STATISTICAL DATA CLONING FOR MACHINE LEARNING

RESEARCH THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCE

GREGORY SHAKHNAROVICH

SUBMITTED TO THE SENATE OF THE TECHNION - ISRAEL INSTITUTE OF TECHNOLOGY

SIVAN, 5761   HAIFA   MAY, 2001

THE RESEARCH THESIS WAS DONE UNDER THE SUPERVISION OF DR. RAN EL-YANIV AND PROF. YORAM BARAM IN THE FACULTY OF COMPUTER SCIENCE

ACKNOWLEDGMENTS

I am grateful to my advisors Ran El-Yaniv and Yoram Baram. They patiently guided me throughout the work, and taught me much about research. I would like to thank Karen Livescu for her help and support. The pleasant company and the helpful feedback of my officemates Avi and Leonid was invaluable during many hours of work on this thesis. Comments of the anonymous reviewers of our paper at ICML-2001 were very helpful. Finally, I thank my mother for her everlasting love, encouragement and support.

THE GENEROUS FINANCIAL HELP OF THE TECHNION IS GRATEFULLY ACKNOWLEDGED

Contents

List of Tables

List of Figures

List of Algorithms

Abstract

This work is concerned with the estimation of a classifier's accuracy. We first review some existing methods for error estimation, focusing on cross-validation and bootstrap, and motivate the use of kernel-based smoothing for small sample size. We use the term *data cloning* to refer to the process of (re)sampling the data via kernel-based smoothed bootstrap. A number of novel estimators based on cloning is presented. Finally, we extend our estimators to to allow cloning of complex real-life data sets, in which a data point may include continuous, bounded, integer and nominal attributes. This allows for better classifier evaluation over heterogeneous real data repositories with limited amount of data, such as the UCI repository. We use the root mean squared error (RMSE) as a measure of estimators quality and support this choice with a probabilistic argument. Using this measure, we report on a set of 28 experiments in which the new cloning methods outperform cross-validation as well as the .632+ bootstrap, which, according to Efron and Tibshirani [13], is the estimator of choice. Although the proposed estimators require more computational effort than the established ones, the increased time complexity is within a constant factor of that of the relevant traditional estimators. Based on the motivation and the empirical results, we suggest that the cloning-based .632+ estimator is superior to the other estimators, and note bootstrapped cross-validation as the second choice.

Notation

$\mathcal{A}(X^{(n)})$	a classifier produced by the learning algorithm $\mathcal{A}$ given the training set $X^{(n)}$
$\mathbb{A}$	a one-dimensional nominal data domain
$\mathcal{D}$	general domain with continuous, discrete and nominal attributes
Err	the true generalization error of a classifier trained on a given data set, $E_{F(\mathbf{x}_0)}[Q(\mathbf{x}_0, X^{(n)})]$
$Err_{CV\times k}$	k-fold cross-validation estimator of Err, $\frac{1}{k}\sum_{j=1}^{k} Q(\mathbf{S}^j, X^{(n)}\backslash\mathbf{S}^j)$
Err_{BS}	ordinary bootstrap estimator of Err, $\frac{1}{B}\sum_{b=1}^{B} Q(X^{(n)}, {X^{(n)}}^*_b)$
$Err^{(1)}_{BS}$	leave-one-out bootstrap estimator of Err, $\frac{1}{n}\sum_{i=1}^{n}\frac{1}{B}\sum_{b=1}^{B} Q(\mathbf{x}_i, {X^{(n)}_{(i)}}^*_b)$
$Err_{.632}$	.632 bootstrap estimator of Err, $.632Err^{(1)}_{BS} + .368\overline{\mathrm{err}}$
$Err_{.632+}$	.632+ bootstrap estimator of Err
$Err_{.632+*}$	cloning-based .632+
$Err_{BSCV\times k}$	bootstrapped k-fold cross-validation estimator of Err
$Err_{RCV\times k}$	repeated k-fold cross-validation estimator of Err
$\overline{\mathrm{err}}$	empirical error of a classifier on a given training set, $Q(X^{(n)}, X^{(n)})$

F	a probability distribution function
f	a probability density function (PDF) or probability mass function
$\hat{F}_n$	the empirical probability distribution estimated on n data points
$\hat{F}_{(i)}$	the empirical probability distribution on $X^{(n)} \backslash \{\mathbf{x}_i\}$
$\hat{F}_{h,n}$	an estimate of F constructed from $X^{(n)}$ using bandwidth h
$\hat{f}$	estimate of the PDF f
h_j	a bandwidth used in kernel density estimation
K	kernel function
$Q\left(\mathbf{x}, X^{(n)}\right)$	error function: for a point $\langle \mathbf{x}, y \rangle$ 0 if $\mathcal{A}(X^{(n)})(\mathbf{x}) = y$ and 1 otherwise
$Q\left(W^{(m)}, X^{(n)}\right)$	the average test error $\frac{1}{m} \sum_{i=1}^{m} Q(\mathbf{w}_i, X^{(n)})$
T	cardinality of a finite data domain
$T_{train}(n)$	time complexity of training a classifier on $X^{(n)}$
T_{test}	time complexity of classifying a single data point
$X^{(n)}$	a sample of n points $\langle \mathbf{x}_1, \ldots, \mathbf{x}_n \rangle$
$X^{(n)^*}$	a bootstrap replication of $X^{(n)}$
$\mathbf{x}$	a d-dimensional vector $[x_1, \ldots, x_d]^T$
x_{ij}	the j-th element of $\mathbf{x}$
$\langle \mathbf{x}, y \rangle$	a labeled vector

α	statistical significance of a hypothesis test
$\hat{\gamma}$	the no-information error rate for a data set and classifier
$\widehat{R}$	relative overfitting rate of a classifier
μ_n	expected true generalization error of a classifier trained on arbitrary $X^{(n)}$

1 Introduction

A common approach to the evaluation and comparison of inductive learning algorithms is to test them on data sets from various "real-life" settings. We shall refer to such data as "real", in contrast to data generated by artificial methods. While many theoretical conclusions can be drawn from an algorithm's performance on synthetic data, good performance on real data sets is believed to be evidence for an algorithm's practical plausibility. Public repositories of real data have appeared in recent years; one of the most known and widely used of these is the University of California at Irvine (UCI) repository [3]. Unfortunately, real data is often expensive to label, sometimes tedious to collect, and in some domains is available only in limited amounts. The UCI repository, which is one of the largest, contains about 100 data sets, just a few compared to the huge number of studies using these data sets. Such over-usage of data can lead to a compromise in the significance of the obtained results [29].

This problem would be greatly alleviated if one could generate arbitrarily many "new" data samples, distributed according to the same probability law as the real data set at hand. This would provide a valuable tool for the machine learning research community, since algorithm performance results on real data would be obtained with greater statistical significance. Since the true distribution of the data is unknown, this task is impossible. However, one can "clone" the available data and generate a family of data sets that are different from but similar to the original one (in a statistical sense to be defined) and evaluate the performance of an algorithm on these data sets. The bootstrap, introduced in

[10], provides a means for such data cloning.

1.1 The learning paradigm

We consider the following supervised learning paradigm. A data set $X^{(n)}$ consists of n labeled pairs $\langle \mathbf{x}_1, y_1 \rangle, \ldots, \langle \mathbf{x}_n, y_n \rangle$, where the points $\mathbf{x}_i$ are generated i.i.d. in a d-dimensional data space $\mathcal{D}$, according to an unknown probability distribution F, and $y = 0, 1, \ldots$ are *class labels*. For brevity, we shall usually use the notation $\mathbf{x}$ to refer to $\langle \mathbf{x}, y \rangle$. Given a data sample $X^{(n)}$, the *learning algorithm* $\mathcal{A}$ produces a *hypothesis* $C_{\mathcal{A}}(X^{(n)}) = \mathcal{A}(X^{(n)})$ from a certain concept class. This hypothesis, called a *classifier*, is a function that assigns a class label to each $\mathbf{x}$.

Throughout the paper, we consider the zero-one loss function. Fixing the learning algorithm $\mathcal{A}$ and following standard notation (e.g. [25]), we denote the error function (the penalty for the classification decision made by $\mathcal{A}(X^{(n)})$ on the *test point* $\mathbf{x}$) by

$$Q\left(\mathbf{x}, X^{(n)}\right) = Q\left(\mathbf{x}, X^{(n)}, \mathcal{A}(X^{(n)})\right).$$

We assume that this penalty is 0 for correct classification and 1 otherwise. For further simplicity, we shall use a similar notation for the average error of the classifier trained on

$X^{(n)}$, over a *test set* $W^{(m)} = (\mathbf{w}_1, \ldots, \mathbf{w}_m)$:

$$Q\left(W^{(m)}, X^{(n)}\right) = \frac{1}{m}\sum_{i=1}^{m} Q(\mathbf{w}_i, X^{(n)})$$

1.2 Generalization error of a classifier

We are interested in measuring the accuracy of a classifier by its ability to generalize, that is, to assign the correct class label to a previously unseen data point. The *true error Err* of a classifier trained on $X^{(n)}$ is

$$\mathit{Err} = \mathit{Err}(X^{(n)}, F) = \mathrm{E}_{F(\mathbf{x}_0)}[Q(\mathbf{x}_0, X^{(n)})]. \tag{1}$$

Here, the assumption is that a random test point $\langle \mathbf{x}_0, y_0 \rangle$ is distributed according to the same F as the training set. *Err* is sometimes called a *conditional true error*, since it depends on the random variable $X^{(n)}$. For a fixed training set, *Err* can be written as

$$\mathit{Err}(X^{(n)}, F) = \int_{\mathcal{D}} Q(\mathbf{x}_0, X^{(n)})\, dF(\mathbf{x}_0) \tag{2}$$

Note that this is different from the *expected true error* for a training set of size n:

$$\mu_n(F) = \mathrm{E}_{F(X^{(n)})}[\mathit{Err}(X^{(n)}, F)]. \tag{3}$$

In practice, given a fixed training set of size n, it is not clear how to estimate the expectation over all possible training sets of size n . We therefore focus in this paper on the task of estimating Err.

1.3 Bias and variance in error estimation

Being based on $X^{(n)}$, which is a random variable, any estimator $\widehat{Err}$ of the true error Err is a random variable itself. Neither its *bias* $\mathrm{E}_{F(X^{(n)})}[\widehat{Err} - Err]$ nor its variance each by itself can serve as a good measure of the estimator's quality. In a single experiment, the error of an unbiased estimator with a large variance can to be greater than that of a mildly biased estimator with a small variance.

We follow the practice of using the squared error $(\widehat{Err} - Err)^2$ as a measure of estimator quality [13], and motivate it as follows. Consider the task of obtaining a single estimate of a classifier's accuracy on a given data set. Assuming that the objective in choosing an estimator is to minimize the probability of a large absolute (or squared) deviation from the true error in this single estimate. Then by Markov's inequality,

$$\Pr\left((Err - \widehat{Err})^2 \geq \epsilon\right) \leq \mathrm{E}\left[(Err - \widehat{Err})^2\right]/\epsilon. \tag{4}$$

The sample mean of $(Err - \widehat{Err})^2$ (MSE) is an estimate of $\mathrm{E}\left[\left(Err - \widehat{Err}\right)^2\right]$ and can be used to estimate the bound in (4). We use the root mean squared error $(RMSE)$ to bring the value to linear scale.

1.4 Goals of this work

In this paper we propose a number of novel bootstrap estimators for a classifier error. The new estimators are based on a smoothed version of the bootstrap method where we use a density estimation technique to resample from the available data with some additional noise. Smoothed bootstrap estimators are already known; as far as we know, however, they have never been employed for classifier error estimation. The new estimators improve on existing ones by variance reduction. We present an extensive set of simulation results on synthetic data that show the consistent advantage of our estimators over the best known estimators to date. In particular, our estimators outperform the .632+ of [13] and the standard cross-validation technique. We repeat the exact experiments performed by Efron and Tibshirani and test our methods with respect to various classifiers, including Support Vector Machines.

Second, we propose algorithms for computing smoothed bootstrap samples on complex real data sets, such as those in the UCI repository, that may include continuous, bounded, integer and nominal attributes. We refer to this method as *statistical data cloning.* Finally, we test our cloning algorithms on a few data sets from UCI, including the two data sets that were used in [13], and show again that our data cloning techniques exhibit performance superior to that of the traditional estimators. Throughout the paper we use the term "cloning" for all smoothed bootstrap sampling techniques.

2 On some known methods for estimation of *Err*

2.1 Empirical error

A classifier can be tested on the same data it was trained on. The *empirical* (or *resubstitution*) *error*

$$\overline{\mathrm{err}} = Q(X^{(n)}, X^{(n)}) \tag{5}$$

is typically over-optimistic (i.e., it underestimates the true error) and has a negative bias that is especially large for learning algorithms with high overfitting. In an extreme case, like that of the nearest neighbor rule, $\overline{\mathrm{err}}$ can be zero even when *Err* is 1/2.

Let us denote the computational complexity of training the classifier on $X^{(n)}$ by $T_{train}(n)$, and the complexity of testing it on a single point by T_{test}. Then the complexity of $\overline{\mathrm{err}}$ is $O(T_{train}(n) + nT_{test})$. It is worth to notice that the dominant factor between $T_{train}(n)$ and T_{test} depends on the learning algorithm. For k-nearest neighbors $T_{train}(n)$ is zero for all n but T_{test} is high, while for a perceptron $T_{train}(n)$ is dominant while $T_{test} = O(1)$.

2.2 Cross-validation and holdout

To avoid underestimating the error due to resubstitution, in the *holdout* the data is split randomly into two parts, and the classifier is trained on one and tested on the other. As has been observed in [4], holdout reduces the amount of data available for training. There is also an issue of statistical dependence between the two subsets.

A way to overcome this by using both subsets for training, is generalized in the *cross-validation* methodology. In the k-fold cross-validation (CV), the data set is partitioned into k mutually disjoint subsets called *folds*. For each fold $\mathbf{S}^j, j = 1, \ldots, k$, the classifier is trained on all of the data except $\mathbf{S}^j$ and tested on $\mathbf{S}^j$. The resulting estimator is computed as the average of the error rates over the k folds:

$$Err_{CV \times k} = \frac{1}{k} \sum_{j=1}^{k} Q(\mathbf{S}^j, X^{(n)} \backslash \mathbf{S}^j). \tag{6}$$

The extreme case of the k-fold CV-based estimator is the *leave-one-out* estimator, $Err_{CV \times n}$. The leave-one-out CV is known to produce an almost unbiased estimate of *Err*, since for every fold the classifier is trained on almost the complete $X^{(n)}$; however, it often suffers from high variance due to the learning algorithm's instability under small perturbations in the data [23]. A possible approach to reducing the variance of CV is to perform k-fold CV in a number of trials, when the random partition of data into k folds is independent in each trial, and average the result. We denote the described estimator by $Err_{RCV \times k}$.

The computational complexity of a k-fold CV is $O(k[T_{train}(n - n/k) + kT_{test}])$. Thus for classifiers in which training is more expensive than generalization, the leave-one-out has an asymptotic complexity $O(nT_{train}(n))$.

2.3 Bootstrap and its use for error estimation

The bootstrap method is based on the following idea. A (nonparametric) maximum likelihood estimator of a statistic $\theta(X^{(n)})$ is given by the expectation of θ with respect to the empirical distribution $\hat{F}_n$, which gives a probability mass of $1/n$ to each sample $\mathbf{x}_i$ in $X^{(n)}$ [12]. Usually no analytical expression for this expectation exists, as it is in case of $\theta(X^{(n)}) = \mathit{Err}$. Combinatorial explosion does not allow the enumerative computation of this expectation: there are approximately $\binom{2n-1}{n-1}$ resampling of $X^{(n)}$, different as sets. However, a Monte-Carlo algorithm allows for numerical evaluation of this expectation by drawing B samples of size n from $\hat{F}_n$. These samples ${X^{(n)}}^*_1, \ldots, {X^{(n)}}^*_B$ are called *bootstrap samples*, and the value $\theta({X^{(n)}}^*_b)$ is called a *bootstrap replication* of θ. A bootstrap estimate of θ is then computed by averaging over the bootstrap replications. The averaging done by bootstrap typically reduces the discontinuities in the statistic, and thus lowers its variability.

2.3.1 Ordinary bootstrap estimator

The ordinary, or "naive", bootstrap estimate of *Err*, constructed with B replications, is given by

$$\mathit{Err}_{BS} = \frac{1}{B} \sum_{b=1}^{B} Q(X^{(n)}, {X^{(n)}}^*_b), \tag{7}$$

that is, in each of the B iterations the classifier is trained on a BS sample $X_b^{(n)*}$ and tested on the original data set. The final estimate is the average over B estimates. Note that a test point $\mathbf{x}_i$ is included in the training set $X^{(n)*}_b$ with probability $1-(1-1/n)^n$, which is approximately .632 for large n. Due to this expected partial resubstitution, one should expect Err_{BS} to be biased downwards.

If we assume that sampling a single point from $X^{(n)}$ costs $O(1)$, then the computational complexity of Err_{BS} is $O(B[T_{train}(n) + nT_{test}])$.

2.3.2 Leave-one-out bootstrap

Leave-one-out bootstrap [13] is a "smoothed" version of $Err_{CV\times n}$. To compute it, one draws the bootstrap samples from the empirical distribution of the original sample with the i-th point removed, $X^{(n)}_{(i)}$. This distribution $\hat{F}_{(i)}$ assigns probability $1/(n-1)$ to all $\mathbf{x}_j$, $j \neq i$. Then,

$$Err^{(1)}_{BS} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{B}\sum_{b=1}^{B} Q(\mathbf{x}_i, X^{(n)*}_{(i)\ b}). \tag{8}$$

This smoothing reduces the variance. However, in computation of a single term in (8), the training set has an expected $.632n$ distinct data points, as opposed to $Err_{CV\times n}$, which is based on classifiers trained on $n-1$ points. Since, for many classifiers, Err tends to decrease as the size of the training set grows, this fact implies an upward bias.

