An Empirical Comparison of V-fold Penalisation and Cross Validation for Model Selection in Distribution-Free Regression

Charanpal Dhanjal, Nicolas Baskiotis, Stéphan Clémençon, Nicolas Usunier

An Empirical Comparison of V-fold Penalisation and Cross Validation for Model Selection in Distribution-Free Regression

Charanpal Dhanjal[1, *], Nicolas Baskiotis[1], Stéphan Clémençon[2], and Nicolas Usunier[1]

[1]LIP6, UPMC, 4 Place Jussieu, 75252 Paris Cedex 05, France
[2]Institut Telecom, LTCI UMR Telecom ParisTech/CNRS No. 5141, 46 rue Barrault, 75634 Paris Cedex 13, France

December 8, 2012

Abstract

Model selection is a crucial issue in machine-learning and a wide variety of penalisation methods (with possibly data dependent complexity penalties) have recently been introduced for this purpose. However their empirical performance is generally not well documented in the literature. It is the goal of this paper to investigate to which extent such recent techniques can be successfully used for the tuning of both the regularisation and kernel parameters in support vector regression (SVR) and the complexity measure in regression trees (CART). This task is traditionally solved via V-fold cross-validation (VFCV), which gives efficient results for a reasonable computational cost. A disadvantage however of VFCV is that the procedure is known to provide an asymptotically suboptimal risk estimate as the number of examples tends to infinity. Recently, a penalisation procedure called V-fold penalisation has been proposed to improve on VFCV, supported by theoretical arguments. Here we report on an extensive set of experiments comparing V-fold penalisation and VFCV for SVR/CART calibration on several benchmark datasets. We highlight cases in which VFCV and V-fold penalisation provide poor estimates of the risk respectively and introduce a modified penalisation technique to reduce the estimation error.

*Author for correspondence (charanpal.dhanjal@lip6.fr)

1 Introduction

Learning algorithms generally depend on a small number of real-valued or discrete parameters such as the size of a tree in hierarchical methods, the stopping criteria in boosting algorithms or explicit regularisation/smoothing parameters. These parameters naturally determine the complexity of the output function, and by doing so, also strongly influence generalisation ability. In a general sense the more "complex" the learnt function is, the more likely it is to overfit to the data. On the contrary, a simple predictor will be suboptimal if the data is informative with regard to the learning problem. From the model selection point of view the challenge consists in selecting values of the parameters of interest with a theoretical risk as small as possible. From a global perspective, there exist essentially two major approaches to model selection: methods related to data-splitting, with cross-validation [1] and its variants, and methods related to penalisation of the empirical risk (that obtained on the training set), with in particular the Structural Risk Minimisation principle [36]. Penalisation-based approaches aim to approximate the ideal model by adding a penalty or complexity-based term to the empirical risk, generally based on theoretical arguments (*i.e.* on probabilistic distribution-free upper bounds for the excess of risk). V-fold cross-validation (VFCV in abbreviated from) is widely used in machine-learning practice due to its (relative) computational tractability and empirical evidence of its good behaviour.

However, there is little theoretical justification concerning VFCV [3] and the question of an automatic choice of the parameter V remains widely open [8]. On the other hand, penalisation procedures generally suffer from two possible drawbacks, despite the fact that they are theoretically well-founded: either they are designed in a simplified framework and thus are not robust to complex situations (typically the case for Mallows' C_p in regression [26]), or they are of general purpose, as for instance the so-called Rademacher complexities [4], but are inaccurate in many cases. In order to refine Rademacher penalties, several authors proposed localisation techniques, giving rise to local Rademacher complexities [23], but these more accurate capacity functions are essentially of theoretical interest and cannot be used in practice due to the presence of unknown constants in their definition.

