few repeats suffice.

5.2.4 Confidence intervals

The bootstrap-based estimation of performance, allows for easy computation of confidence intervals (CIs) as described in Sect. 3.1. We investigated the accuracy of the CIs (calibration) produced by the proposed BBC-CV, BBCD-CV and BBC-CV^{10} protocols. To this end, we computed the coverage of the $\{50\%, 55\%, \ldots, 95\%, 99\%\}$ CIs estimated by the protocols, defined as the ratio of the computed CIs that contain the corresponding true performances of the produced models. For a given sample size, the coverage of a CI was computed over all 20 sub-datasets and 9 datasets. To further examine the effect of multiple repeats on CIs, we computed their average width (over all 20 sub-datasets) for each dataset and different number of repeats (1–10).

Figure 6 shows the estimated coverage of the CIs constructed with the use of the percentile method relative to the expected coverage for the BBC-CV, BBCD-CV, and BBC-CV^{10} protocols. We present results for sample sizes $N = 20$ (left), $N = 100$ (middle), and $N = 500$

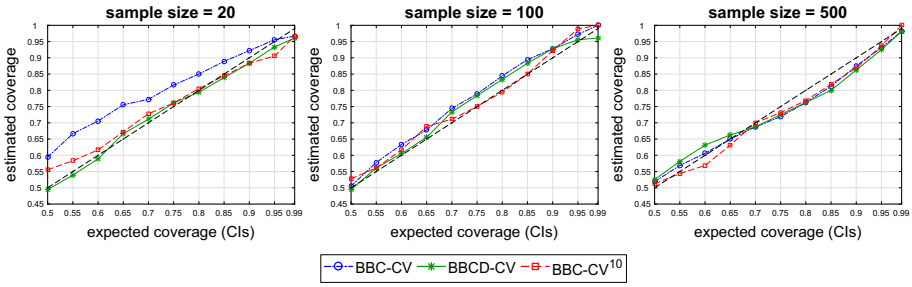

Fig. 6 Coverage of the {50%, 55%, ..., 95%, 99%} CIs returned by BBC-CV, BBCD-CV, and BBC-CV^{10}, defined as the ratio of the estimated CIs that contain the corresponding true performances of the produced models. The CIs are mainly conservative and become more accurate with increasing sample size and multiple repeats

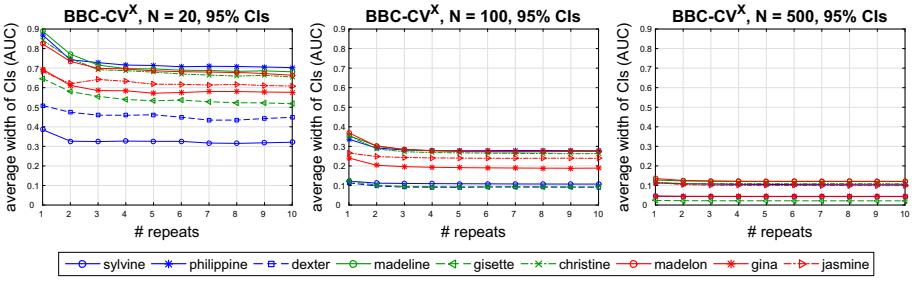

Fig. 7 Average width (over all 20 sub-datasets) of CIs with increasing number of repeats (BBC-CV^X, $X = 1...10$), for each dataset. CIs shrink with increasing sample size and number of repeats

(right). Figure 7 shows, for the same values for N and for each dataset, the average width of the CIs with increasing number of repeats.

We notice that for $N = 20$ the CIs produced by BBC-CV are conservative, that is, they are wider than ought to be. As sample size increases ($N \geq 100$), BBC-CV returns more calibrated CIs which are still conservative. The use of 10 repeats (BBC-CV^{10}) greatly shrinks the width of the CIs and improves their calibration (i.e., their true coverage is closer to the expected one). The same holds when using dropping of under-performing configurations (BBCD-CV). For $N = 500$ the intervals appear to not be conservative. After closer inspection, we saw that this is caused by two datasets (*madeline* and *jasmine*) for which the majority of the true performances are higher than the upper bound of the CI. We note that those datasets are the ones with the highest negative bias (see Fig. 2 for $N = 500$), which implicitly causes the CIs to also be biased downwards, thus failing to capture performance estimates above the CI limits.

In conclusion, the proposed BBC-CV method provides mainly conservative CIs of the true performance of the returned models which become more accurate with increasing sample size. The use of multiple repeats improves the calibration of CIs and shrinks their width, for small sample sizes (< 100). The use of 3-4 repeats seems to suffice and further repeats provide small added value in CI estimation.