The complexity of this estimator is $O(B[T_{train}(n-1) + T_{test}])$.

2.3.3 Hybrid bootstrap and .632+

The error estimate of the general form of a *hybrid bootstrap error estimator* is given by

$$\widehat{Err}^{\lambda} = \lambda Err^{(1)}_{BS} + (1 - \lambda)\overline{\mathrm{err}},$$

and a mixing parameter λ is sought that minimizes the bias. Many authors (e.g. [25], [11], [5]) report "substantial empirical evidence" favoring $\lambda = 0.632$. Efron also gives a heuristic motivation for this number: a bootstrap sample of size n is expected to be supported by approximately $.632n$ original data points. This choice of λ gives rise to the .632 bootstrap estimator given by $Err_{.632} = .632 Err^{(1)}_{BS} + .368\overline{\mathrm{err}}$. The intuition behind $Err_{.632}$ suggests compensation for the downward bias of $\overline{\mathrm{err}}$ via the upward bias of $Err^{(1)}_{BS}$. However, for a classifier with high overfitting (such as 1-NN) $\overline{\mathrm{err}} \equiv 0$, and $Err_{.632}$ is downward biased itself.

The computation of $Err_{.632}$ does not require any applications of the classifier beyond the computation of $Err^{(1)}_{BS}$ and $\overline{\mathrm{err}}$.

2.3.4 The .632+ estimator

$Err_{.632+}$, proposed by Efron and Tibshirani in [13], is a sophisticated estimator that attempts to estimate the amount of overfitting and adjust λ accordingly. Let F_{ind} be the probability distribution on points $\langle \mathbf{x}, y\rangle$ with the same marginals over $\mathbf{x}$ and y as the true

F, but with the label y independent of $\mathbf{x}$. The authors define the "no-information" error rate γ, which is the error rate of the classifier when the data conveys no information for the classifier ($\mathbf{x}$'s and y's independent):

$$\gamma = \mathrm{E}_{F_{ind}(\mathbf{x})}[Q(\mathbf{x}, X^{(n)})],$$

and obtain its estimate $\hat{\gamma}$ by averaging the error rate of the classifier trained on $X^{(n)}$ over all possible permutations of the data and the labels:

$$\hat{\gamma} = \sum_{i=1}^{n}\sum_{j=1}^{n} Q(\langle \mathbf{x}_i, y_i \rangle, X^{(n)})/n^2 \quad .$$

Intuitively, $\hat{\gamma} - \overline{\mathrm{err}}$ characterizes the maximum amount of the overfitting of the classifier, for given classification problem and learning algorithm. For example, for a binary classification problem, with class labels 0 and 1, let $\hat{p}_1$ be the observed proportions of points labeled 1 in $X^{(n)}$, and $\hat{q}_1$ be the proportion of points in $X^{(n)}$ that are assigned 1 by a classifier trained on $X^{(n)}$. Then,

$$\hat{\gamma} = \hat{p}_1(1 - \hat{q}_1) + (1 - \hat{p}_1)\hat{q}_1.$$

In particular, if $\hat{p_1}$ is 1/2, then $\hat{\gamma} = 1/2$ for any classifier.

From the no-information error rate estimate $\hat{\gamma}$, the estimated *relative overfitting rate*

$\widehat{R}$ is derived:

$$\widehat{R} = \frac{Err^{(1)}_{BS} - \overline{\text{err}}}{\hat{\gamma} - \overline{\text{err}}}.$$

(The values of $Err^{(1)}_{BS}$ and $\widehat{R}$ are corrected if necessary, to ensure that $\widehat{R}$ falls within $[0, 1]$.) For a classifier with no overfitting $\widehat{R} = 0$, while the highest possible overfitting corresponds to $\widehat{R} = 1$.

Finally the .632+ estimator is obtained as

$$Err_{.632+} = Err_{.632} + (Err^{(1)}_{BS} - \overline{\text{err}})\frac{.368 \cdot .632 \cdot \widehat{R}}{1 - .368\widehat{R}}.$$

The computation of $Err_{.632+}$ involves obtaining the values of $Err_{.632}$ and $\overline{\text{err}}$, and computing the $\hat{\gamma}$; the total complexity is

$$O\left(B[T_{train}(n-1) + T_{test}] + O(T_{train}(n) + T_{test}) + O(n^2 T_{test})\right).$$

For classifiers with training more expensive than testing, and for values of n larger than B (which is fixed, typically 50 or 100), this can be less than the complexity of CV methods.

3 Related work

3.1 Estimating classifier accuracy

A great deal of work exists on error estimation for both regression (with the prediction error usually measured in terms of squared loss) and classification (with various loss functions). A majority of the researchers in Machine Learning measure a classifiers performance by its misclassification rate, which is equivavlent to a symmetric, zero-one loss function [24]. An up-to-date survey of some of the most important results in this field can be found in Chapter 9 of [9].

It has been noted (see discussion on the “No Free Lunch Theorem” in [9]) that there is no single estimator that could be theoretically proven or even empirically shown to be optimal for any given data domain and classifier. For many classifiers, especially the more complex ones such as SVMs, it is difficult to describe the error function in terms of the data distribution, and to predict the behavior of a certain estimator for that classifier, on a particular problem domain. While some bounds on the discrepancy of estimators can be established, the choice of an estimator largely relies on empirical evidence of its performance, that is, simulation studies, in which accuracies of different estimators are compared for classifiers at hand, on synthetic and real data.

Different authors seem to use different measures to quantify and compare accuracy of estimators. In the statistical literature it is common to use mean square error (MSE) over a number of trials. In the machine learning literature, often bias and variance of an estimator are referred to as the properties that define its accuracy. This parameters are also estimated by taking the results from a number of trials.

3.2 Studies of different accuracy estimators

Vapnik in [35] provides a bound on the discrepancy of $\overline{\mathrm{err}}$, which is expressed in terms of the VC-dimension of the concept class. Cross-validation has been studied by many authors. Kearns and Ron prove in [22] that under certain conditions on the algorithmic stability of the learning algorithm, there is an upper bound on the absolute deviation of leave-one-out estimator from the true error, and this bound is of the same order of magnitude as the one for $\overline{\mathrm{err}}$. Holdout is studied in [4], that shows that k-fold CV for $k > 2$ is more accurate than a single holdout estimate with test subset of size n/k. Based on this result we decided not to include holdout in our empirical study.

According to reports in the ML community (e.g., [2]), cross-validation is the most widely used method of accuracy estimation for this task. Bootstrap methods are less popular. From empirical results bootstrap estimators is known to have higher biases than CV but lower variances, and consequently show better results in terms of squared error when a large number of trials is performed ([11],[31]). Kohavi [23] shows that for 10 UCI data sets, with respect to zero-one loss, CV-based estimators work better due to the large bias of the .632 bootstrap. However, the main criterion used in the study is the bias of the estimator, and no

RMSE perfomance is reported. Dietterich in [8] considers a few statistical tests for comparing classifiers, and while he concludes that all the discussed tests have some shortcomings, he recommends the 5x2CV (2-fold CV repeated 5 times) and the McNemar's test. He does not address the task of estimating the accuracy of a single classifier, however. More recently, Efron and Tibshirani [13] obtained the best performance with their .632+ estimator for a number of data sets, including some of those used by Kohavi. Therefore .632+ appears to be the state-of-the-art BS-based estimator, in terms of RMSE performance.

There have been previous efforts at classification using density estimation for complex, real-life data involving non-numeric features (e.g. [19]), but with no clear means of sampling from the estimated density. We also do not know of any published system that would implement all the currently established techniques in density estimation, such as whitening of the data, and allow the user to sample a value from the estimated probability function without a need to "tune" the system.

3.3 Bootstrap methods

Bootstrap was introduced by Efron in [10], and since has become a topic of active research in the statistics community. It has been successful in many cases of statistical estimation ([31],[5]). A comprehensive description of the bootstrap method is given in [14].

Investigation of the properties of smoothed bootstrap is still an ongoing research effort in statistics. A brief summary follows.

3.4 Smoothed bootstrap

Constructing a bootstrap replication of the data by sampling with replacement is equivalent to randomly drawing samples from the empirical probability distribution $\hat{F}_n$ [5]. Instead, one may estimate the *PDF* of the data, using non-parametric density estimation, and use the estimated density to draw samples. This is the underlying idea of *smoothed bootstrap* [33]. In this work we focus on kernel density estimation method, in which a smoothing value h must be chosen to generate the kernel-based distribution $\hat{F}_{h,n}$.

No general conditions have been established under which the smoothed bootstrap improves on the results of the ordinary bootstrap. The accepted intuition is that if the true distribution F is smooth, it is reasonable to prefer the smooth $\hat{F}_{h,n}$ to the non-smooth $\hat{F}_n$. More importantly, the effect of smoothing depends on the statistic under consideration. This dependence has been explored by several authors ([33],[31],[18],[15]). Generally, smoothing is considered to be beneficial when the estimated statistic depends on local behavior of F, for instance, if a small change in F significantly changes the value of the statistic. Our intuition is that the generalization error of many classifiers behaves in this way, as a small change in F which produces the training and the test sets may change the decision boundary, and as a result the error on a test point may change from 0 to 1 or vice versa.

Since both the empirical distribution $\hat{F}_n$ and the kernel-based estimate $\hat{F}_{h,n}$ are consistent estimators of the true F, the difference between the smoothed BS and the non-smoothed one disappears as $n \to \infty$. The difference can be significant in the case of small sample size, with which we are concerned in this work.

Examples of a statistics for which smoothing the bootstrap has been shown to significantly improve the estimation accuracy, include sample quantiles [31] and sample correlation coefficient [33]. Recent reports suggest that smoothing is beneficial for computation of confidence intervals for continuous [15] and discrete [17] data. One of the main obstacles in applying smoothed bootstrap methods to real problems with complex data domains has been the complexity of implementation of the density estimation and the resampling for such data. In this work we attempt to alleviate the task, and discuss both technical and conceptual problems that arise on the way. In the remainder of this section we describe the basic mechanism for resampling from the kernel-based density estimate for the simplest case, in which all the attributes are continuous and unbounded. In Section 7 we extend this mechanism for the general case of arbitrary data domains.

4 Smoothed bootstrap for accuracy estimation

4.1 Kernel-based nonparametric density estimation

We use kernel-based density estimation (Parzen windows), as described, for example, in [30]. The estimated *PDF* of a point $\mathbf{x}$, based on the sample $X^{(n)}$, is

$$\hat{f}(\mathbf{x}) = \frac{1}{n|\mathbf{H}|} \sum_{i=1}^{n} K\left(\mathbf{H}^{-1}\left(\mathbf{x} - \mathbf{x}_i\right)\right), \tag{9}$$

where the kernel K is a probability density function, and $\mathbf{H}$ is a matrix which essentially describes the covariance structure of K. It has been suggested in the literature [16] that if the d-dimensional data set is *whitened*, that is, transformed to have a unit covariance matrix, one can use the product kernel, which is a product of d one-dimensional kernels:

$$\hat{f}(\mathbf{x}) \; = \; \frac{1}{n} \prod_{j=1}^{d} \frac{1}{h_j} \sum_{i=1}^{n} \prod_{j=1}^{d} K\left(\frac{x_j - x_{ij}}{h_j}\right) \quad . \tag{10}$$

To apply a kernel method, two parameters must be set: the *kernel function* K and its *bandwidth* h_j. The choice of bandwidth is critical to the success of the method, and a large number of methods for automatic bandwidth selection exist [21]. Intuitively, the optimal bandwidth should enforce some smoothness conditions on $\hat{f}$, which can be expressed as bounds on $\int \hat{f}''$. Our algorithms use a direct plug-in method, based on kernel-based estimation of the derivatives of f. The algorithm is described in detail in [36], and here we provide a brief explanation.

It can be shown [36], that the asymptotic mean integrated squared error (MISE) of density estimate is minimized for the value of bandwidth

$$h_{AMISE} = \left[\frac{R(K)}{\mu_2(K)^2 \psi_4 n} \right], \tag{11}$$

where for any r,

$$\psi_r = \int f^{(r)}(x) f(x) dx, \tag{12}$$

and for a function K

$$R(K) = \int [K(x)]^2 dx. \tag{13}$$

Since f is unknown, so is also the functional ψ_4. We have to estimate it by a kernel estimate $\hat{\psi}_4(g)$ where g is a bandwidth. Now, we face the problem of choosing g. The optimal g, in turn, is given in terms of ψ_6. This goes on, as the optimal bandwidth for estimation of ψ_r depends on ψ_{r+2}. To stop this dependence, one can estimate ψ_r, for some r, using *normal scale*. The data is assumed to be distributed normally, with variance σ_x^2. Then, the variance is estimated using the standard sample variance estimate $\hat{\sigma}_x^2$, and we have

$$\hat{\psi}_r = \frac{(-1)^{r/2} r!}{(2\sigma)^{r+1} (r/2)! \pi^{1/2}}$$

From here, we estimate g_{r-2}, g_{r-4} etc., and finally estimate the h.

The choice of kernel is known to be much less important for the quality of the density estimate ([32]). The Epanechnikov kernel $K(x) = 3(1 - x^2)/4$, with support $|x| \leq 1$, has been shown to be the optimal kernel for density estimation: used with the optimal value of its bandwidth, it minimizes the mean integrated squared error (MISE) of the estimation [30]. This is only slightly better than many other kernels, and optimality criterion unrealistically assumes the ability of finding the optimal bandwidth. Also, it has not been proven that minimal MISE for density estimation corresponds to the best performance in smoothed bootstrap. Nevertheless, the Epanechnikov kernel allows for simple resampling procedure, which further motivated us to use it. The shape of 2-D product Epanechnikov kernel is shown in Figure 1(a).

4.2 Sampling from the estimated density

Kernel-based *PDF* estimation is a computationally expensive task. However, to sample from $\hat{f}$, one does not need to compute (9) explicitly. For arbitrary $\mathbf{x}$, the value of $\hat{f}(\mathbf{x})$ is the sum of the contributions of all of the sample points, each weighted by $1/n$. Therefore, the result of adding random noise $\mathbf{w}$ drawn from K to a uniformly chosen $\mathbf{x}_i$ will have density $\hat{f}$. This is formalized in Lemma 1 [5], given below with our proof.

Lemma 1. *The random variate generated by Alg. 1 has the PDF given by* (10)*.*

Proof. Let $p_x(\cdot)$ be the *PDF* of an $\mathbf{x} = \mathbf{x}_l + \mathbf{w_h}$ generated by the algorithm, and $p_{wh}(\cdot)$ the *PDF* of $\mathbf{w_h}$. Note that $\mathbf{w_h} = \mathbf{x} - \mathbf{x}_l$. Since the elements of $\mathbf{w}$ (and therefore those of $\mathbf{w_h}$)

Algorithm 1 Generation of $\mathbf{x}$ distributed with $\hat{f}$, a *PDF* estimate based on $X^{(n)}$, with kernel K and bandwidths $\mathbf{h}$.

```
Input: A sample $X^{(n)}$, set of bandwidths $h_1, \dots, h_d$, and a kernel $K$.
Output: $\mathbf{x} \sim \hat{f}$ as in (10)
  Draw $l$ uniformly from $\{1, \dots, n\}$
  Generate a random vector $\mathbf{w} = [w_1, \dots, w_d]^T$ with elements $\texttt{iid} w_i \sim K$.
  Let $\mathbf{w_h} \leftarrow [h_1 w_1, \dots, h_d w_d]^T$.
  Return $\mathbf{x} \leftarrow \mathbf{x}_l + \mathbf{w_h}$.
```

are independent, we have

$$p_{wh}(\mathbf{w_h}) = \prod_{j=1}^{d} K\left(x_j - x_{lj}\right),$$

and using the fact that for any *PDF* p_t, $p_{at}(at) = \frac{1}{|a|} p_t(t)$ ([27]),

$$p_{wh}(\mathbf{w_h}) = \prod_{j=1}^{d} \frac{1}{h_j} K\left(\frac{x_j - x_{lj}}{h_j}\right).$$

Since the algorithm assigns an equal probability mass of $\frac{1}{n}$ to all of the $\mathbf{x}_z, \quad z = 1, \dots, n$, one can write

$$\begin{aligned} p_x(\mathbf{x}) &= \sum_{z=1}^{n} \frac{1}{n} p_x\left(\mathbf{x} | l = z\right) = \frac{1}{n} \sum_{z=1}^{n} p_w(\mathbf{x} - \mathbf{x}_z) \\ &= \frac{1}{n} \sum_{z=1}^{n} \prod_{j=1}^{d} \frac{1}{h_j} K\left(\frac{x_j - x_{zj}}{h_j}\right) \\ &= \hat{f}(\mathbf{x}) \end{aligned}$$

$\square$

Using the Epanechnikov kernel, each component of $\mathbf{w}$ can be generated using the rejection method [7]. The expected number of trials of the rejection method for generating an element of $\mathbf{w}$, with K being the Epanechnikov kernel, is 3 [6].