Combining both the robustness of cross-validation estimates and theoretical guarantees of penalisation procedures, a new type of general purpose penalisation procedures, called V-fold penalisation, has been recently proposed [2]. Both empirical and mathematical evidence of its efficiency have been shown in a heteroscedastic with random design regression framework, when considering the selection of finite-dimensional histogram models. While the selection of regressograms studied in [2] is convenient for theory since it allows precise mathematical investigations and is however general enough to show some relevant complex phenomena, we investigate in this paper the behaviour of V-fold penalties, and compare it to VFCV, for the tuning of the hyperparameters involved in the Support Vector Regression algorithm (SVR, [14]) and Classification and Regression Trees (CART, [6]) for regression. Indeed, these algorithms are two of the most extensively used regression tools in a wide variety of areas and the choice of efficient hyperparameters is known to be a decisive step of the learning process to attain good generalisation performance. Model selection for SVR has been addressed by several authors and many attempts, theoretically well-founded, have been proposed to answer this problem, among which: estimation of the hyperparameters from the data and the level of noise [34][24][12], leave-one-out bounds for SVR [11]. However, methods based on resampling procedures for the evaluation of the risk of each model have been proven to be significantly better than most of the other proposed automatic procedures [12, 29] and VFCV is generally the chosen method in practice [35]. For CART regression, the issue of model selection has not received as much attention, however [18] provides a theoretical validation of the standard CART pruning criterion. In this paper, our aim is to study V-fold penalisation for model selection and give insights into situations when one can improve on VFCV in practice. Particularly, the comparison of V-fold penalisation with VFCV on the problem of SVR and CART calibration takes importance, due to the highlighted relevance of VFCV in this central issue.

The remainder of this paper is organised as follows: Section 2 describes the statistical framework related to model selection for kernel SVR and CART. In Section 3 we recall VFCV and related works, we introduce in Section 4 V-fold penalisation and our improved penalisation approach. Experiments are addressed in Section 6, and conclusions are presented in Section 8.

2 Background and Preliminaries

As a first go, we outline the statistical setting of the model selection we shall subsequently study (generally referred to as the *distribution-free regression* setup). Here and throughout, a column vector is written in bold lowercase *e.g.* $\mathbf{x}$. Let $\mathcal{X} \times \mathcal{Y}$ be a measurable space endowed with an unknown probability measure P, with $\mathcal{Y} = \mathbb{R}$. $\mathcal{X}$ is called the *input space* and is usually a compact subset of $\mathbb{R}^d$, $d \geq 1$, and $\mathcal{Y}$ is the *target space*. We observe n i.i.d. labelled observations or examples $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subset \mathcal{X} \times \mathcal{Y}$. Furthermore, (X, Y) denotes a generic random variable, independent from the data S, drawn from P. Let $\mathcal{S}$ be the set of all measurable functions $s : \mathcal{X} \to \mathcal{Y}$ mapping from the input to target space. In the present paper, focus is on the mean absolute deviation:

$$L(s) = \mathbb{E}[|s(X) - Y|].$$

The regression task can thus be rewritten in these notations as finding minimum of the so-called least

absolute Bayes loss s_*, defined by:

$$L(s^*) = \min_{s:\mathcal{X}\to\mathcal{Y}} L(s).$$

2.1 Support Vector Regression

The SVR prediction function is of the form $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$ where $\mathbf{w} \in \mathbb{R}^d$ is a weight vector and b is a constant. In this case, one is interested in errors greater than a user-defined value $\epsilon \in \mathbb{R}^+$ (known as the ϵ-insensitive loss). Hence the optimisation task can be written as:

$$\begin{array}{rll} t = \arg\min_{\mathbf{w},b} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^n (\xi_i + \xi_i^*) & \\ \text{s. t.} & y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b \leq \epsilon + \xi_i & \\ & \langle \mathbf{w}, \mathbf{x}_i \rangle - y_i + b \leq \epsilon + \xi_i^* & \\ & \xi_i, \xi_i^* \geq 0, & \end{array} \quad (1)$$

where ξ_i and ξ_i^* are slack variables, and C is a *user-defined* trade-off between minimising the norm of the weight vector $\mathbf{w}$ (which can be seen as regularisation) and penalising errors greater than ϵ. A high value of C thus corresponds to a low regularisation level and the objective becomes then closer to that of minimising the empirical risk. The value of ϵ affects the number of Support Vectors (SV's in short), with larger values resulting in fewer SV's. In a slight abuse of notation, minimum values of $\mathbf{w}$ and b form a prediction function t. The SVR algorithm is often performed using kernels to model non-linear functions, where a kernel function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is used to find the inner product of the transformation of the input space $\mathcal{X}$ into its associated Reproducing Kernel Hilbert Space (RKHS), denoted by $\mathcal{H}_\kappa$. Note that κ can be written in terms of a transformation ϕ from input to kernel space $\langle \mathbf{u}, \mathbf{v} \rangle_{\mathcal{H}_\kappa} = \langle \phi(\mathbf{u}), \phi(\mathbf{v}) \rangle = \kappa(\mathbf{u}, \mathbf{v})$. Kernels functions usually depend on one or a few hyperparameters, *e.g.* polynomial, Gaussian Radial Basis Function (RBF) and sigmoid kernels [32]. The Gaussian RBF kernel is one of the most commonly used kernels and Boser, Guyon and Vapnik suggested its widespread use [5][19][37]. In the experiments described in Section 6, we therefore consider the Gaussian RBF kernel,