6 Discussion and conclusions

Pooling together the out-of-sample predictions during cross-validation of multiple configurations (i.e., combinations of algorithms and their hyper-parameter values that leads to a model) and employing bootstrapping techniques on them addresses in a simple and general way three long-standing, important data analysis tasks: (a) removing the optimism of the performance estimation of the selected configuration, (b) estimating confidence intervals of performance, and (c) dropping from further consideration during tuning inferior configurations. While other methods have also been proposed, they lack the simplicity and the generality in applicability in all types of performance metrics. The ideas above are implemented in the method BBC-CV tackling points (a) and (b) and BBCD-CV that includes (c).

Simulation studies and experiments on real datasets show empirically that BBC-CV and BBCD-CV outperform the alternatives (nested cross-validation and the TT method) by either providing more accurate, almost unbiased, conservative estimates of performance even for smaller sample sizes and/or by having much lower computational cost (speed-up of up to 10). We examined the effect of repeatedly applying our methods on multiple fold partitions of the data, and found that we acquire better results in terms of tuning (i.e., better-performing configurations are selected) compared to BBC-CV and NCV. Finally, in our experiments, the confidence intervals produced by bootstrapping are shown to be mainly conservative, improving with increasing sample size and multiple repeats.

Future work includes a thorough evaluation of the methods on different types of learning tasks such as regression, and survival analysis (however, preliminary results have shown that they are equivalently efficient and effective).

For a practitioner, based on the results on our methods we offer the following suggestions: first, to forgo the use of the computationally expensive nested cross-validation. Instead, we suggest the use of BBC-CV for small sample sizes (e.g., less than 100 samples). BBCD-CV could also be used in these cases to reduce the number of trained models (which may be negligible for such small sample sizes) but it may select a slightly sub-optimal configuration. For larger sample sizes, we advocate the use BBCD-CV that is computationally more efficient and maintains all benefits of BBC-CV. We also suggest using as many repeats with different partitions to folds as computational time allows, particularly for small sample sizes, as they reduce the widths of the confidence intervals and lead to a better selection of the optimal configuration. Finally, we'd like to note that the experimental results presented, are corroborated by results obtained with the application of the protocol to specific domains such as the prediction of protein properties based on their aminoacid sequence (Orfanoudaki et al. 2017), chemical properties of nanomaterials (Borboudakis et al. 2017), classification of voice pathology (Simantiraki et al. 2017), and prediction of suicides based on structured as well as textual data (Adamou et al. 2018a, b).

Acknowledgements IT, EG and GB received funding from the European Research Council under the European Unions Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement No. 617393. EG also received funding from the Toshiba project: "Feasibility study towards the Next Generation of statistical Text to Speech Synthesis System" and from Gnosis Data Analysis PC, Greece. We would like to thank Michalis Tsagris, Pavlos Charoniktakis, and Damjan Krstajic for constructive feedback.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

Adamou, M., Antoniou, G., Greasidou, E., Lagani, V., Charonyktakis, P., Tsamardinos, I., & Doyle, M. Towards automatic risk assessment to support suicide prevention. Crisis (to appear)

Adamou, M., Antoniou, G., Greasidou, E., Lagani, V., Charonyktakis, P., Tsamardinos, I., & Doyle, M. (2018). Mining free-text medical notes for suicide risk assessment. In: *Proceedings of the 10th hellenic conference on artificial intelligence*, SETN 2018, Patras, Greece, July 9-15, 2018. ACM.

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, *19*(6), 716–723.

Bernau, C., Augustin, T., & Boulesteix, A. L. (2013). Correcting the optimal resampling-based error rate by estimating the error rate of wrapper algorithms. *Biometrics*, *69*(3), 693–702.

Borboudakis, G., Stergiannakos, T., Frysali, M., Klontzas, E., Tsamardinos, I., & Froudakis, G. E. (2017). Chemically intuited, large-scale screening of MOFs by machine learning techniques. *npj Computational Materials*, *3*(1), 40.

Boulesteix, A. L., & Strobl, C. (2009). Optimal classifier selection and negative bias in error rate estimation: an empirical study on high-dimensional prediction. *BMC Medical Research Methodology*, *9*(1), 85.

Braga-Neto, U. M., & Dougherty, E. R. (2004). Is cross-validation valid for small-sample microarray classification? *Bioinformatics*, *20*(3), 374–380.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Cochran, W. G. (1950). The comparison of percentages in matched samples. *Biometrika*, *37*(3/4), 256–266.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap methods and their application*. Cambridge: Cambridge University Press.

Ding, Y., Tang, S., Liao, S. G., Jia, J., Oesterreich, S., Lin, Y., et al. (2014). Bias correction for selecting the minimal-error classifier from many machine learning models. *Bioinformatics*, *30*(22), 3152–3158.

Efron, B., & Tibshirani, R. J. (1993). *An introduction to the bootstrap*. Boca Raton: CRC Press.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874.

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, *33*(1), 1–22.

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

Garnett, R., Osborne, M. A., & Roberts, S. J. (2010). Bayesian optimization for sensor set selection. In: *Proceedings of the 9th ACM/IEEE international conference on information processing in sensor networks*, (pp. 209–219).