Algorithm 2 Generation of $\mathbf{W} \sim K$ for Epanechnikov kernel K.

```
repeat
  Generate $W$ uniformly on $[-1, 1]$.
  Generate $U$ uniformly on $[0, 1]$.
until $U \le 1 - W^2$.
return $W$.
```

Lemma 2. *Algorithm 2 generates a random variate distributed by K, and has expected time complexity $O(1)$, namely, 3.*

Proof. The distribution of the returned value of W' is

$$F(x) = \Pr(W' \le x) = \Pr(W \le x | U \le 1 - W^2) = \frac{\Pr(W \le x, U \le 1 - W^2)}{\Pr(U \le 1 - W^2)}.$$

We compute the denominator:

$$\begin{aligned}\Pr(U \le 1 - W^2) &= \int_{-1}^{1} \Pr(U \le 1 - w^2 | w = W) dF_W(w) = \\ &= \frac{1}{2} \int_{-1}^{1} (1 - w^2) dw = 2/3,\end{aligned}$$

using the fact that U has a uniform distribution. The joint distribution in the nominator is computed in a similar way:

$$\Pr(W \leq x, U \leq 1 - W^2) = \frac{1}{2}\int_{-1}^{x}(1-w^2)dw = \frac{x}{2} - \frac{x^3}{6} = \frac{1}{3}.$$

Now, to compute the density of W' we take the derivative:

$$f(x) = \frac{3}{2}\left(\frac{1}{2} - \frac{x^2}{2}\right) = \frac{3}{4}(1-x^2) = K(x).$$

Now we prove the expected time complexity. Let us denote $U' = 1 - U$. For fixed U, the probability of rejection of the chosen W in a single iteration of the algorithm equals

$$\begin{aligned}\Pr(W^2 < U'|U) &= \Pr(0 < W < U'^{\frac{1}{2}}|U) + \Pr(-U'^{\frac{1}{2}} < W < 0|U)\\ &= 2\Pr(0 < W < U'^{\frac{1}{2}}|U).\end{aligned}$$

(the last equality follows from the symmetry of the uniform *PDF* of W). Then, the total probability to reject the uniformly chosen U is

$$\Pr(W^2 < U') = 2\int_0^1 \Pr(0 < W < U'^{\frac{1}{2}})du = \int_0^1 U^{\frac{1}{2}}du = \frac{2}{3}.$$

Then, each iteration step in Algorithm 2 is a Bernoulli trial with $\frac{2}{3}$ failure probability, and the expected number of trials for a success is thus 3. $\square$

Thus, the expected time complexity of Algorithm 1 is $O(1)$, and the computational cost of resampling n points in smoothed BS has the same order of magnitude as that in ordinary BS.

4.3 Whitening the data

Consider a sample (dataset) of n points, $X^{(n)} = \langle \mathbf{X}_1, \ldots, \mathbf{X}_n \rangle$, which are mutually *independent and identically distributed* (`iid`), drawn from probability distribution function $F(\mathbf{X})$ with *PDF* (or probability mass function in case of discrete feature space) $f(\mathbf{X})$; that is, each sample point (which is actually a d-dimensional vector) $\mathbf{X}_i = [x_{i1}, \ldots, x_{id}]^T$ is a value of the random variable $\mathbf{X} \sim f$. It may be the case, and indeed usually is with the real data, that we don't know what F or f is, but assume their existence. Let $\mathbf{M}$ be the expected (mean) value of $\mathbf{X}$, $\mathbf{M}(\mathbf{X}) = \mathrm{E}[\mathbf{X}]$, and Σ the *covariance matrix* of $\mathbf{X}$:

$$\mathbf{\Sigma}(\mathbf{X}) = \mathrm{E}[(\mathbf{X} - \mathbf{M})(\mathbf{X} - \mathbf{M})^T] = S - \mathbf{M}\mathbf{M}^T \quad ,$$

where $\mathbf{S} = \mathrm{E}[\mathbf{X}\mathbf{X}^T]$ is the *autocorrelation* matrix of $\mathbf{X}$. Since one cannot compute the $\mathbf{\Sigma}$ and related values without knowing the underlying *PDF*, it is customary to use their sample estimates. The *sample autocorrelation* matrix is defined by

$$\widehat{\mathbf{S}}(X^{(n)}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{X}_i \mathbf{X}_i^T, \tag{14}$$

the *sample mean*

$$\widehat{\mathbf{M}}(X^{(n)}) \;=\; \frac{1}{n}\sum_{i=1}^{n}\mathbf{X}_i, \tag{15}$$

(which are unbiased consistent estimates of $\mathbf{S}$ and $\mathbf{M}$, respectively), and the *sample covariance* matrix

$$\hat{\mathbf{\Sigma}}(X^{(n)}) \;=\; \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{X}_i - \widehat{\mathbf{M}})(\mathbf{X}_i - \widehat{\mathbf{M}}^T). \tag{16}$$

The division by $n-1$ rather than n eliminates the bias in the estimator. Generally, values of different features in the data have different scales, and different sample variances. The whitening transformation suggested in [16], is meant to make the covariance matrix of the transformed sample to equal $\mathbf{I}$. Let $\mathbf{\Lambda}$ be the diagonal matrix of the eigenvalues of $\hat{\mathbf{\Sigma}}(X^{(n)})$, and $\mathbf{\Phi}$ be the matrix whose columns are the eigenvectors of $\hat{\mathbf{\Sigma}}(X^{(n)})$, corresponding to $\mathbf{\Lambda}$. The whitening transformation is defined by

$$\mathbf{W} \;=\; (\mathbf{\Phi}\mathbf{\Lambda}^{-\frac{1}{2}})^T \tag{17}$$

It can be shown [16] that $\hat{\mathbf{\Sigma}}\left(\mathbf{W}X^{(n)}\right) \;=\; \mathbf{I}$. Subtracting the sample mean from the sample before the multiplication by $\mathbf{W}$, will ensure the mean of the transformed sample to be $\mathbf{0}$. The reverse transformation is done by multiplication by $\mathbf{\Phi}\mathbf{\Lambda}^{\frac{1}{2}}$ and further addition of $\widehat{\mathbf{M}}$.

5 New estimation schemes

It is evident from both theoretical and empirical results that variance reduction in an estimator of can be advantageous even if the bias somewhat increases. This is the essence of the so called "bias-variance dilemma" [9]. In order to reduce the variance, we consider two extensions of known estimators. Both are guided by the objective to smooth the discontinuities in the estimation of *Err* that may be caused by perturbations in data. We would like to reduce the effect that substitution of one point in a subset of $X^{(n)}$ has on the estimated accuracy of a classifier, to decrease the variability in the estimate, and eventually to reduce the probability of having a large deviation in a single estimation trial.

5.1 Using cloning in BS-based estimators

The first idea is to use cloning where normally an ordinary BS is used. Possible improvement in the expected squared error here is achieved by using a more sophisticated sampling method. Smoothed bootstrap is known [31] to outperform the ordinary BS for statistics that are sensitive to local properties of the underlying distribution. Our intuition is that this holds for *Err* in the case of many classifiers.

We denote BS-type estimators that use cloning - resampling from the estimated density - instead of ordinary BS by appending an asterisk. For example, to construct $Err^{(1)}_{BS}*$, the $X^{(n)*}_{(i)}$ in (8) is a the clone of the $X^{(n)}\backslash\{\mathbf{x}_i\}$ rather than its BS replica. From the previous section, we know that this change does not increase the computational intensity of the procedure.

5.2 Bootstrapped cross-validation

An alternative approach is to smooth the cross-validated estimation by bootstrapping the data on which the computation is performed. Here too the cloning method may replace the ordinary bootstrap. Incorporation of a BS or cloning allows us to obtain a less noisy estimate. To smooth the discontinuities in the results of CV caused by perturbations in the training and test sets, CV is performed on each BS sample (or clone) of $X^{(n)}$:

$$Err_{BSCV\times k} = \frac{1}{B}\sum_{b=1}^{B} Err_{CV\times k}\left(X^{(n)}{}^{*}_{b}\right). \tag{18}$$

The computational cost is similar to that of computing $Err_{RCV\times k}$ with B trials.

6 Simulation study with synthetic data

In order to test the plausibility of the proposed estimators, we followed the experimental framework of [13], described in Table 1. In all cases, there were two classes, represented in the by an equal number of data points. In experiments 2 and 4, the distribution of the data was independent of the class label, and each data point had probability of 1/2 to be assigned to each class, so that any classifier had an expected accuracy of 1/2. The true error *Err* was approximated by testing each trained classifier on 20,000 data points drawn from the same distribution (a validation set). Both the training data and the validation data are balanced: the same amount of data is assigned to each class. In all the experiments, the classifiers used were 1- and 3-NN, Fisher's Linear Discriminant Functions (LDF), and Support Vector Machines (SVMs) with radial basis function (RBF) kernels. The σ parameter for the SVMs was chosen from a range of possible values by minimizing the generalization error over a large set of test points. Except for using the SVMs, this setup is identical to that reported in [13]. We repeated the computation of different estimators for a number of *trials.* In each trial, to reduce the variability in the differences due to random factors, the same data sets and the same bootstrap samples and clones were used for all of the estimators.

In Table 2 we compare the best of the new methods to the best of the old ones, described in section 2. All the statistics were computed from the corresponding number of trials given in Table 1. To determine the statistical significance of these results, we perform hypothesis testing for each classifier/data set. The null hypothesis is that the expected squared errors of

Table 1: Setting of the simulation with synthetic data described in Section 6. All data sets have two balanced classes. In 2 and 4 the distribution of a point is independent of its class label. The size of each data set is given by n.

#	Distribution	n	Trials
1	$N\left((\pm 1, 0, 0, 0, 0)^T, \mathbf{I}_5\right)$	14	200
2	$N\left(\mathbf{0}_5, \mathbf{I}_5\right)$	14	200
3	$N\left((\pm .5, 0)^T, \mathbf{I}_2\right)$	20	200
4	$N\left(\mathbf{0}_2, \mathbf{I}_2\right)$	20	200
5	$N\left(\mathbf{0}_{10}, \mathbf{I}_{10}\right)$ vs. $\prod_{j=1}^{10} N\left(\sqrt{j}/2, 1/j\right)$	100	50

the two methods are equal. We attempt to reject this hypothesis in favor of the alternative that the expected error of the new method is smaller. Since all of the estimators are applied to the same data set in each trial, the obtained values are dependent, and the paired single-tailed test for comparison of the means is appropriate [27]. The sample of interest here is $d = (\widehat{Err}_{old} - Err)^2 - (\widehat{Err}_{new} - Err)^2$. The significance α of the test corresponds to the z-value $z_\alpha = \mu_d/(\sigma_d/\sqrt{N})$, where N is the number of trials, μ_d is the sample mean, and σ_d is the sample variance of the differences. These values appear in the last three columns of Table 2.

In 17 out of 20 cases, one of the new estimators was significantly better ($\alpha \leq .01$) in terms of RMSE than any of the old ones. While our estimators are sometimes more biased, their variances are noticeably smaller, as hoped. We note the particularly good performance of $Err_{.632+}*$, which in 12 cases was better than $Err_{.632+}$, and of $Err_{BSCV\times 5}*$, which outperformed all versions of $Err_{CV\times k}$ and $Err_{BSCV\times k}$ in 16 cases. The detailed results for all the estimators can be found in Appendix A.

Table 2: Results on synthetic data. Err is the true accuracy estimated on a large validation set. $\widehat{Err}_{\mathbf{new}}$ corresponds to the best-performing (in terms of RMSE) new estimator; $\widehat{Err}_{\mathbf{old}}$ to the best-performing among the old ones for each classifier and data set. Significantly better RMSE shown in bold.

#	Classifier	Err	$\widehat{Err}_{\mathbf{new}}$	Mean	STD	RMSE	$\widehat{Err}_{\mathbf{old}}$	Mean	STD	RMSE	$\sqrt{\mu_{\mathbf{d}}}$	$\sigma_{\mathbf{d}}$	α
1	LDF	.262	$Err_{BSCV\times n}*$	.2668	.0626	.0767	Err_{BS}	.2537	.0662	.0776	.0121	.006	.37
	1-NN	.2785	$Err_{.632}*$	.217	.0707	**.0863**	$Err_{.632+}$	.256	.1139	.0999	.05	.0105	.01
	3-NN	.2462	$Err_{BSCV\times 5}*$	.24	.0651	**.0598**	Err_{BS}	.199	.0745	.0815	.055	.0061	10^{-13}
	SVM	.2367	$Err_{BSCV\times 5}*$	.247	.0632	**.0671**	$Err_{.632}$	.146	.0626	.1108	.055	.0061	10^{-13}
2	LDF	.5	$Err_{BSCV\times n}*$	.511	.0729	.0731	$Err^{(1)}_{BS}$	.507	.0705	.0703	-.02	.0024	.15
	1-NN	.5	$Err_{BSCV\times n}*$	.535	.0729	**.0803**	$Err^{(1)}_{BS}$	.54	.1051	.1118	.078	.0137	10^{-10}
	3-NN	.5	$Err_{.632+}*$	.4585	.0526	**.0668**	$Err_{.632+}$	.4588	.066	.0772	.039	.0066	.01
	SVM	.5	$Err_{.632+}*$	.412	.0472	.1005	$Err_{.632+}$	.4173	.0562	.1003	-.0057	.0084	.48
3	LDF	.35	$Err_{BSCV\times 5}*$	.35	.0794	**.0813**	Err_{BS}	.313	.091	.0996	.0575	.0078	10^{-10}
	1-NN	.414	$Err_{BSCV\times 5}*$	.363	.0646	**.0737**	$Err_{.632+}$	.354	.0834	.1002	.068	.0106	10^{-10}
	3-NN	.392	$Err_{BSCV\times 5}*$	.361	.0679	**.0689**	$Err_{.632+}$	.391	.0848	.0816	.044	.0067	10^{-5}
	SVM	.356	$Err_{BSCV\times 5}*$	.368	.072	**.0776**	Err_{BS}	.32	.0886	.0959	.056	.01	10^{-6}
4	LDF	.5	$Err_{.632+}*$	.471	.0771	**.0828**	$Err_{.632+}$	.468	.086	.0924	.041	.0026	10^{-19}
	1-NN	.5	$Err^{(1)}_{BS}*$	.527	.0572	**.0633**	$Err^{(1)}_{BS}$	.533	.1005	.1061	.085	.0126	10^{-15}
	3-NN	.5	$Err_{.632+}*$	.458	.0457	**.0625**	$Err_{.632+}$	.457	.0622	.0763	.0438	.0065	10^{-5}
	SVM	.5	$Err_{.632+}*$	.485	.0511	**.0537**	$Err_{.632+}$	.484	.0614	.0641	.035	.0044	.0002
5	LDF	.018	$Err_{BSCV\times n}*$	.009	.0084	.0148	$Err_{.632}$	.0085	.0076	.0147	.0017	$5\cdot 10^{-5}$	.33
	1-NN	.022	$Err_{BSCV\times n}*$	.016	.0094	**.0123**	$Err^{(1)}_{BS}$	.014	.01	.0141	.007	$3\cdot 10^{-5}$	0
	3-NN	.027	$Err_{BSCV\times 10}*$	.0171	.0073	**.013**	$Err^{(1)}_{BS}$	.0171	.0106	.0156	.009	.0001	10^{-6}
	SVM	.0036	$Err_{BSCV\times n}*$	.002	.0018	**.0029**	$Err_{.632}$	.001	.0018	.0033	.0015	10^{-6}	10^{-7}

7 Cloning real-life data sets

So far we have only considered data sets with unbounded continuous attributes. We now turn to the issue of extending our methods to real-life data. In general, the data space $\mathcal{D}$ of an arbitrary problem domain can be represented as

$$\mathcal{D} \subseteq \mathbb{R}^{m_c} \times \mathbb{Z}^{m_o} \times \mathbb{A}_1 \times \ldots \times \mathbb{A}_{m_n} \quad , \tag{19}$$

where m_c, m_o and m_n are the numbers of continuous, discrete, and nominal attributes, respectively. $\mathbb{A}_i$ denotes a nominal domain (possibly different for each i). We shall denote a general point in $\mathcal{D}$ by $\langle \mathbf{v}, \mathbf{u}, \mathbf{w} \rangle$, where $\mathbf{v} \subseteq \mathbb{R}^{m_c}$ etc.

Straightforward kernel density estimation as in (9) is not applicable in the general case, since it assumes that the data are continuous and unbounded.

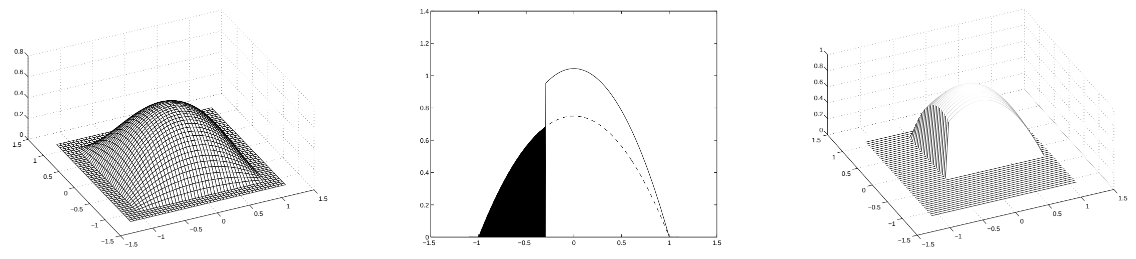
7.1 Cloning continuous data with boundaries

The standard kernel estimate (9) becomes inaccurate near points of discontinuity of the actual *PDF*. Such a discontinuity occurs at an endpoint of an interval outside which the density vanishes. Kernel estimate overestimates the density outside the boundaries. Most boundary correction techniques mentioned in the literature involve kernel functions with negative values (see [30]). Such a kernel is no longer a *PDF*, making sampling problematic. In addition, different *PDF*s behave in different ways at boundaries. For instance, at $x \to 0_+$

the negative exponential *PDF* tends to ∞, while the *PDF* of the Γ-distribution vanishes. No general solution to the task of finding the "right" type of boundary kernel to fit the specific behavior of the estimated *PDF* is known to date. Moreover, one may need to apply different boundary kernels at different boundaries for the same data set in order to obtain reasonable behavior.