$$\kappa_\gamma(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2),$$

which depends on one real-valued positive parameter γ. In the following we denote the Gaussian RBF kernel κ_γ, $\gamma \in \mathbb{R}_+$. The optimal value of the regularisation parameter C can significantly change, and depends on the data. To ensure the performance in prediction of the SVR algorithm, the regularisation parameter, as well as the kernel, should thus be calibrated in each application. Formally, the question to be addressed is to find the best parameters (γ, C, ϵ) in terms of prediction. We thus aim at estimating the *oracle*, which is the model with the smallest risk, (where $t(\gamma, C, \epsilon)$ is the SVR learnt using parameters γ, C, ϵ),

$$\arg \min_{(\gamma,C,\epsilon)\in\mathbb{R}_+^3} L\left(t\left(\gamma, C, \epsilon\right)\right),$$

which is unknown since it depends on the law P of data and which optimises the least squares error.

2.2 CART Regression

Another important algorithm for regression is CART in which one learns a tree like the one exemplified in Figure 1. To regress a new example $\mathbf{x}$, it is filtered down to a leaf node via a decision at each link and then assigned a real number target. In the illustration, if the first feature of $\mathbf{x}$, $\mathbf{x}_{(1)} \leq \theta_1$, for some threshold θ_1, then it is labelled 0.0. Similarly, if $\mathbf{x}_{(1)} > \theta_1$ and $\mathbf{x}_{(2)} > \theta_3$ then the example is labelled -0.3. Regression trees have been successfully used in a variety of applications in such as vector quantisation [13], meteorology [10] and medicine [38] for example, and have the crucial advantage of being easy to interpret and easy to compute. To construct a regression tree one starts with

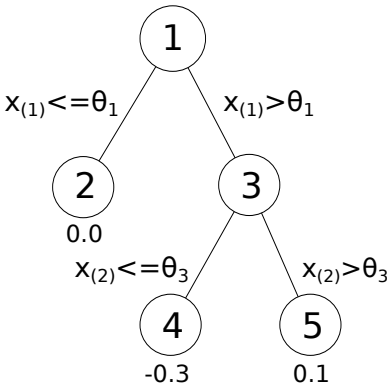


Figure 1: An example of a decision tree.

the root node which contains all of the training examples $S_0 = S$. One then decides how to split the examples based on a feature k and a threshold θ.

Given a choice of these values, the left child contains the examples $S_L = \{(\mathbf{x}, y) \in S_0 \mid \mathbf{x}_{(k)} \leq \theta\}$ and the right one is $S_R = \{(\mathbf{x}, y) \in S_0 \mid \mathbf{x}_{(k)} > \theta\}$. The optimal feature-threshold pair k^*, θ^* is found by minimising the squared error of the split, i.e. k^*, θ^* are found using:

$$\begin{aligned} \arg\min_{k,\theta} \quad & \sum_{(\mathbf{x},y)\in S_L} (y - \mu_{S_L})^2 + \sum_{(\mathbf{x},y)\in S_R} (y - \mu_{S_R})^2 \\ \text{s. t.} \quad & S_L = \{(\mathbf{x}, y) \in S_0 \mid \mathbf{x}_{(k)} \leq \theta\} \\ & S_R = \{(\mathbf{x}, y) \in S_0 \mid \mathbf{x}_{(k)} > \theta\}, \end{aligned}$$

where $\mu_{S_L} = \frac{1}{|S_L|} \sum_{(\mathbf{x},y)\in S_L} y$ and $\mu_{S_R} = \frac{1}{|S_R|} \sum_{(\mathbf{x},y)\in S_R} y$ are the mean labels for the left and right nodes and hence $(y - \mu_{S_0})^2$ is the squared error between y and the mean label. A simple way to solve this optimisation is to iterate through each feature and threshold and choose the one with the lowest objective value. After splitting on the root node, one recursively splits on the resulting child nodes until no more splits are possible, *i.e.* a node contains fewer examples than a user defined value ℓ. Following the growing phase, one prunes the resulting tree as smaller trees have been shown to improve generalisation error. In CART, one uses an approach called *cost complexity pruning* which generates a series of trees pruned from the original tree and then selects one of the trees in the sequence. For the ith node in the unpruned tree which contains examples S_i, one computes the error if the tree was pruned at that node and compares it to the error if the subtree starting at that node R_i is kept. The difference in these errors divided by the number of leaves of the subtree gives an indication of the error difference per leaf, i.e.