Greasidou, E. (2017). Bias correction of the cross-validation performance estimate and speed up of its execution time. Master's thesis, University of Crete, School of Sciences and Engineering, Computer Science Department.

Guyon, I., Alamdari, A.R.S.A., Dror, G., & Buhmann, J.M. (2006). Performance prediction challenge. In: *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, (pp. 1649–1656). IEEE.

Guyon, I., Bennett, K., Cawley, G., Escalante, H. J., Escalera, S., Ho, T. K., Macià, N., Ray, B., Saeed, M., Statnikov, A., & Viegas, E. (2015). Design of the 2015 ChaLearn AutoML Challenge. In: *Proceedings of IJCNN*

Guyon, I., Gunn, S., Ben-Hur, A., & Dror, G. (2004). Result analysis of the NIPS 2003 feature selection challenge. In: *Advances in neural information processing systems*, (pp. 545–552).

Harrell, F. E., Califf, R. M., Pryor, D. B., Lee, K. L., & Rosati, R. A. (1982). Evaluating the yield of medical tests. *Journal of the American medical association*, *247*(18), 2543–2546.

Iizuka, N., Oka, M., Yamada-Okabe, H., Nishida, M., Maeda, Y., Mori, N., et al. (2003). Oligonucleotide microarray for prediction of early intrahepatic recurrence of hepatocellular carcinoma after curative resection. *Lancet*, *361*(9361), 923–929.

Jensen, D. D., & Cohen, P. R. (2000). Multiple comparisons in induction algorithms. *Machine Learning*, *38*(3), 309–338.

Jensen, J. L. W. V. (1906). Sur les fonctions convexes et les Inégalités entre les valeurs moyennes. *Acta mathematica*, *30*(1), 175–193.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Ijcai*, *14*, 1137–1145.

Krstajic, D., Buturovic, L. J., Leahy, D. E., & Thomas, S. (2014). Cross-validation pitfalls when selecting and assessing regression and classification models. *Journal of Cheminformatics*, *6*(1), 10.

Krueger, T., Panknin, D., & Braun, M. (2015). Fast cross-validation via sequential testing. *Journal of Machine Learning Research*, *16*, 1103–1155.

Lagani, V., Athineou, G., Farcomeni, A., Tsagris, M., & Tsamardinos, I., et al. (2017). Feature Selection with the R Package MXM: Discovering Statistically Equivalent Feature Subsets. Journal of Statistical Software **80**(i07).

Maron, O., & Moore, A. W. (1994). Hoeffding races: Accelerating model selection search for classification and function approximation. In: *Advances in neural information processing systems*, (pp. 59–59).

Nankervis, J. C. (2005). Computational algorithms for double bootstrap confidence intervals. *Computational Statistics & Ddata Analysis*, *49*(2), 461–475.

Orfanoudaki, G., Markaki, M., Chatzi, K., Tsamardinos, I., & Economou, A. (2017). MatureP: Prediction of secreted proteins with exclusive information from their mature regions. *Scientific Reports*, *7*(1), 3263.

Salzberg, S. L. (1997). On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, *1*(3), 317–328.

Schwarz, G., et al. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*(2), 461–464.

Seibold, H., Bernau, C., Boulesteix, A.L., & De Bin, R. (2017). On the choice and influence of the number of boosting steps for high-dimensional linear cox-models. *Computational Statistics*. https://doi.org/10.1007/s00180-017-0773-8.

Simantiraki, O., Charonyktakis, P., Pampouchidou, A., Tsiknakis, M., & Cooke, M. (2017). Glottal source features for automatic speech-based depression assessment. *Proceedings of Interspeech*, *2017*, 2700–2704.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In: *Advances in neural information processing systems*, (pp. 2951–2959).

Statnikov, A., Aliferis, C. F., Tsamardinos, I., Hardin, D., & Levy, S. (2005). A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis. *Bioinformatics*, *21*(5), 631–643.

Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, *58*(1), 267–288.

Tibshirani, R. J., & Tibshirani, R. (2009). A bias correction for the minimum error rate in cross-validation. *The Annals of Applied Statistics*, *3*(2), 822–829.

Tsamardinos, I., Rakhshani, A., & Lagani, V. (2015). Performance-estimation properties of cross-validation-based protocols with simultaneous hyper-parameter optimization. *International Journal on Artificial Intelligence Tools*, *24*(05), 1540,023.

Varma, S., & Simon, R. (2006). Bias in error estimation when using cross-validation for model selection. *BMC Bioinformatics*, *7*(1), 91.

Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. Burlington: Morgan Kaufmann.

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). *Data mining: Practical machine learning tools and techniques*. Burlington: Morgan Kaufmann.

Yousefi, M. R., Hua, J., & Dougherty, E. R. (2011). Multiple-rule bias in the comparison of classification rules. *Bioinformatics*, *27*(12), 1675–1683.

Zheng, A. X., & Bilenko, M. (2013). Lazy paired hyper-parameter tuning. In: *IJCAI*