To generate boundary-obeying values, we use the following method. Let D be the set of admissible values ("inside" the boundaries). We force the modified kernel $K_D(x)$ to be zero outside D and divide by $s = \int_{x \in D} K(x)dx$ in order for the kernel K_D to integrate to unity, as illustrated in Figure 1(b). The shape of a boundary-corrected 2-D product kernel shown in Figure 1(c).

Figure 1: Boundary correction for the Epanechnikov kernel $K(x) = 3(x-1)^2/4$. (a) 2-dimensional product kernel (b) Boundary-corrected kernel in one dimension. The density is known to vanish outside $[-.3, 1]$. The dashed line shows the original kernel; the rejection region is shaded (c) Boundary-corrected product kernel in 2 dimensions. The density vanishes outside $[-.3, 1] \times [-.5, 1]$.

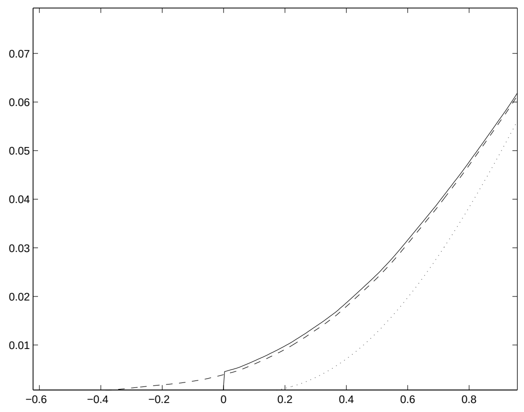


(a) (b) (c)

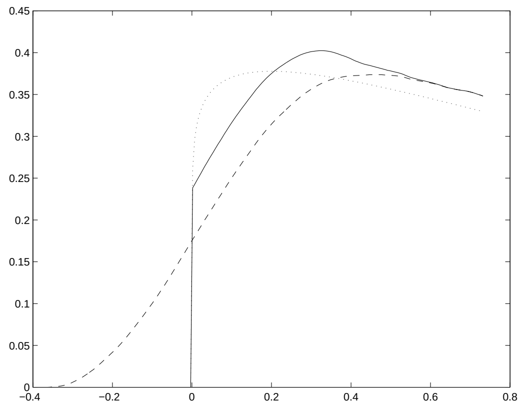
An example of applying our boundary correction method is shown on Fig 2, for two gamma distributions, that vanish for $x < 0$. In each case, the corresponding gamma PDF was estimated based on a random sample of 1000 points. Boundary correction has no effect for points that are farther from the boundary than h, therefore we look at the improvement in PDF estimate only over $[-h, h]$, where h is the chosen bandwidth. For $\Gamma(4, 1)$, shown in Figure 2(a), the improvement is small, and only due to the region $x < 0$ where the density is overestimated by the non-corrected kernel. Within the support region of the true density the boundary corrected kernel is in fact slightly worse. Overall, the L_1 error in the estimate is reduced by about 1%. In case of $\Gamma(1.1, 2)$, shown in Figure 2(b), the shape of the function near boundary is much closer to that of the kernel, and the boundary correction results in a much more well-behaved estimate. The reduction in L_1 error in this case is more than 60%.

Sampling from this boundary-corrected kernel is straightforward. We repeat the algorithm for sampling from $\hat{f}$ until the result falls within D. The failure probability of a single trial is $1 - s$. The algorithm is always applied for an actual data point in the original data set, which is within the boundaries. Therefore, as long as the point has a neighborhood over which the distribution is positive, a positive lower bound can be established on s, from which an upper bound on the expected time to success can be derived.

Figure 2: Boundary corrected estimation of gamma distribution with different parameters. The true distribution (dotted line), non-corrected kernel estimate (dashed line), and our boundary-corrected estimates are shown over $[-h, h]$.



(a) $\Gamma(4, 1)$, $h = .95$



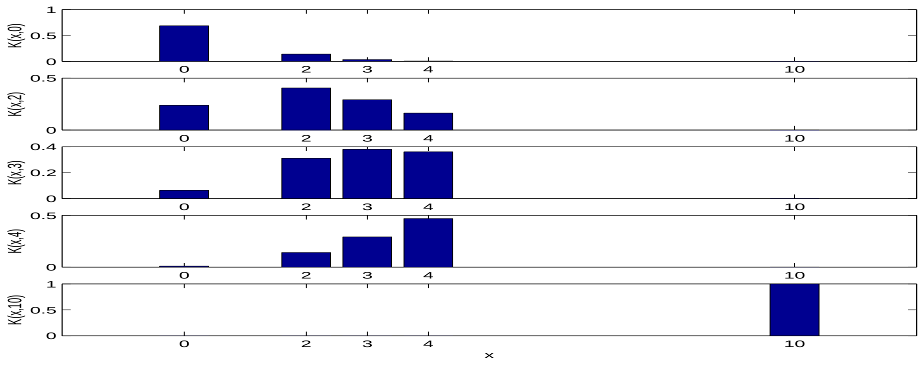
(b) $\Gamma(1.1, 2)$, $h = .37$

7.2 Cloning data with discrete attributes

Discrete feature values come from a domain with countable cardinality. We, however, observe only a finite subset of it with cardinality T. We assume that a corresponding metric and order are defined, in contrast to nominal domain (see Section 7.3). Such a domain can be mapped into $\mathbb{Z}$. In our experience, applying kernel functions of a continuous variable to a discrete attribute might lead to poor results when applied to resampling, and we do not see clear and natural solution to the related rounding problem. Instead we follow [1] and use the following family of discrete kernels:

$$K_h^d(x, x_i) = K^d(h, x, x_i) \;=\; \frac{h^{\|x-x_i\|^2}}{\sum_{k=1}^{T} h^{\|x-x_i\|^2}}. \tag{20}$$

Figure 3: A family of discrete smoothing kernels. The values of $K_h^d(x, x_i)$ are shown for $X^{(5)} = \langle 0, 2, 3, 4, 10 \rangle$.



$K_h^d(x, x_i)$ is centered on x_i and assigns to x a weight that is inversely proportional to its distance from x_i; the rate at which it drops depends on h, which is between 0 and 1. Since we have no theoretically solid method of choosing h, the following heuristic was used to avoid over-smoothing. For each x, we would like to keep a large proportion, say, 95% of the probability mass assigned to x by the empirical distribution $\hat{F}_n$ close to x. We take the standard deviation σ_x of the values of x as a measure of spread, and set $h = .05^{1/\sigma_x^2}$, which enforces this spread constraint, since for a y which has $\|x - y\|^2 = \sigma_x$, we will have $h^{\|x - x_i\|^2} = .05$. The discrete attributes are sampled similarly to the continuous ones, with the substitution of K^d for K, as described in Algorithm 3. Figure 3 shows an example of a kernel family, computed for a discrete data set.

As a note of caution, we point out that, in some data sets, the integer attribute serves as a label or encodes nominal values. In such cases, there is no meaning to the metric or the order defined over $\mathbb{Z}$, and smoothing the resampling may be meaningless. Such an attribute

Algorithm 3 Resampling a discrete attribute.

```
Input: An ordinal attribute domain $\mathbb{A} = \{a_1, \dots, a_T\}$, a value $v \in \mathcal{A}$, and a set of kernels
    $\{K_h^d(\cdot, \cdot)\}$ defined by the data set $X^{(n)}$ as in (20).
Output: $u \in \mathbb{A}$.
  for $i = 1$ to $T$ do
    Let $p_i = K_h^d(a_i, v)$.
  end for
  Let $P = \langle p_1, \dots, p_T \rangle$.
  Draw $i$ according to a polynomial distribution with probability vector $P$.
  Return $u \leftarrow a_i$.
```

should be treated as nominal.

7.3 Cloning data with nominal attributes

The support space of a *nominal* (sometimes called *categorical*) attribute is a non-metric unordered space $\mathbb{A}$ of finite cardinality T. In this case there is no reasonable meaning for a distance between two possible values.

Smoothing of the marginal distribution of nominal attributes could introduce impossible data points for which the true probability to appear in the given domain is zero. For example, in the Adult data set in [3], in which data of different people are labeled with income level, two of the nominal attributes are “marital status” and “relationship”. If we allow smoothing over the nominal domain, we might create a data point with the value of the attribute “relationship”, say, “Husband”, but with “marital status” set to “Never married” - an impossible combination. We feel that this hazard overwhelms the possible benefit from enriching the cloned data, and choose not to perform smoothing directly on a nominal domain. For data with multiple nominal attributes, this means that a combination of nominal attribute values can appear in the clone only if it appears in the original sample.

To sample from the conditional probability mass function of a nominal attribute, we need to estimate it explicitly. Abusing notation, denote by f both the joint *PDF* over the whole domain, and its marginals over sub-domains. Recall from (19) that we are estimating the density at a "mixed point" $\langle \mathbf{v}, \mathbf{u}, \mathbf{w} \rangle \in \mathcal{D}$:

$$\begin{aligned} f(\langle \mathbf{v}, \mathbf{u}, \mathbf{w} \rangle) &= f(\langle \mathbf{v}, \mathbf{u} \rangle) f(\mathbf{w} | \mathbf{v}, \mathbf{u}) \\ &= f(\langle \mathbf{v}, \mathbf{u} \rangle) f(w_1 | \mathbf{v}, \mathbf{u}) f(w_2 | \mathbf{v}, \mathbf{u}, w_1) \dots f(w_{m_n} | \mathbf{v}, \mathbf{u}, w_1, \dots, w_{m_n - 1}). \end{aligned} \tag{21}$$

Thus in order to estimate $f(\langle \mathbf{v}, \mathbf{u}, \mathbf{w} \rangle)$, one can find $f(w_j | \mathbf{v}, \mathbf{u}, w_1, \dots, w_{j-1})$ for all $j = 1, \dots, m_n$. Of course, all the conditional densities appearing above are estimated rather than used with their true (unknown) values.

For simplicity we use the following "generalized kernel" notation:

$$K(x, x_i) = \begin{cases} I(x, x_i) & \text{if } x, x_i \in \mathbb{A}_s, \quad 1 \le s \le m_n; \\ \frac{1}{h} K(\frac{x - x_i}{h}) & \text{if } x, x_i \in \mathbb{R}; \\ K_h^d(x, x_i) & \text{if } x, x_i \in \mathbb{Z}, \end{cases} \tag{22}$$

where $I(x, x_i)$ denotes the identity function of the corresponding nominal domain, K is the chosen univariate kernel, K_h^d is the ordinal kernel, and h is the bandwidth (smoothing parameter) for the corresponding dimension, automatically set as described in Sections 4.1 and 7.2. In these terms,

$$\hat{f}(\langle \mathbf{v}, \mathbf{u}, w_1, \dots, w_j \rangle) = \frac{1}{n} \sum_{i=1}^{n} \prod_{s=1}^{j} K(w_s, w_{is}) \prod_{l=1}^{m_c} K(u_l, u_{il}) \prod_{m=1}^{m_o} K(v_m, v_{im}). \tag{23}$$

Clearly,

$$\sum_{w_j \in \mathbb{A}_j} \hat{f}(w_j | \mathbf{v}, \mathbf{u}, w_1, \dots, w_{j-1}) \; = \; \hat{f}(\langle \mathbf{v}, \mathbf{u}, w_1, \dots, w_{j-1} \rangle). \tag{24}$$

All that remains is to normalize the values obtained in (23) by dividing them by their total sums for all possible values of w_j. Now we are ready to generate a value for the j-th nominal feature, such that it is distributed according to the estimated distribution. This procedure is given in Algorithm 4.

Algorithm 4 Resampling a nominal value from an unordered domain with finite cardinality, given the values of the numeric features and the preceding nominal features.

```
Input: A sample $X^{(n)}$, a point $\langle \mathbf{v}, \mathbf{v}, w_1, \dots, w_{j-1} \rangle$, an index $1 \leq j \leq m_n$.
Output: $w \in \mathbb{A}$.
  for all $a \in \mathbb{A}$ do
    Let $p_a \leftarrow \hat{f}(a, w_1, \dots, w_{j-1}, \mathbf{u}, \mathbf{v})$ as given by (23).
  end for
  Let $S \leftarrow \sum_{a \in \mathbb{A}} p_i$.
  Let $p_a \leftarrow p_a / S$.
  Draw $a$ according to a polynomial distribution with probability vector $\mathbf{p} = [p_1, \dots, p_T]$.
  Return $w \leftarrow a$.
```

7.4 Cloning a data set

Building a clone of a complex (that is, multi-typed) data set $X^{(n)}$ consists of repeated independent generation of points $\langle \mathbf{v}, \mathbf{u}, \mathbf{w} \rangle$. Algorithm 5 describes the process. The smoothing parameters for the numerical attributes are set automatically, based on the input. Therefore, the cloning algorithm does not require tuning or setting any parameter by the user. Note, however, that boundaries for the numeric values are part of the definition of $\mathcal{D}$ and should be explicitly provided to the algorithm.

Algorithm 5 Cloning a data set from domain $\mathcal{D}$.

```
Input: A data set $X^{(n)}$ of points in $\mathcal{D}$, integer $R$
Output: A cloned data set $\mathbf{X}^{(R)^*}$ of size $R$ defined on the same space, so that $\hat{f}(\mathbf{x}) = \hat{f}(\mathbf{x}^*)$.
  Set the smoothing parameters:
  $h_1, \ldots, h_{m_c} \in \mathbb{R}$ for the continuous kernels, Section 4.1,
  $h_{m_c+1}, \ldots, h_{m_c+m_o} \in (0,1)$ for the ordinal kernels, Section 7.2.
  for $k = 1$ to $R$ do
    Draw an index $i$ uniformly from $\{1, \ldots, n\}$.
    Let $\langle \mathbf{v}, \mathbf{u}, \mathbf{w} \rangle \leftarrow \mathbf{x}_i$.
    Generate $[v_1^*, \ldots, v_{m_c}^*]^T$ from $[v_1, \ldots, v_{m_c}]^T$ by Alg. 1.
    for $r = 1$ to $m_o$ do
      Generate $u_r^*$ from $u_r$ by Alg. 3.
    end for
    for $s = 1$ to $m_n$ do
      for all $a \in \mathbb{A}_t$ do
        Estimate $p(a) = \hat{f}(a|\mathbf{v}^*, \mathbf{u}^*, w_1^*, \ldots, w_{s-1}^*)$ by Alg. 4.
        Draw $w_s^* \sim p(a)$.
      end for
    end for
    Let $\mathbf{x}_k^* \leftarrow \langle \mathbf{v}^*, \mathbf{u}^*, \mathbf{w}^* \rangle$.
  end for
```

8 Simulation study with data from UCI machine learning repository

For the experiments with real data, we used 3 data sets from the UCI Machine Learning repository [3]. Table 3 gives the settings for the simulation. The data sets used were the Wisconsin Breast Cancer, Vehicle and Pima Indians Diabetes. Breast Cancer and Vehicle were the only two real data sets used in [13]. We chose Pima since it has complex domain with both continuous and integer attributes, which allowed us to examine some of the algorithms developed in Section 7. For each data set, we chose classifiers reported to work well on that domain. In each trial, a small data subset $X^{(n)}$ of size n was chosen as the input for the experiments. A classifier was then trained on this $X^{(n)}$ and tested on the remaining larger subset to estimate the conditional true error. This value was then compared to the predictions of different estimators, to which only $X^{(n)}$ was made available. For example, in a trial on the Pima data set, a subset of 60 points was drawn uniformly at random from the entire data set and a classifier, e.g., an SVM, was trained on these 60 data points. Then the error of the classifier was computed on the remaining 708 data points, and the result was taken as an approximate value of Err. We next estimated the error of the SVM trained on the chosen 60 points, and compared the estimate to the value of Err computed as above. For SVMs on the multi-class Vehicle data set, we used the max-win paradigm: decomposition of the classification problem into a number of binary classification problems, and combining the resulting classifiers by a voting scheme with values of the discriminant function as the weights.

Table 3: Settings for the simulation study with UCI data sets (Section 8).

Data set	Size	n	Number of features	Type of features	Number of classes	Number of trials
Breast Cancer	683	36	9	all integer, bounded	2	150
Vehicle	846	100	18	all continuous, bounded	4	100
Pima	768	60	8	6 integer, 2 continuous, all bounded	2	150

Table 4: Results on UCI data sets, experiments in Section 8.

Data set	Classifier	Err	$\widehat{Err}_{\mathbf{new}}$	Mean	STD	RMSE	$\widehat{Err}_{\mathbf{old}}$	Mean	STD	RMSE	$\sqrt{\mu_{\mathbf{d}}}$	$\sigma_{\mathbf{d}}$	α
Breast cancer	1-NN	.052	$Err_{.632+}*$	.036	.0259	.0269	$Err_{.632}$	.035	.0255	.0271	.0025	.0002	.3
	3-NN	.045	$Err_{BSCV\times 5}*$	.037	.0225	.0248	Err_{BS}	.036	.0234	.0259	.007	.0002	.002
	SVM linear	.04	$Err_{BSCV\times 5}*$	.028	.0203	.0254	Err_{BS}	.028	.0206	.0259	.005	.0002	.058
	SVM RBF	.04	$Err_{.632}*$	.027	.0199	.022	$Err_{.632}$	.028	.0209	.0222	.003	.0001	.09
Pima	17-NN	.33	$Err_{BSCV\times 5}$	.3135	.0235	.0284	$Err_{RCV\times 10}$	.333	.0236	.025	-.013	.001	.019
	SVM RBF	.294	$Err_{.632+}*$	.302	.0567	.0584	$Err_{.632+}$	.286	.0596	.0624	.022	.003	.0355
Vehicle	1-NN	.438	$Err_{BSCV\times 5}*$	.447	.0278	.0338	$Err_{BS}^{(1)}$	.464	.0431	.0537	.042	.004	10^{-5}
	SVM RBF	.351	$Err_{BSCV\times 5}*$	.334	.0264	.0399	$Err_{.632+}$	.321	.0384	.0567	.04	.003	10^{-8}

As with the synthetic data, in the majority of cases (6 out of 8) one of the new estimators performed significantly better in terms of RMSE than any of the old estimators, at a significance level .1 (Table 4). In most cases, the winning estimator had the lowest variance. In Table 5 we show that the cloned version of $Err_{.632+}$ outperformed its traditional counterpart in 6 out of 8 cases with significance below .05, and in a seventh case with significance .15. We also show (Table 6), in accordance with the suggestion in [29], that repeating the standard CV multiple times on the same data provides only a partial solution, in exchange for a significant increase in computational cost. In particular, $Err_{.632+}*$ clearly outperforms $Err_{RCV\times k}$ for various values of k (in 7 out of 8 cases), and $Err_{BSCV\times 5}*$ was significantly better in 5 out of 8, with one tie.