$$\alpha_i = \frac{\tilde{L}(S_i, \hat{R}_i) - \tilde{L}(S_i, R_i)}{|R_i|_l - |\hat{R}_i|_l},$$

in which $\tilde{L}(S_i, R_i)$ is the squared error of a set of examples S_i using subtree R_i, $\hat{R}_i$ is the root of R_i, and $|R_i|_l$ is the number of leaves in R_i. In simple terms, the higher the value of α_i, the bigger the reduction in error of the subtree per leaf. One can compute α_i for all nodes in the tree and hence, if we prune nodes with α_i greater than a threshold $\sigma \in \{\alpha_1, \dots \alpha_{|T|}\}$, where $|T|$ is the number of nodes in the tree, we obtain a sequence of trees which decrease in size as σ increases. Instead of choosing σ directly in the model selection stage, we pick a threshold t and choose the largest tree smaller or equal in size to t. Therefore, as before, the model selection task can be written in terms of search for the parameter t. We aim at estimating the oracle (where $f(t)$ is the decision tree learnt using parameter t),

$$\arg\min_{t\in\mathbb{Z}} L\left(f\left(t\right)\right),$$

Estimating the oracle is a model selection task, each model being represented here by a fixed value of t, where penalisation is a natural way to proceed, as explained in Section 4 below. However, let us first briefly recall the method which is usually employed for model selection, namely V-fold cross-validation.

3 V-fold Cross Validation

The idea of cross-validation for model selection is to estimate the risk of the considered estimator on each model by using a repeated data-splitting scheme, and then to select the model that minimises these estimates of the risk. The fact that data-splitting strategies give accurate estimates of the risks only relies on the independence between each training and testing set. Consequently, the interest of CV is that it is based on a heuristic that can be applied with great universality. Many data-splitting rules have been proposed, such as leave-one-out (LOO, [1]), leave-p-out (LPO, [33]), balanced incomplete CV (BICV, [33]), repeated learning-testing (RLT, [7]). V-fold cross-validation (VFCV) was introduced by Geisser [17], see also [7] as a computationally efficient alternative to LOO cross validation. We will consider primarily VFCV, which is certainly the most commonly used cross-validation rule in practice. Moreover, it is generally the procedure which is considered for the calibration of the SVR hyperparameters [35] [21]. In VFCV, the examples are partitioned into V subsamples of n/V examples each (with a maximal deviation of one) $B_1, \dots, B_V$. At the jth fold one trains on $S \setminus B_j$ and then evaluates the error on B_j, and one averages the errors over all V folds. For model selection this is repeated over a grid of parameters in order to select those with the lowest error. Despite the generality of the heuristic underlying the procedure, there are two drawbacks in the V-fold cross-validation method for model selection. First, at a fixed V, the procedure is known

to be asymptotically suboptimal in the sense that the risk of the selected estimator is not equivalent to the risk of the oracle when the number of data tends to infinity. More precisely, [2] shows in a heteroscedastic with random design regression framework, that VFCV with fixed V satisfies an oracle inequality with a constant $A > 1$ which relates the excess risk of the selected model to the excess risk of the oracle, and that this suboptimal constant cannot be improved asymptotically. The keystone of such a result is that at a fixed V, the VFCV criterion is biased compared to the true risk [9][31]. Indeed, since the validation sets are independent of their respective training sets, the expectation of the VFCV criterion can be related to the expectation of the true risk $\mathbb{E}[\mathrm{crit}_{\mathrm{VFCV}}\, s\,(Q)]$ for a learner s with parameter set $Q \in \mathcal{Q}$ as follows,

$$\mathbb{E}\left[L_S^{(j)}\left(s^{(-j)}\,(Q)\right)\right] = \mathbb{E}\left[L\left(s^{[-1]}\,(Q)\right)\right]\ ,$$