Table 5: RMSE of .632+ with and without cloning. The better estimator for each test is shown in bold. α is the significance level of the test.

	Breast Cancer				Pima Indians		Vehicle	
	1-NN	3-NN	SVM linear	SVM RBF	17-NN	SVM RBF	1-NN	SVM RBF
$Err_{.632+}$	.0275	.0307	.0276	.0230	**.0301**	.0624	.076	.0567
$Err_{.632+}*$	**.0269**	**.0297**	**.0271**	**.0225**	.0324	**.0584**	**.0417**	**.0438**
α	.15	.0093	.0064	.0262	.0463	.0355	10^{-11}	10^{-6}

Table 6: RMSE performance of repeated CVs versus the proposed estimators. '-' indicates that $Err_{RCV\times k}$ is worse, '+' that it is better, and '?' that it is not different with significance below .1

	$Err_{.632+}*$			$Err_{BSCV\times 5}*$			$Err_{BSCV\times n}*$		
	-	+	?	-	+	?	-	+	?
$Err_{RCV\times n}$	8	0	0	6	1	1	4	2	2
$Err_{RCV\times 10}$	7	1	0	5	2	1	4	2	2
$Err_{RCV\times 5}$	7	0	1	6	2	0	4	2	2

Figure 4 displays the behavior of $Err_{.632+}$ (dashed line) and of $Err_{.632+}*$ (solid line) for an SVM classifier, applied on two of the UCI data sets used in the simulation. For the Pima data set (left column, (a)-(c)), the cloning-based estimator is more biased than $Err_{.632+}$. However, as Figure 4(b) shows, the estimated PDF of its absolute error has a thinner tail, corresponding to its lower variance. Integration over possible error values produces the estimated probability distribution of absolute error, shown in Fig.4 (c). For the values of absolute error ϵ that exceed .033, the graph corresponding to the cloning-based estimator stays below the graph for $Err_{.632+}$. The results for the Vehicle data set (right column of Figure 4, (d)-(f)) are even clearer. In particular, for any value of ϵ, the estimated probability of the absolute error in $\widehat{Err}$ to exceed ϵ is greater for the traditional $Err_{.632+}$. This behavior of absolute error is consistent with the relative bounds that can be established using (4).

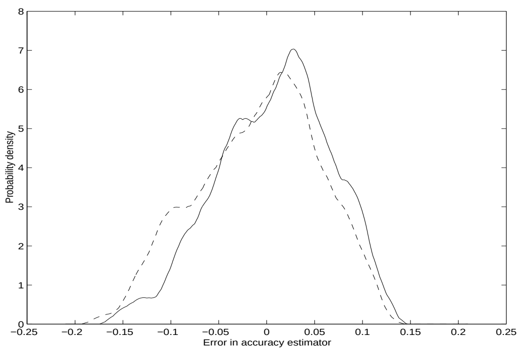
The bottom two plots of Fig. 4 show, for each estimator, the estimated probability distribution of absolute error. These probabilities, that correspond to the left-hand side of (4), exhibit behavior consistent with the prediction by (4).

The case in which the cloning-based estimators failed compared to the traditional ones is the 17-NN classifier on the Pima data set. (We chose this classifier because of the reported good performance of k-NN with $k = 17, 19$ on this domain [26].) As can be seen in Figure 5, $Err_{.632+}*$ had larger variance than $Err_{.632+}$, and consequently had a higher RMSE.

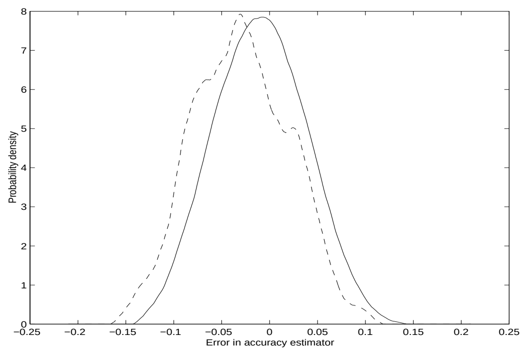
Some graphical representation of the resampling results obtained with our system is given in Figures 6-7. Each figure presents a two-dimensional subspace of the original 8-dimensional Pima data set(the upper subplot) and a clone (the lower subplot). Dimensions are labeled by their indices in the data set. Attribute 6 is continuous , while attributes 1,3 and 5 are discrete. Figure 7 reveals a certain shortcoming of the current algorithm. The kernel method is still unable to capture some properties of real life data. Here, there is a cluster of points with the 5th attribute value being zero. We believe this is due to the true density being a mixture of some smooth density and a δ-function in zero. However, due to the smoothing, this property of the data is not represented in the clone, and is preserved in the ordinary BS sample. In our experience this phenomenon is not uncommon in real data sets. This artifact may be critical for some classifiers, such as decision trees, where

Figure 4: Analysis of the results for an SVM with an RBFkernel, on the Pima (left column) and Vehicle (right column) data sets. Solid lines correspond to the cloned .632+, and dashed lines to the traditional one.

Estimated distribution of error $\widehat{Err} - Err$

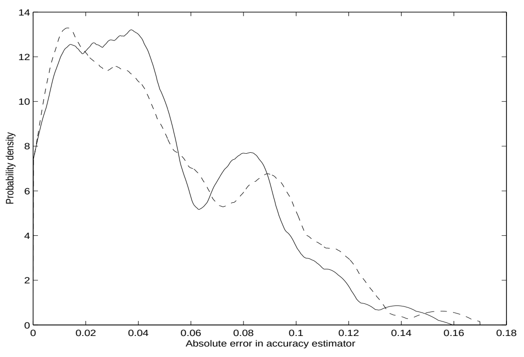


(a)

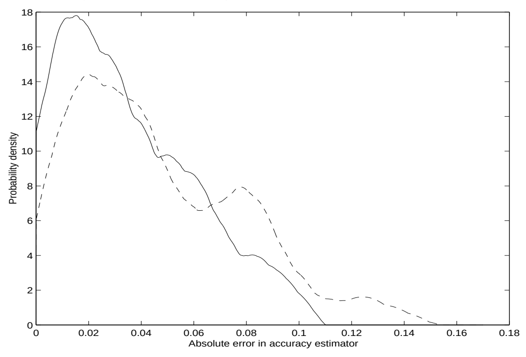


(d)

Estimated distribution of $|\widehat{Err} - Err|$

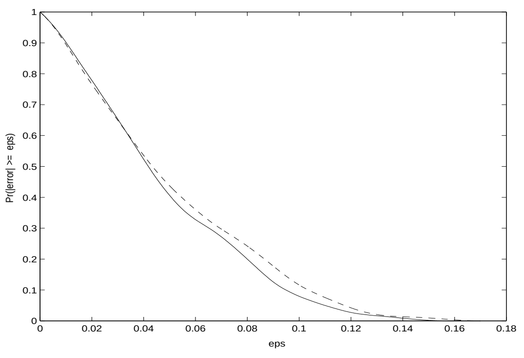


(b)

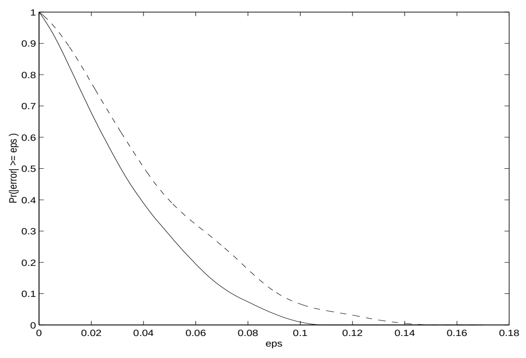


(e)

Estimated probability of the absolute error to exceed ϵ.



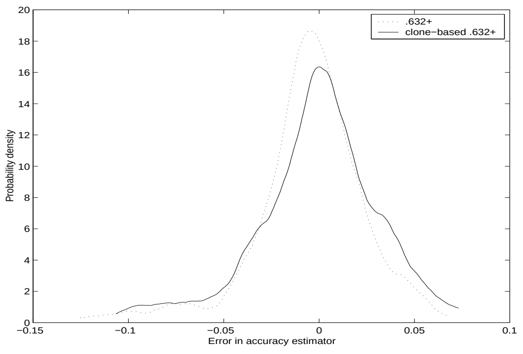
(c)



(f)

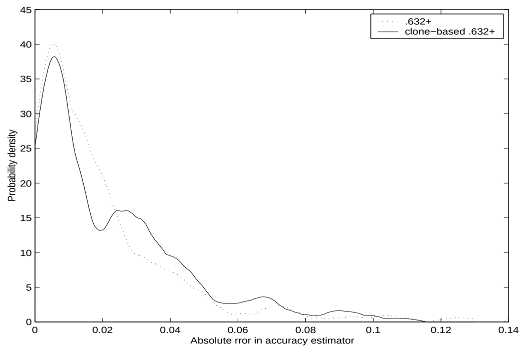
Figure 5: Analysis of the results for the Pima data set, with the 17-NN classifier. The cloned .632+ (solid line) does not achieve lower variance than .632+ (dashed line), and shows worse performance.

Distribution of $\widehat{Err} - Err$



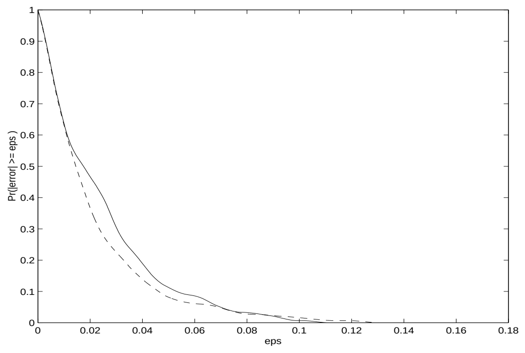
(a)

Distribution of $|\widehat{Err} - Err|$



(b)

Probability of the absolute error to exceed ϵ



(c)

an important rule may involve testing equality to an exact value. For comparison, Figure 7 shows an ordinary BS sample of Pima data as well. While the phenomenon mentioned above does not affect the BS sample (note a cluster of points with $x_5 = 0$), there are much fewer distinct points than the original. The original data set has 768 distinct points, and so does the clone, but the displayed ordinary BS sample contains only 483 distinct points, which is very close to .632*768.

The actual computation cost of various estimators in our experiments was consistent with the predictions based on the expected running times. Table 7 gives the CPU time, in seconds, averaged over 30 trials for each estimator, on the three UCI data sets. The cloning-based estimators required only slightly more time than their respective ordinary BS-based counterparts. We ran the experiments, implemented in C++ and Matlab, on a dual-600MHz machine running Linux. Appendix B briefly describes the data cloning software package written for this purpose.

Figure 6: A look at the original versus a clone of the Pima data set, for two two-dimensional subspaces. The upper plot is the original data set, the lower plot corresponds to a clone.

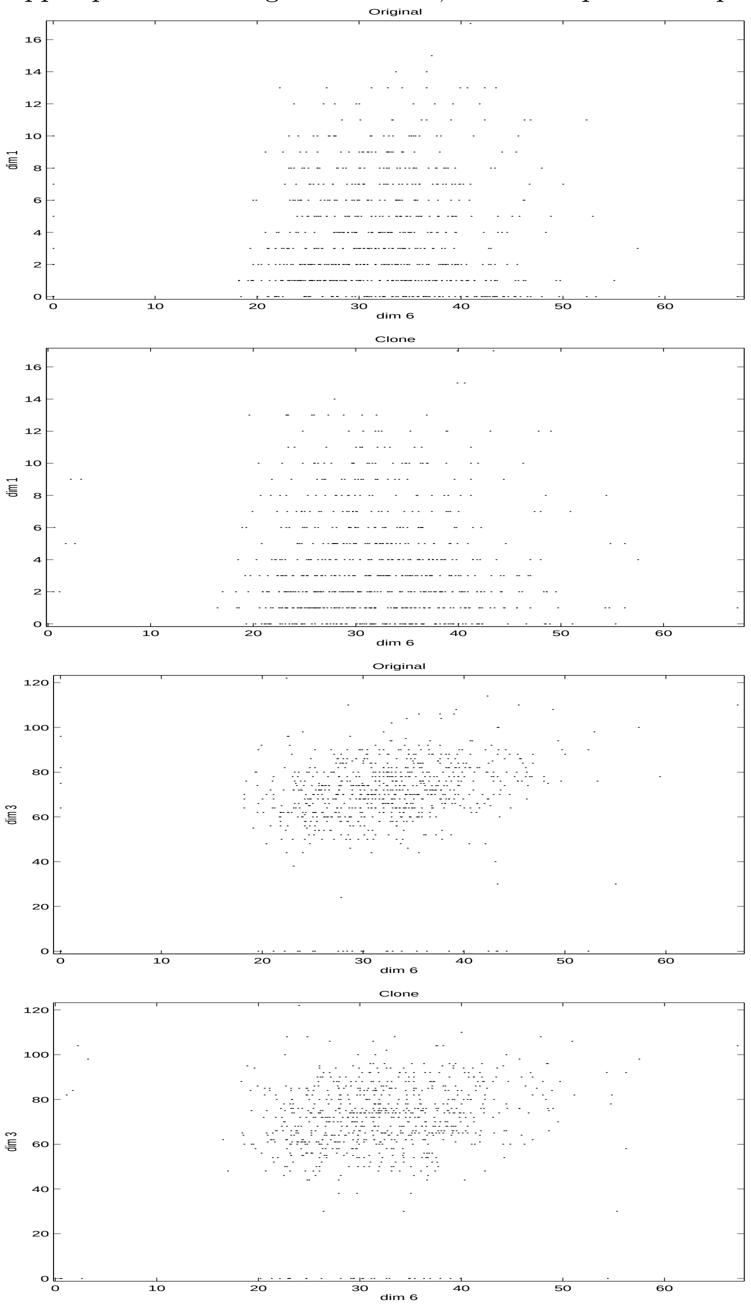


Figure 7: Original (the upper plot) versus a clone (second from above) and an ordinary BS sample (the lower plot) of the Pima data set. Note the cluster of points with dim5=0, which is not properly represented in the clone. BS sample is better in this sense, but has only 483 distinct points, compared to 768 in the original and the clone.

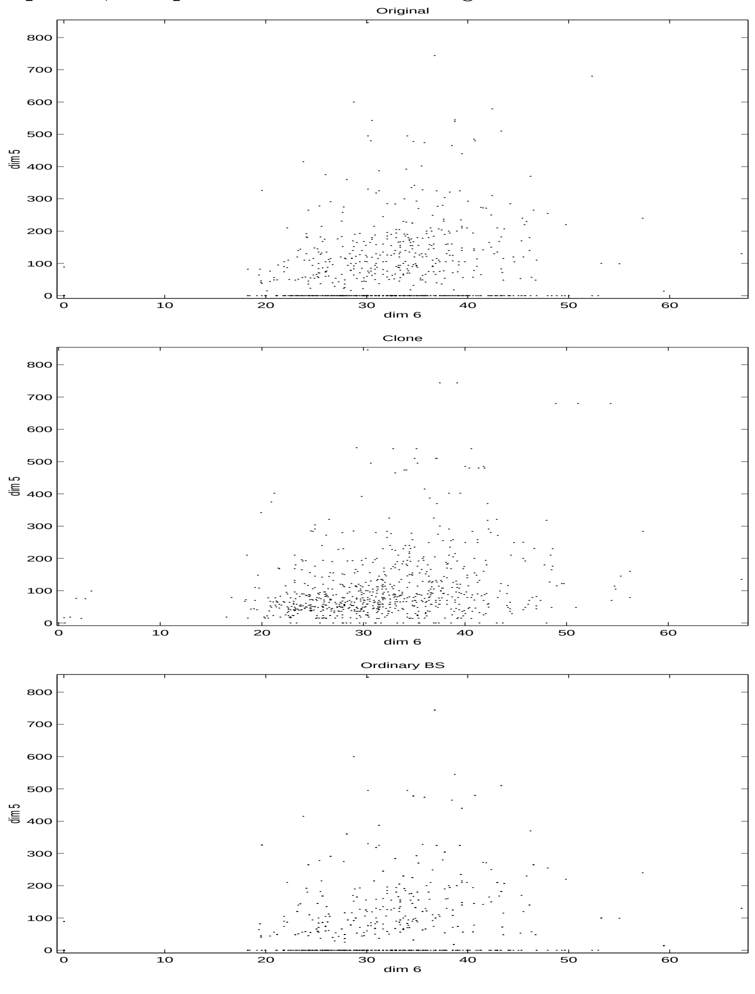


Table 7: Average CPU time in seconds for computing a single instance of an estimator in the UCI data experiments, on a dual 600MHz processor machine. Resampling algorithms are implemented in C++; the rest of the computation was done in Matlab.

Estimator	Breast Cancer, $n = 36$				Vehicle, $n = 100$		Pima, $n = 60$	
	1-NN	3-NN	linear SVM	RBFSVM	1-NN	RBFSVM	17-NN	RBFSVM
Err	0.06	0.07	0.02	0.03	0.23	0.68	0.14	0.07
$\overline{\text{err}}$	0.00	0.01	0.01	0.01	0.03	0.25	0.02	0.04
$Err_{.632}$	26.24	26.52	55.87	54.15	136.81	1563.76	58.91	174.79
$Err_{.632+}$	26.10	26.62	55.22	54.03	136.93	1541.28	60.08	173.12
$Err_{CV\times n}$	0.10	0.11	0.49	0.49	0.40	18.98	0.29	2.00
$Err_{CV\times 10}$	0.03	0.03	0.13	0.13	0.06	1.59	0.05	0.31
$Err_{CV\times 5}$	0.02	0.02	0.06	0.06	0.04	0.68	0.03	0.13
Err_{BS}	0.99	1.02	1.57	1.55	3.91	20.82	1.94	3.14
$Err_{BS}^{(1)}$	26.08	26.79	54.81	53.49	137.12	1588.49	60.08	174.02
$Err_{BS}^{\times 2}$	1.02	1.05	1.62	1.61	3.95	21.08	2.00	3.08
$Err_{BSCV\times n}$	10.23	10.87	40.63	40.04	41.04	1469.27	28.46	145.62
$Err_{BSCV\times 10}$	3.25	3.48	11.36	11.14	6.76	127.49	5.96	22.48
$Err_{BSCV\times 5}$	1.99	2.08	5.62	5.47	4.64	55.75	3.62	10.42
$Err_{.632}*$	25.36	26.16	62.83	63.08	124.95	2026.89	57.74	205.13
$Err_{.632+}*$	25.49	26.13	61.40	63.41	124.59	2072.86	57.65	204.02
$Err_{BS}*$	1.40	1.51	2.25	2.26	4.62	26.86	2.37	4.06
$Err_{BS}^{(1)}*$	0.86	0.87	0.88	0.87	1.70	1.70	0.99	1.00
$Err_{BSCV\times n}*$	10.59	11.29	49.60	49.57	41.96	1952.16	28.80	181.33
$Err_{BSCV\times 10}*$	0.86	0.88	0.88	0.88	1.70	1.72	0.97	0.99
$Err_{BSCV\times 5}*$	2.42	2.54	6.70	6.71	5.30	71.81	4.09	12.83
$Err_{BS}^{\times 2}*$	1.41	1.47	2.24	2.27	4.58	26.87	2.36	4.12
$Err_{RCV\times n}$	9.78	10.47	48.71	49.85	39.92	1920.13	27.89	208.41
$Err_{RCV\times 10}$	1.51	1.63	6.00	6.08	3.65	64.73	3.06	13.12
$Err_{RCV\times 5}$	2.83	3.04	12.69	13.02	5.77	157.98	5.38	29.92

9 Conclusions and future work

The contribution of this work is two-fold. First, we propose two new smoothed bootstrap schemes to clone data for better error estimation for supervised learning algorithms, and report on extensive simulations with both synthetic and real data. We show that in terms of the root mean squared error (RMSE), cloning improves the quality of the bootstrap-based estimators. Our argument in favor of RMSE as a measure of estimator quality is based on Markov's inequality, which bounds the probability of getting a large absolute error in an estimator. Our results show a high correlation of the improvement in RMSE with a reduction in the variance of the estimator. In particular, in 7 out of 8 experiments on the UCI data sets (Table 5), and in 12 out of 20 on synthetic data sets, .632+ is outperformed by its cloning-based counterpart. In general, while no estimator wins in all cases, in 22 of 28 cases, including 7 of 8 experiments on UCI data , one of the new estimators was significantly better in terms of RMSE than any of the old ones. Based on these observations, we recommend the novel cloning-based .632+* estimator. Cloned 5-fold CV was the best estimator for synthetic data, but showed inferior performance on UCI sets. We intend to investigate the properties of the data that affect the relative performance of these two estimators.

Second, we propose a suite of algorithms that can statistically clone real data from "complex" domains in which data points have several types of dependent attributes (continuous, integer, bounded and nominal). While our algorithms perform well on a number of UCI data sets, some limitations (such as the one illustrated in Figure 7) exist. We would like to overcome these limitations in future work.

We should emphasize that the use of statistical cloning techniques comes with increased computation. However, with the proposed choice of kernel, the increase in computation needed for cloning is within a constant factor relative to the ordinary bootstrap. Bootstrapped cross-validation involves $O(n^2)$ additional applications of the learning algorithm, which is expensive. However, in the spirit of work done recently to allow effective leave-one-out estimation "customized" for specific classifiers, such as SVMs [20], the design of an efficient cloning technique for particular classifiers is an interesting problem.

One of the advantages of the bootstrap over cross-validation is the ability to provide confidence intervals for the estimated value. We are working on expanding our system to include this capability.

Another area in which a study is needed is the performance of smoothed bootstrap and the factors that determine it. The current choice of smoothing parameters is based on results from density estimation. However, it was not proven that the optimal bandwidth for density estimation is also the best choice for the smoothed bootstrap error estimation.

Finally, and perhaps most importantly, an advance in theory of smoothed bootstrap may allow for a better understanding of the conditions that make it suitable for accuracy estimation of certain classifiers and domains, and eventually for a better choice of classifier for a given classification problem.

References

[1] J. Aitchison and C. G. G. Aitken. Multivariate binary discrimination by the kernel method. *Biometrika*, 63:413–420, 1976.

[2] Timothy L. Bailey and Charles Elkan. Estimating the accuracy of learned concepts. In *IJCAI*, pages 895–901, 1993.

[3] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. [http://www.ics.uci.edu/∼mlearn/MLRepository.html], 1998.

[4] A. Blum, A. Kalai, and J. Langford. Beating the hold-out: bounds for k-fold and progressive cross-validation. In *Proc. 12th Annu. Conf. on Comput. Learning Theory*, pages 203–208. ACM Press, New York, NY, 1999.

[5] A. C. Davison and D. V. Hinkley. *Bootstrap methods and their aplication.* Cambridge University Press, Cambridge, 1998.

[6] L. Devroye. *Non-Uniform Random Variate Generation.* Springer-Verlag, 1986.

[7] L. Devroye and L. Győfri. *Nonparametric Density Estimation: The L_1 View.* Wiley, New York, 1985.

[8] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.

[9] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification.* John Wiley & sons, New York, second edition, 2001.

[10] B. Efron. Bootstrap methods: another look at the jackknife. *Annals of Statistics*, 7:1–26, 1979.

[11] B. Efron. Estimating the error rate of a prediction rule: improvements on cross-validation. *Journal of the American Statistical Association*, 78:316–331, 1983.

[12] B. Efron. Jackknife-after-bootstrap standard errors and influence functions *(with discussion). Journal of Royal Statistical Society*, 54(1):83–127, 1992.

[13] B. Efron and R. Tibshirani. Improvements on cross-validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560, June 1997.

[14] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap.* Chapman and Hall, London, UK, 1993.

[15] C. El-Nouty and A. Guillou. On the smoothed bootstrap. *Journal of Statistical Planning and Inference*, 83(1):203–220, January 2000.

[16] K. Fukunaga. *Introduction to Statistical Pattern Recognition, Second Edition.* Academic Press, Boston, MA, 1990.

[17] R. Guerra, A. M. Polansky, and W. R. Schucany. Smoothed bootstrap confidence intervals with discrete data. *Computational Statistics and Data Analysis*, 26:163–176, 2000.

[18] P. Hall, T. J. DiCiccio, and J. P. Romano. On smoothing and the bootstrap. *The Annals of Statistics*, 17(2):692–704, 1989.

[19] J. Hermans, J.D.F. Habbema, T.K.D. Kasanmoentalib, and J.W. Raatgeven. Manual for the ALLOC80 discriminant analysis program. Leiden, Netherlands, 1982.

[20] T. Joachims. Estimating the generalization performance of an SVM efficiently. In *Proceedings of the 17th International Conf. on Machine Learning*, pages 431–438. Morgan Kaufmann, San Francisco, CA, 2000.

[21] M. C. Jones, J. S. Marron, and S. J. Sheather. A brief survey of bandwidth selection for density estimation. *Journal of the American Statistical Association*, 91(433):401–407, ???? 1996.

[22] M. Kearns and D. Ron. Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. *Neural Computation*, 11(6):1427–1453, 1999.

[23] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint*

Conference on Artificial Intelligence, volume 2, pages 1137–1145, San Mateo, August 1995. Morgan Kaufmann.

[24] Ron Kohavi and David H. Wolpert. Bias plus variance decomposition for zero-one loss functions. In Lorenza Saitta, editor, *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 275–283. Morgan Kaufmann, 1996.

[25] G. J. McLachlan. *Discriminant Analysis and Statistical Pattern Recognition*, chapter 10, pages 337–377. Wiley, New York, 1992.

[26] D. Michie, D. Spiegelhalter, and C. Taylor, editors. *Machine learning, neural and statistical classification.* Ellis Horwood series in Artificial Intelligence. Ellis Horwood, New York, 1994.

[27] A. Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGrow Hill, New York, 3rd edition, 1991.

[28] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipies in C: The art of Scientific Programming.* Cambridge University Press, Cambridge, England, second edition, 1992.

[29] S. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–327, 1997.

[30] D. W. Scott. *Multivariate Density Estimation. Theory, Practice and Visualization.* Wiley, New York, 1992.

[31] J. Shao and D. Tu. *The Jackknife and Bootstrap.* Springer-Verlag, New York, 1995.

[32] B. W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman & Hall, London, 1986.

[33] B. W. Silverman and G. A. Young. The bootstrap: to smooth or not to smooth? *Biometrika*, 74(3):469–479, 1987.

[34] D. E. Stewart and Z. Leyk. *Meschach: Matrix Computations in C.* Centre for Mathematics and its Applications, School of Mathematical Sciences, Australian National University, Canberra, ACT 0200, Australia, 1994.

[35] V. N. Vapnik. *Estimation of Dependencies Based on Empirical Data.* Springer-Verlag, Berlin, 1982.

[36] M. P. Wand and M. C. Jones. *Kernel Smoothing.* Chapman & Hall, London, 1995.

A Detailed results of simulation experiments

A.1 Synthetic data sets

The experiments are described in Section 6. For each estimator, we report the mean value of the prediction (first column) and the standard deviation (second column) over the trials. Third column gives the RMSE. First row in each table gives the value of the true error *Err* estimated on a large validation set.

Description of the data sets id given in Table 1. Here we repeat the values of the parameters n (size of the data set), d (dimensionality0 and the number of trials in the simulation with each data set.

For data sets with less than 20 points, we did not perform 10-fold cross validation. The corresponding entries in the tables are marked NA (Not Available).

Table 8: Syntehtic data set 1, $n = 14$, $d = 5$, 200 trials.

LDF

Estimator	Mean	STD	RMSE
Err	0.262	0.0710	0.0000
$\overline{\text{err}}$	0.093	0.0832	0.1917
$Err_{.632}$	0.268	0.0706	0.0826
$Err_{.632+}$	0.327	0.0778	0.1096
$Err_{CV\times n}$	0.261	0.1272	0.1371
$Err_{CV\times 10}$	NA	NA	NA
$Err_{CV\times 5}$	0.293	0.1261	0.1349
Err_{BS}	0.254	0.0662	0.0776
$Err_{BS}^{(1)}$	0.371	0.0714	0.1378
$Err_{BSCV\times n}$	0.273	0.0655	0.0767
$Err_{BSCV\times 10}$	NA	NA	NA
$Err_{BSCV\times 5}$	0.311	0.0585	0.0874
$Err_{.632}*$	0.265	0.0758	0.0873
$Err_{.632+}*$	0.320	0.0846	0.1135
$Err_{BS}*$	0.230	0.0661	0.0834
$Err_{BS}^{(1)}*$	0.365	0.0797	0.1383
$Err_{BSCV\times n}*$	0.267	0.0626	0.0767
$Err_{BSCV\times 10}*$	NA	NA	NA
$Err_{BSCV\times 5}*$	0.299	0.0573	0.0823

1-NN

Estimator	Mean	STD	RMSE
Err	0.278	0.0517	0.0000
$\overline{\text{err}}$	0.000	0.0000	0.2832
$Err_{.632}$	0.199	0.0812	0.1053
$Err_{.632+}$	0.256	0.1139	0.0999
$Err_{CV\times n}$	0.296	0.1583	0.1450
$Err_{CV\times 10}$	NA	NA	NA
$Err_{CV\times 5}$	0.302	0.1587	0.1483
Err_{BS}	0.111	0.0457	0.1737
$Err_{BS}^{(1)}$	0.315	0.1284	0.1161
$Err_{BSCV\times n}$	0.122	0.0492	0.1644
$Err_{BSCV\times 10}$	NA	NA	NA
$Err_{BSCV\times 5}$	0.141	0.0542	0.1467
$Err_{.632}*$	0.216	0.0707	0.0863
$Err_{.632+}*$	0.279	0.0990	0.0877
$Err_{BS}*$	0.146	0.0594	0.1414
$Err_{BS}^{(1)}*$	0.343	0.1119	0.1133
$Err_{BSCV\times n}*$	0.178	0.0604	0.1136
$Err_{BSCV\times 10}*$	NA	NA	NA
$Err_{BSCV\times 5}*$	0.192	0.0617	0.1016

3-NN

Estimator	Mean	STD	RMSE
Err	0.246	0.0451	0.0000
$\overline{\text{err}}$	0.124	0.0945	0.1511
$Err_{.632}$	0.256	0.1016	0.0901
$Err_{.632+}$	0.288	0.1106	0.1068
$Err_{CV\times n}$	0.280	0.1580	0.1460
$Err_{CV\times 10}$	NA	NA	NA
$Err_{CV\times 5}$	0.288	0.1641	0.1555
Err_{BS}	0.199	0.0745	0.0815
$Err_{BS}^{(1)}$	0.333	0.1134	0.1313
$Err_{BSCV\times n}$	0.208	0.0762	0.0773
$Err_{BSCV\times 10}$	NA	NA	NA
$Err_{BSCV\times 5}$	0.228	0.0776	0.0697
$Err_{.632}*$	0.268	0.0947	0.0873
$Err_{.632+}*$	0.306	0.1011	0.1088
$Err_{BS}*$	0.188	0.0680	0.0834
$Err_{BS}^{(1)}*$	0.352	0.1037	0.1402
$Err_{BSCV\times n}*$	0.218	0.0655	0.0662
$Err_{BSCV\times 10}*$	NA	NA	NA
$Err_{BSCV\times 5}*$	0.239	0.0651	0.0598

SVM

Estimator	Mean	STD	RMSE
Err	0.237	0.0423	0.0000
$\overline{\text{err}}$	0.039	0.0521	0.2058
$Err_{.632}$	0.238	0.0908	0.0869
$Err_{.632+}$	0.295	0.1104	0.1225
$Err_{CV\times n}$	0.296	0.1803	0.1841
$Err_{CV\times 10}$	NA	NA	NA
$Err_{CV\times 5}$	0.335	0.1952	0.2128
Err_{BS}	0.146	0.0626	0.1108
$Err_{BS}^{(1)}$	0.354	0.1238	0.1665
$Err_{BSCV\times n}$	0.158	0.0636	0.1018
$Err_{BSCV\times 10}$	NA	NA	NA
$Err_{BSCV\times 5}$	0.184	0.0651	0.0839
$Err_{.632}*$	0.265	0.0804	0.0834
$Err_{.632+}*$	0.329	0.0890	0.1292
$Err_{BS}*$	0.174	0.0683	0.0921
$Err_{BS}^{(1)}*$	0.397	0.1088	0.1920
$Err_{BSCV\times n}*$	0.219	0.0648	0.0692
$Err_{BSCV\times 10}*$	NA	NA	NA
$Err_{BSCV\times 5}*$	0.247	0.0632	0.0671

Table 9: Syntehtic data set 2, $n = 14$, $d = 5$, 200 trials.

LDF

Estimator	Mean	STD	RMSE
Err	0.500	0.0031	0.0000
$\overline{\text{err}}$	0.231	0.1166	0.2930
$Err_{.632}$	0.406	0.0819	0.1247
$Err_{.632+}$	0.446	0.0597	0.0804
$Err_{CV\times n}$	0.493	0.1459	0.1452
$Err_{CV\times 10}$	-1.000	0.0000	1.5002
$Err_{CV\times 5}$	0.494	0.1511	0.1505
Err_{BS}	0.343	0.0573	0.1668
$Err_{BS}^{(1)}$	0.507	0.0705	0.0703
$Err_{BSCV\times n}$	0.355	0.0567	0.1557
$Err_{BSCV\times 10}$	-1.000	0.0000	1.5002
$Err_{BSCV\times 5}$	0.381	0.0487	0.1283
$Err_{.632}*$	0.408	0.0839	0.1240
$Err_{.632+}*$	0.442	0.0623	0.0844
$Err_{BS}*$	0.343	0.0613	0.1683
$Err_{BS}^{(1)}*$	0.511	0.0729	0.0731
$Err_{BSCV\times n}*$	0.371	0.0567	0.1406
$Err_{BSCV\times 10}*$	-1.000	0.0000	1.5002
$Err_{BSCV\times 5}*$	0.391	0.0491	0.1198

1-NN

Estimator	Mean	STD	RMSE
Err	0.500	0.0035	0.0000
$\overline{\text{err}}$	0.000	0.0000	0.5000
$Err_{.632}$	0.341	0.0664	0.1717
$Err_{.632+}$	0.384	0.0618	0.1315
$Err_{CV\times n}$	0.545	0.1518	0.1575
$Err_{CV\times 10}$	-1.000	0.0000	1.4999
$Err_{CV\times 5}$	0.537	0.1546	0.1582
Err_{BS}	0.192	0.0379	0.3101
$Err_{BS}^{(1)}$	0.540	0.1051	0.1118
$Err_{BSCV\times n}$	0.205	0.0405	0.2973
$Err_{BSCV\times 10}$	-1.000	0.0000	1.4999
$Err_{BSCV\times 5}$	0.234	0.0430	0.2697
$Err_{.632}*$	0.338	0.0461	0.1681
$Err_{.632+}*$	0.376	0.0523	0.1345
$Err_{BS}*$	0.247	0.0431	0.2561
$Err_{BS}^{(1)}*$	0.535	0.0729	0.0803
$Err_{BSCV\times n}*$	0.280	0.0463	0.2244
$Err_{BSCV\times 10}*$	-1.000	0.0000	1.4999
$Err_{BSCV\times 5}*$	0.297	0.0452	0.2082

3-NN

Estimator	Mean	STD	RMSE
Err	0.500	0.0035	0.0000
$\overline{\text{err}}$	0.261	0.1017	0.2598
$Err_{.632}$	0.439	0.0819	0.1015
$Err_{.632+}$	0.459	0.0660	0.0772
$Err_{CV\times n}$	0.550	0.1506	0.1576
$Err_{CV\times 10}$	-1.000	0.0000	1.5001
$Err_{CV\times 5}$	0.548	0.1660	0.1719
Err_{BS}	0.338	0.0578	0.1723
$Err_{BS}^{(1)}$	0.543	0.0807	0.0908
$Err_{BSCV\times n}$	0.347	0.0580	0.1636
$Err_{BSCV\times 10}$	-1.000	0.0000	1.5001
$Err_{BSCV\times 5}$	0.370	0.0572	0.1415
$Err_{.632}*$	0.440	0.0704	0.0918
$Err_{.632+}*$	0.459	0.0526	0.0668
$Err_{BS}*$	0.311	0.0456	0.1939
$Err_{BS}^{(1)}*$	0.545	0.0643	0.0779
$Err_{BSCV\times n}*$	0.337	0.0463	0.1699
$Err_{BSCV\times 10}*$	-1.000	0.0000	1.5001
$Err_{BSCV\times 5}*$	0.357	0.0434	0.1498

SVM

Estimator	Mean	STD	RMSE
Err	0.500	0.0033	0.0000
$\overline{\text{err}}$	0.092	0.0707	0.4144
$Err_{.632}$	0.396	0.0692	0.1255
$Err_{.632+}$	0.417	0.0562	0.1003
$Err_{CV\times n}$	0.653	0.2258	0.2714
$Err_{CV\times 10}$	-1.000	0.0000	1.5004
$Err_{CV\times 5}$	0.638	0.1824	0.2278
Err_{BS}	0.259	0.0514	0.2468
$Err_{BS}^{(1)}$	0.572	0.0891	0.1139
$Err_{BSCV\times n}$	0.272	0.0506	0.2342
$Err_{BSCV\times 10}$	-1.000	0.0000	1.5004
$Err_{BSCV\times 5}$	0.299	0.0487	0.2068
$Err_{.632}*$	0.400	0.0563	0.1155
$Err_{.632+}*$	0.412	0.0472	0.1005
$Err_{BS}*$	0.297	0.0494	0.2096
$Err_{BS}^{(1)}*$	0.578	0.0681	0.1033
$Err_{BSCV\times n}*$	0.330	0.0474	0.1772
$Err_{BSCV\times 10}*$	-1.000	0.0000	1.5004
$Err_{BSCV\times 5}*$	0.351	0.0435	0.1558

Table 10: Syntehtic data set 3, $n = 20$, $d = 2$, 200 trials.

LDF

Estimator	Mean	STD	RMSE
Err	0.350	0.0504	0.0000
$\overline{\mathrm{err}}$	0.289	0.0953	0.1138
$Err_{.632}$	0.348	0.1019	0.1010
$Err_{.632+}$	0.356	0.1020	0.1024
$Err_{CV\times n}$	0.349	0.1151	0.1117
$Err_{CV\times 10}$	0.350	0.1218	0.1193
$Err_{CV\times 5}$	0.359	0.1235	0.1212
Err_{BS}	0.313	0.0910	0.0996
$Err_{BS}^{(1)}$	0.382	0.1100	0.1126
$Err_{BSCV\times n}$	0.314	0.0914	0.0997
$Err_{BSCV\times 10}$	0.316	0.0910	0.0990
$Err_{BSCV\times 5}$	0.320	0.0896	0.0965
$Err_{.632}*$	0.354	0.0992	0.0988
$Err_{.632+}*$	0.364	0.0991	0.1009
$Err_{BS}*$	0.322	0.0900	0.0953
$Err_{BS}^{(1)}*$	0.391	0.1061	0.1127
$Err_{BSCV\times n}*$	0.345	0.0800	0.0819
$Err_{BSCV\times 10}*$	0.350	0.0794	0.0813
$Err_{BSCV\times 5}*$	0.350	0.0794	0.0813

1-NN

Estimator	Mean	STD	RMSE
Err	0.414	0.0422	0.0000
$\overline{\mathrm{err}}$	0.000	0.0000	0.4158
$Err_{.632}$	0.279	0.0711	0.1477
$Err_{.632+}$	0.354	0.0834	0.1002
$Err_{CV\times n}$	0.437	0.1387	0.1298
$Err_{CV\times 10}$	0.440	0.1433	0.1347
$Err_{CV\times 5}$	0.441	0.1461	0.1393
Err_{BS}	0.158	0.0403	0.2589
$Err_{BS}^{(1)}$	0.442	0.1125	0.1021
$Err_{BSCV\times n}$	0.167	0.0433	0.2503
$Err_{BSCV\times 10}$	0.176	0.0448	0.2415
$Err_{BSCV\times 5}$	0.196	0.0490	0.2222
$Err_{.632}*$	0.289	0.0547	0.1336
$Err_{.632+}*$	0.364	0.0715	0.0913
$Err_{BS}*$	0.339	0.0703	0.0954
$Err_{BS}^{(1)}*$	0.457	0.0866	0.0840
$Err_{BSCV\times n}*$	0.358	0.0653	0.0772
$Err_{BSCV\times 10}*$	0.363	0.0646	0.0737
$Err_{BSCV\times 5}*$	0.363	0.0646	0.0737

3-NN

Estimator	Mean	STD	RMSE
Err	0.392	0.0514	0.0000
$\overline{\mathrm{err}}$	0.211	0.0811	0.2010
$Err_{.632}$	0.359	0.0889	0.0887
$Err_{.632+}$	0.391	0.0848	0.0816
$Err_{CV\times n}$	0.426	0.1452	0.1317
$Err_{CV\times 10}$	0.427	0.1482	0.1354
$Err_{CV\times 5}$	0.442	0.1561	0.1485
Err_{BS}	0.277	0.0657	0.1316
$Err_{BS}^{(1)}$	0.444	0.1031	0.1041
$Err_{BSCV\times n}$	0.284	0.0684	0.1264
$Err_{BSCV\times 10}$	0.291	0.0695	0.1205
$Err_{BSCV\times 5}$	0.307	0.0704	0.1076
$Err_{.632}*$	0.367	0.0818	0.0804
$Err_{.632+}*$	0.400	0.0775	0.0779
$Err_{BS}*$	0.331	0.0735	0.0906
$Err_{BS}^{(1)}*$	0.457	0.0947	0.1053
$Err_{BSCV\times n}*$	0.353	0.0694	0.0741
$Err_{BSCV\times 10}*$	0.361	0.0679	0.0689
$Err_{BSCV\times 5}*$	0.361	0.0679	0.0689

SVM

Estimator	Mean	STD	RMSE
Err	0.356	0.0492	0.0000
$\overline{\mathrm{err}}$	0.248	0.0880	0.1406
$Err_{.632}$	0.377	0.1054	0.1055
$Err_{.632+}$	0.402	0.1036	0.1133
$Err_{CV\times n}$	0.446	0.2273	0.2330
$Err_{CV\times 10}$	0.450	0.1997	0.2096
$Err_{CV\times 5}$	0.448	0.1913	0.2044
Err_{BS}	0.320	0.0886	0.0959
$Err_{BS}^{(1)}$	0.452	0.1208	0.1512
$Err_{BSCV\times n}$	0.326	0.0902	0.0949
$Err_{BSCV\times 10}$	0.330	0.0891	0.0926
$Err_{BSCV\times 5}$	0.339	0.0874	0.0889
$Err_{.632}*$	0.390	0.0963	0.1019
$Err_{.632+}*$	0.417	0.0910	0.1109
$Err_{BS}*$	0.340	0.0825	0.0858
$Err_{BS}^{(1)}*$	0.472	0.1072	0.1571
$Err_{BSCV\times n}*$	0.356	0.0756	0.0787
$Err_{BSCV\times 10}*$	0.368	0.0724	0.0776
$Err_{BSCV\times 5}*$	0.368	0.0724	0.0776

Table 11: Syntehtic data set 4, $n = 20$, $d = 2$, 200 trials.

LDF

Estimator	Mean	STD	RMSE
Err	0.501	0.0031	0.0000
$\overline{\mathrm{err}}$	0.394	0.0874	0.1384
$Err_{.632}$	0.462	0.0902	0.0983
$Err_{.632+}$	0.468	0.0860	0.0924
$Err_{CV\times n}$	0.487	0.1316	0.1320
$Err_{CV\times 10}$	0.496	0.1423	0.1423
$Err_{CV\times 5}$	0.502	0.1423	0.1424
Err_{BS}	0.405	0.0715	0.1198
$Err_{BS}^{(1)}$	0.502	0.0969	0.0972
$Err_{BSCV\times n}$	0.404	0.0710	0.1200
$Err_{BSCV\times 10}$	0.406	0.0709	0.1184
$Err_{BSCV\times 5}$	0.408	0.0689	0.1157
$Err_{.632}*$	0.464	0.0828	0.0909
$Err_{.632+}*$	0.471	0.0771	0.0828
$Err_{BS}*$	0.413	0.0678	0.1111
$Err_{BS}^{(1)}*$	0.505	0.0848	0.0852
$Err_{BSCV\times n}*$	0.423	0.0604	0.0984
$Err_{BSCV\times 10}*$	0.427	0.0580	0.0941
$Err_{BSCV\times 5}*$	0.427	0.0580	0.0941

1-NN

Estimator	Mean	STD	RMSE
Err	0.500	0.0036	0.0000
$\overline{\mathrm{err}}$	0.000	0.0000	0.5005
$Err_{.632}$	0.337	0.0635	0.1757
$Err_{.632+}$	0.376	0.0544	0.1359
$Err_{CV\times n}$	0.538	0.1352	0.1407
$Err_{CV\times 10}$	0.536	0.1398	0.1445
$Err_{CV\times 5}$	0.537	0.1266	0.1322
Err_{BS}	0.191	0.0363	0.3119
$Err_{BS}^{(1)}$	0.533	0.1005	0.1061
$Err_{BSCV\times n}$	0.202	0.0380	0.3010
$Err_{BSCV\times 10}$	0.213	0.0393	0.2907
$Err_{BSCV\times 5}$	0.234	0.0414	0.2699
$Err_{.632}*$	0.333	0.0361	0.1711
$Err_{.632+}*$	0.376	0.0437	0.1323
$Err_{BS}*$	0.398	0.0519	0.1147
$Err_{BS}^{(1)}*$	0.527	0.0572	0.0633
$Err_{BSCV\times n}*$	0.408	0.0480	0.1043
$Err_{BSCV\times 10}*$	0.413	0.0473	0.0998
$Err_{BSCV\times 5}*$	0.413	0.0473	0.0998

3-NN

Estimator	Mean	STD	RMSE
Err	0.500	0.0033	0.0000
$\overline{\mathrm{err}}$	0.257	0.0880	0.2587
$Err_{.632}$	0.432	0.0803	0.1058
$Err_{.632+}$	0.457	0.0622	0.0763
$Err_{CV\times n}$	0.529	0.1517	0.1551
$Err_{CV\times 10}$	0.535	0.1532	0.1577
$Err_{CV\times 5}$	0.545	0.1416	0.1491
Err_{BS}	0.333	0.0563	0.1758
$Err_{BS}^{(1)}$	0.534	0.0823	0.0897
$Err_{BSCV\times n}$	0.341	0.0580	0.1694
$Err_{BSCV\times 10}$	0.349	0.0577	0.1615
$Err_{BSCV\times 5}$	0.366	0.0565	0.1455
$Err_{.632}*$	0.433	0.0655	0.0935
$Err_{.632+}*$	0.458	0.0457	0.0625
$Err_{BS}*$	0.395	0.0524	0.1170
$Err_{BS}^{(1)}*$	0.537	0.0625	0.0732
$Err_{BSCV\times n}*$	0.411	0.0503	0.1020
$Err_{BSCV\times 10}*$	0.419	0.0478	0.0944
$Err_{BSCV\times 5}*$	0.419	0.0478	0.0944

SVM

Estimator	Mean	STD	RMSE
Err	0.500	0.0033	0.0000
$\overline{\mathrm{err}}$	0.321	0.0747	0.1943
$Err_{.632}$	0.474	0.0726	0.0777
$Err_{.632+}$	0.484	0.0614	0.0641
$Err_{CV\times n}$	0.727	0.2502	0.3375
$Err_{CV\times 10}$	0.650	0.1781	0.2330
$Err_{CV\times 5}$	0.622	0.1491	0.1928
Err_{BS}	0.403	0.0634	0.1165
$Err_{BS}^{(1)}$	0.563	0.0793	0.1020
$Err_{BSCV\times n}$	0.406	0.0637	0.1142
$Err_{BSCV\times 10}$	0.408	0.0625	0.1115
$Err_{BSCV\times 5}$	0.416	0.0584	0.1033
$Err_{.632}*$	0.475	0.0628	0.0681
$Err_{.632+}*$	0.485	0.0511	0.0537
$Err_{BS}*$	0.413	0.0552	0.1032
$Err_{BS}^{(1)}*$	0.565	0.0642	0.0914
$Err_{BSCV\times n}*$	0.420	0.0528	0.0967
$Err_{BSCV\times 10}*$	0.429	0.0488	0.0866
$Err_{BSCV\times 5}*$	0.429	0.0488	0.0866

Table 12: Syntehtic data set 5, $n = 100$, $d = 10$, 50 trials.

LDF

Estimator	Mean	STD	RMSE
Err	0.018	0.0044	0.0000
$\overline{\mathrm{err}}$	0.000	0.0000	0.0187
$Err_{.632}$	0.008	0.0076	0.0147
$Err_{.632+}$	0.009	0.0078	0.0147
$Err_{CV\times n}$	0.007	0.0086	0.0159
$Err_{CV\times 10}$	0.008	0.0100	0.0161
$Err_{CV\times 5}$	0.007	0.0096	0.0161
Err_{BS}	0.010	0.0105	0.0159
$Err_{BS}^{(1)}$	0.013	0.0121	0.0160
$Err_{BSCV\times n}$	0.010	0.0103	0.0157
$Err_{BSCV\times 10}$	0.013	0.0118	0.0159
$Err_{BSCV\times 5}$	0.021	0.0161	0.0196
$Err_{.632}*$	0.007	0.0064	0.0148
$Err_{.632+}*$	0.007	0.0065	0.0148
$Err_{BS}*$	0.006	0.0064	0.0158
$Err_{BS}^{(1)}*$	0.011	0.0101	0.0150
$Err_{BSCV\times n}*$	0.009	0.0084	0.0148
$Err_{BSCV\times 10}*$	0.016	0.0126	0.0159
$Err_{BSCV\times 5}*$	0.016	0.0126	0.0159

1-NN

Estimator	Mean	STD	RMSE
Err	0.022	0.0020	0.0000
$\overline{\mathrm{err}}$	0.000	0.0000	0.0222
$Err_{.632}$	0.009	0.0063	0.0155
$Err_{.632+}$	0.009	0.0064	0.0155
$Err_{CV\times n}$	0.009	0.0087	0.0164
$Err_{CV\times 10}$	0.010	0.0095	0.0161
$Err_{CV\times 5}$	0.011	0.0099	0.0161
Err_{BS}	0.005	0.0037	0.0179
$Err_{BS}^{(1)}$	0.014	0.0099	0.0141
$Err_{BSCV\times n}$	0.005	0.0038	0.0179
$Err_{BSCV\times 10}$	0.006	0.0042	0.0173
$Err_{BSCV\times 5}$	0.007	0.0045	0.0166
$Err_{.632}*$	0.010	0.0059	0.0141
$Err_{.632+}*$	0.010	0.0061	0.0141
$Err_{BS}*$	0.005	0.0036	0.0174
$Err_{BS}^{(1)}*$	0.016	0.0094	0.0123
$Err_{BSCV\times n}*$	0.008	0.0040	0.0149
$Err_{BSCV\times 10}*$	0.010	0.0047	0.0132
$Err_{BSCV\times 5}*$	0.010	0.0047	0.0132

3-NN

Estimator	Mean	STD	RMSE
Err	0.027	0.0024	0.0000
$\overline{\mathrm{err}}$	0.009	0.0087	0.0206
$Err_{.632}$	0.014	0.0096	0.0171
$Err_{.632+}$	0.014	0.0097	0.0171
$Err_{CV\times n}$	0.014	0.0121	0.0189
$Err_{CV\times 10}$	0.015	0.0124	0.0186
$Err_{CV\times 5}$	0.016	0.0135	0.0182
Err_{BS}	0.011	0.0072	0.0186
$Err_{BS}^{(1)}$	0.017	0.0106	0.0156
$Err_{BSCV\times n}$	0.011	0.0073	0.0188
$Err_{BSCV\times 10}$	0.011	0.0076	0.0182
$Err_{BSCV\times 5}$	0.012	0.0079	0.0175
$Err_{.632}*$	0.015	0.0094	0.0162
$Err_{.632+}*$	0.015	0.0094	0.0162
$Err_{BS}*$	0.010	0.0067	0.0187
$Err_{BS}^{(1)}*$	0.018	0.0106	0.0147
$Err_{BSCV\times n}*$	0.014	0.0064	0.0151
$Err_{BSCV\times 10}*$	0.017	0.0073	0.0130
$Err_{BSCV\times 5}*$	0.017	0.0073	0.0130

SVM

Estimator	Mean	STD	RMSE
Err	0.004	0.0008	0.0000
$\overline{\mathrm{err}}$	0.000	0.0000	0.0037
$Err_{.632}$	0.001	0.0018	0.0033
$Err_{.632+}$	0.001	0.0018	0.0033
$Err_{CV\times n}$	0.000	0.0000	0.0037
$Err_{CV\times 10}$	0.000	0.0019	0.0038
$Err_{CV\times 5}$	0.000	0.0019	0.0039
Err_{BS}	0.001	0.0010	0.0033
$Err_{BS}^{(1)}$	0.002	0.0029	0.0037
$Err_{BSCV\times n}$	0.001	0.0010	0.0033
$Err_{BSCV\times 10}$	0.001	0.0013	0.0032
$Err_{BSCV\times 5}$	0.001	0.0015	0.0033
$Err_{.632}*$	0.001	0.0014	0.0033
$Err_{.632+}*$	0.001	0.0014	0.0033
$Err_{BS}*$	0.001	0.0008	0.0034
$Err_{BS}^{(1)}*$	0.002	0.0023	0.0035
$Err_{BSCV\times n}*$	0.002	0.0018	0.0029
$Err_{BSCV\times 10}*$	0.003	0.0023	0.0030
$Err_{BSCV\times 5}*$	0.003	0.0023	0.0030

A.2 Real data sets from UCI repository

Methodology of the simulation is described in section 8. Table 3 contains description of the data sets. In addition to the estimators tested on dynthetic data sets, repeated cross-validation was included in the study for UCI data.

Table 13: Breast Cancer , $n = 36$, $d = 9$, 150 trials.

1-NN

Estimator	Mean	STD	RMSE
Err	0.052	0.0189	0.0000
$\overline{\text{err}}$	0.000	0.0000	0.0553
$Err_{.632}$	0.035	0.0255	0.0271
$Err_{.632+}$	0.038	0.0289	0.0275
$Err_{CV \times n}$	0.052	0.0445	0.0383
$Err_{CV \times 10}$	0.055	0.0459	0.0389
$Err_{CV \times 5}$	0.053	0.0449	0.0388
Err_{BS}	0.020	0.0146	0.0354
$Err^{(1)}_{BS}$	0.056	0.0404	0.0336
$Err_{BSCV \times n}$	0.021	0.0154	0.0355
$Err_{BSCV \times 10}$	0.023	0.0166	0.0340
$Err_{BSCV \times 5}$	0.025	0.0177	0.0320
$Err_{.632}*$	0.033	0.0232	0.0272
$Err_{.632+}*$	0.036	0.0259	0.0269
$Err_{BS}*$	0.019	0.0134	0.0369
$Err^{(1)}_{BS}*$	0.053	0.0367	0.0303
$Err_{BSCV \times n}*$	0.020	0.0142	0.0361
$Err_{BSCV \times 10}*$	0.024	0.0166	0.0329
$Err_{BSCV \times 5}*$	0.024	0.0166	0.0329
$Err_{RCV \times n}$	0.052	0.0451	0.0386
$Err_{RCV \times 10}$	0.054	0.0426	0.0358
$Err_{RCV \times 5}$	0.053	0.0439	0.0374

3-NN

Estimator	Mean	STD	RMSE
Err	0.045	0.0116	0.0000
$\overline{\text{err}}$	0.029	0.0250	0.0321
$Err_{.632}$	0.046	0.0298	0.0303
$Err_{.632+}$	0.047	0.0303	0.0307
$Err_{CV \times n}$	0.046	0.0346	0.0344
$Err_{CV \times 10}$	0.047	0.0343	0.0334
$Err_{CV \times 5}$	0.048	0.0354	0.0353
Err_{BS}	0.036	0.0234	0.0259
$Err^{(1)}_{BS}$	0.056	0.0347	0.0360
$Err_{BSCV \times n}$	0.037	0.0234	0.0260
$Err_{BSCV \times 10}$	0.038	0.0236	0.0257
$Err_{BSCV \times 5}$	0.040	0.0243	0.0256
$Err_{.632}*$	0.044	0.0287	0.0295
$Err_{.632+}*$	0.045	0.0291	0.0297
$Err_{BS}*$	0.033	0.0211	0.0256
$Err^{(1)}_{BS}*$	0.053	0.0330	0.0339
$Err_{BSCV \times n}*$	0.033	0.0214	0.0255
$Err_{BSCV \times 10}*$	0.037	0.0225	0.0248
$Err_{BSCV \times 5}*$	0.037	0.0225	0.0248
$Err_{RCV \times n}$	0.046	0.0341	0.0338
$Err_{RCV \times 10}$	0.047	0.0323	0.0322
$Err_{RCV \times 5}$	0.046	0.0331	0.0328

SVM, linear kernel

Estimator	Mean	STD	RMSE
Err	0.040	0.0085	0.0000
$\overline{\text{err}}$	0.022	0.0219	0.0300
$Err_{.632}$	0.037	0.0251	0.0273
$Err_{.632+}$	0.037	0.0254	0.0276
$Err_{CV \times n}$	0.039	0.0286	0.0306
$Err_{CV \times 10}$	0.040	0.0315	0.0333
$Err_{CV \times 5}$	0.041	0.0306	0.0323
Err_{BS}	0.028	0.0206	0.0259
$Err^{(1)}_{BS}$	0.045	0.0290	0.0313
$Err_{BSCV \times n}$	0.029	0.0207	0.0258
$Err_{BSCV \times 10}$	0.030	0.0209	0.0257
$Err_{BSCV \times 5}$	0.031	0.0213	0.0255
$Err_{.632}*$	0.036	0.0246	0.0269
$Err_{.632+}*$	0.036	0.0249	0.0271
$Err_{BS}*$	0.027	0.0204	0.0259
$Err^{(1)}_{BS}*$	0.044	0.0282	0.0302
$Err_{BSCV \times n}*$	0.026	0.0196	0.0259
$Err_{BSCV \times 10}*$	0.028	0.0203	0.0254
$Err_{BSCV \times 5}*$	0.028	0.0203	0.0254
$Err_{RCV \times n}$	0.038	0.0286	0.0306
$Err_{RCV \times 10}$	0.041	0.0279	0.0300
$Err_{RCV \times 5}$	0.039	0.0282	0.0301

SVM, RBFkernel

Estimator	Mean	STD	RMSE
Err	0.040	0.0092	0.0000
$\overline{\text{err}}$	0.000	0.0000	0.0409
$Err_{.632}$	0.028	0.0209	0.0222
$Err_{.632+}$	0.029	0.0230	0.0230
$Err_{CV \times n}$	0.043	0.0359	0.0341
$Err_{CV \times 10}$	0.043	0.0363	0.0341
$Err_{CV \times 5}$	0.044	0.0385	0.0356
Err_{BS}	0.016	0.0120	0.0266
$Err^{(1)}_{BS}$	0.044	0.0331	0.0303
$Err_{BSCV \times n}$	0.016	0.0126	0.0266
$Err_{BSCV \times 10}$	0.018	0.0135	0.0257
$Err_{BSCV \times 5}$	0.020	0.0152	0.0243
$Err_{.632}*$	0.027	0.0199	0.0220
$Err_{.632+}*$	0.029	0.0217	0.0225
$Err_{BS}*$	0.015	0.0118	0.0269
$Err^{(1)}_{BS}*$	0.043	0.0315	0.0287
$Err_{BSCV \times n}*$	0.016	0.0126	0.0266
$Err_{BSCV \times 10}*$	0.019	0.0146	0.0247
$Err_{BSCV \times 5}*$	0.019	0.0146	0.0247
$Err_{RCV \times n}$	0.043	0.0359	0.0341
$Err_{RCV \times 10}$	0.043	0.0340	0.0315
$Err_{RCV \times 5}$	0.043	0.0351	0.0330

Table 14: Vehicle, $n = 100$, $d = 18$, 100 trials

1-NN

Estimator	Mean	STD	RMSE
Err	0.438	0.0180	0.0000
$\overline{\mathrm{err}}$	0.000	0.0000	0.4387
$Err_{.632}$	0.293	0.0273	0.1488
$Err_{.632+}$	0.381	0.0460	0.0760
$Err_{CV\times n}$	0.434	0.0567	0.0603
$Err_{CV\times 10}$	0.440	0.0557	0.0583
$Err_{CV\times 5}$	0.445	0.0587	0.0620
Err_{BS}	0.170	0.0159	0.2696
$Err^{(1)}_{BS}$	0.464	0.0431	0.0537
$Err_{BSCV\times n}$	0.172	0.0166	0.2673
$Err_{BSCV\times 10}$	0.190	0.0175	0.2493
$Err_{BSCV\times 5}$	0.212	0.0188	0.2279
$Err_{.632}*$	0.330	0.0203	0.1117
$Err_{.632+}*$	0.445	0.0365	0.0417
$Err_{BS}*$	0.385	0.0298	0.0644
$Err^{(1)}_{BS}*$	0.522	0.0321	0.0919
$Err_{BSCV\times n}*$	0.425	0.0278	0.0355
$Err_{BSCV\times 10}*$	0.447	0.0278	0.0338
$Err_{BSCV\times 5}*$	0.447	0.0278	0.0338
$Err_{RCV\times n}$	0.434	0.0568	0.0605
$Err_{RCV\times 10}$	0.448	0.0495	0.0542
$Err_{RCV\times 5}$	0.440	0.0528	0.0565

3-NN

Estimator	Mean	STD	RMSE
Err	0.351	0.0215	0.0000
$\overline{\mathrm{err}}$	0.094	0.0259	0.2589
$Err_{.632}$	0.283	0.0316	0.0797
$Err_{.632+}$	0.321	0.0384	0.0567
$Err_{CV\times n}$	0.350	0.0549	0.0615
$Err_{CV\times 10}$	0.360	0.0538	0.0619
$Err_{CV\times 5}$	0.371	0.0505	0.0616
Err_{BS}	0.186	0.0224	0.1683
$Err^{(1)}_{BS}$	0.393	0.0390	0.0648
$Err_{BSCV\times n}$	0.187	0.0230	0.1670
$Err_{BSCV\times 10}$	0.201	0.0233	0.1541
$Err_{BSCV\times 5}$	0.218	0.0238	0.1376
$Err_{.632}*$	0.296	0.0274	0.0657
$Err_{.632+}*$	0.341	0.0334	0.0438
$Err_{BS}*$	0.279	0.0276	0.0810
$Err^{(1)}_{BS}*$	0.414	0.0333	0.0767
$Err_{BSCV\times n}*$	0.305	0.0254	0.0575
$Err_{BSCV\times 10}*$	0.334	0.0264	0.0399
$Err_{BSCV\times 5}*$	0.334	0.0264	0.0399
$Err_{RCV\times n}$	0.350	0.0549	0.0613
$Err_{RCV\times 10}$	0.371	0.0447	0.0577
$Err_{RCV\times 5}$	0.360	0.0477	0.0567

Table 15: Pima Indians, $n = 60$, $d = 8$, 150 trials

17-NN

Estimator	Mean	STD	RMSE
Err	0.331	0.0161	0.0000
$\overline{\text{err}}$	0.295	0.0403	0.0500
$Err_{.632}$	0.321	0.0326	0.0313
$Err_{.632+}$	0.325	0.0312	0.0301
$Err_{CV\times n}$	0.330	0.0428	0.0421
$Err_{CV\times 10}$	0.330	0.0356	0.0358
$Err_{CV\times 5}$	0.337	0.0297	0.0332
Err_{BS}	0.301	0.0285	0.0398
$Err^{(1)}_{BS}$	0.335	0.0306	0.0299
$Err_{BSCV\times n}$	0.302	0.0285	0.0389
$Err_{BSCV\times 10}$	0.307	0.0264	0.0341
$Err_{BSCV\times 5}$	0.314	0.0235	0.0284
$Err_{.632}*$	0.324	0.0324	0.0324
$Err_{.632+}*$	0.329	0.0311	0.0324
$Err_{BS}*$	0.302	0.0303	0.0418
$Err^{(1)}_{BS}*$	0.340	0.0334	0.0366
$Err_{BSCV\times n}*$	0.280	0.0364	0.0616
$Err_{BSCV\times 10}*$	0.297	0.0290	0.0446
$Err_{BSCV\times 5}*$	0.297	0.0290	0.0446
$Err_{RCV\times n}$	0.330	0.0428	0.0421
$Err_{RCV\times 10}$	0.333	0.0236	0.0250
$Err_{RCV\times 5}$	0.331	0.0319	0.0320

3-NN

Estimator	Mean	STD	RMSE
Err	0.294	0.0212	0.0000
$\overline{\text{err}}$	0.123	0.0380	0.1768
$Err_{.632}$	0.246	0.0471	0.0696
$Err_{.632+}$	0.286	0.0596	0.0624
$Err_{CV\times n}$	0.293	0.0714	0.0717
$Err_{CV\times 10}$	0.293	0.0711	0.0709
$Err_{CV\times 5}$	0.305	0.0776	0.0783
Err_{BS}	0.175	0.0353	0.1256
$Err^{(1)}_{BS}$	0.318	0.0559	0.0629
$Err_{BSCV\times n}$	0.176	0.0357	0.1246
$Err_{BSCV\times 10}$	0.183	0.0359	0.1181
$Err_{BSCV\times 5}$	0.193	0.0360	0.1093
$Err_{.632}*$	0.257	0.0452	0.0605
$Err_{.632+}*$	0.302	0.0567	0.0584
$Err_{BS}*$	0.222	0.0427	0.0844
$Err^{(1)}_{BS}*$	0.335	0.0547	0.0695
$Err_{BSCV\times n}*$	0.178	0.0440	0.1261
$Err_{BSCV\times 10}*$	0.189	0.0442	0.1160
$Err_{BSCV\times 5}*$	0.189	0.0442	0.1160
$Err_{RCV\times n}$	0.293	0.0715	0.0717
$Err_{RCV\times 10}$	0.301	0.0617	0.0627
$Err_{RCV\times 5}$	0.297	0.0669	0.0671

B Description of the Statistical Data Cloning software

The software is written in C++ language, and was successfully compiled and run on different UNIX architectures (RehaHat Linux, SunOS, Solaris).

B.1 User interface

The program is invoked in command line, in the following format:

```
clone <filename> <c> <dir> <R> [n]
```

where

filename is the name of the file containing the original data,

c is the index of the class label in each data point (starting from zero),

dir is the directory where the clones should be written to,

R is the number of clones,

n is the size of each clone (by default, the same as the size of the original data set).

For example, suppose we need to generate 6 clones of the Pima Indians data set, each clone of the same size as the original data set. We assume that the data set is in file pima.dat. The class label is the last (9th) attribute. Assume we want to put the clones in the directory test under the current directory. Then, we should type

```
clone pima.dat 8 test 6
```

B.1.1 Description of the domain

The program receives a description of the data attributes in its standard input. A user can feed this description from a pre-written file using UNIX-like redirection mechanism. General format for a single feature is
<*type*> [<*constraints*> [<*n*> <l_1 r_1> …<l_n r_n>]]
where

type is either 'c' ('**c**'continuous), 'o' (discrete **o**rdinal), or 'n' (**n**ominal).

constraints applicable only for continuous or ordinal features, this is either a (no constraints, **a**ny value is admissible), 'x' (fi**x**ed values - only the values of this feature that appear in the original will appear in the clone), or 'b' (**b**ounded).

n applicable only for bounded features, this is the number of disjoint intervals in the admissible range of the feature. Must be followed by n interval descriptions.

$\mathbf{l_i r_i}$ applicable only for bounded features, this is a description of a single interval, given by left end and right end respectively. Open ends are denoted by (, closed by [. (use opening brackets for both right and left ends). ∞ is denoted by +, $-\infty$ by -.

For the Pima example, where all the attributes are discrete except for the 6th and the 7th, we will have a following file:

```
9

o b 1 [1 +

o b 1 [0 +

o b 1 [0 +
o b 1 [0 +

o b 1 [0 +

c b 1 [0 +

c b 1 [0 +

o b 1 [0 [150

o x
```

Note that we may as well denote the class label (the last attribute) as nominal, since for classification its value conveys no numerical information and can be treated as a text string.

B.2 Mathematical computation

B.2.1 Linear algebra

The core of our vector and matrix computation is based on the MESCHACH package [34]. The package is written in C. To facilitate its incorporation in the C++ code, a simple "wrapper" was written for it. The following files are related to this mathematical library:

algebra.h,algebra.cpp - the definitions of the basic classes: Vector, Matrix and IVector (integer vectors).

elmat.h,elmat.cpp - elementary functions: vector/matrix arithmetic, statistics, sorting etc.

matfun.h,matfun.cpp - more advanced functions: inner product, determinant, matrix inverse, eigenvalues etc.

B.2.2 Random number generation

To generate random number (key functionality for a resampling system) we used the random number generator in [28]. The files random.h,random.cpp provide the ability to generate

B.3 File organization

The files are organized in two directories. Directory source/ contains all the source C++ files, and the project's makefile. Directory include/ contains the header files. A third directory called meschach contains the ditribution of MESCHACH library, which is also available from the authors [34].