where $s^{[-1]}\,(Q)$ is the output of the learner trained with $(1-1/V)\,n$ i.i.d. examples, $s^{(-j)}$ is the learner trained with $S \setminus B_j$, and $L_S^{(j)}$ is the loss with respect to the partition B_j of S. Since the true risk generally decreases with more data, it appears that the expectation of VFCV criterion roughly overestimates the expectation of the true risk, and that this bias should be decreasing whenever V increases. The previous observation suggests that, in order to mimic the oracle in terms of performance in prediction, one should take a V which is as large as possible. This is where appears the second drawback concerning VFCV: there is no rule in practice to choose the optimal V. Indeed, the best CV estimator of the risk is not necessarily the best model selection procedure, and [8] highlight that LOO is the best estimator of the risk, whereas 10-fold cross-validation is more efficient for model selection purpose. This can be explained by the fact the bias in the VFCV estimation of the risk is actually an advantage for model selection with a few or medium number of examples, contrary to the asymptotic framework. Indeed, as claimed in [2] a slightly over-pessimistic estimation of the risk, as in VFCV, gives for a fixed number of observations a more robust model selection procedure and roughly contradicts the bad effects of the variance of risk estimation.

4 V-fold Penalisation

Penalisation is a natural strategy for the task of estimating the oracle $s\,(Q^*)$. Indeed, the definition of the oracle can be rewritten as the sum of the empirical loss and an unknown term, which is thus an ideal penalty in the sense that it allows one to recover the oracle:

$$\arg\min_{Q\in\mathcal{Q}} L_S\left(s\,(Q)\right) + \mathrm{pen}_{\mathrm{id}}\,(Q)\,,$$

in which $s(Q)$ is a function mapping from input to target space under hyperparameters Q, and the ideal penalty is as follows,

$$\mathrm{pen}_{\mathrm{id}}\,(Q) = L\left(s\,(Q)\right) - L_S\left(s\,(Q)\right).$$

Hence, penalisation aims at mimicking the oracle by selecting, for a known penalty function the estimator

$$\arg\min_{Q\ \in\mathcal{Q}} L_S\left(s\,(Q)\right) + \mathrm{pen}\,(Q)\,.$$

A good penalty in terms of prediction is one which gives an accurate estimate of the ideal penalty $\mathrm{pen}_{\mathrm{id}}$. The central idea of V-fold penalties proposed in [2] is to directly estimate the ideal penalty by a subsampled version of it. For some constant $C_V \geq V-1$, the V-fold penalty $\mathrm{pen}_{\mathrm{VF}}\,(Q)$ is

$$\frac{C_V}{V}\sum_{j=1}^{V}\left[L_S(s^{(-j)}(Q)) - L_S^{(-j)}(s^{(-j)}(Q))\right],$$

and so the corresponding selected hyperparameters are given by

$$\arg\min_{Q\in\mathcal{D}} L_S\left(s\,(Q)\right) + \mathrm{pen}_{\mathrm{VF}}\,(Q)\,,$$

where $\mathcal{D}$ is a discrete grid upon the set $\mathcal{Q}$. The V-fold penalty indeed mimics the structure of the ideal penalty, in such a way that the quantities related to the unknown law of data P (respectively to the empirical measure P_S) are replaced by quantities related to the empirical measure P_S (respectively to the subsampling measures $P_S^{(-j)}$), in the same analytic manner. The design of the V-fold penalties is thus an adaptation of Efron's resampling heuristics [15] to the subsampling scheme of the V-fold procedure. It has been shown in [2] by

considering the selection of regressograms in a heteroscedastic regression framework, that V-fold penalisation with $C_V = V - 1$ is asymptotically optimal for a fixed V, whereas in this case, VFCV is asymptotically suboptimal, due to its bias on the estimation of the risk. Moreover, the choice of $C_V = V - 1$ in the definition of the V-fold penalty corresponds to the Burman's corrected VFCV criterion [9]. Therefore, we use in our experiments $C_V = V - 1$ although we also explore different values. Another advantage, highlighted in [2], of V-fold penalisation, compared to VFCV, is that it seems to be more regular with respect to the choice of V. At a heuristic level, this can be explained by observing that since V-fold penalisation corrects the bias of VFCV, it is only variability of V-fold estimates that matters here, a variability which is smaller for a larger V. Finally, it should be noted that the constant C_V in the definition of the V-fold penalty can be viewed as a degree of freedom, which potentially allows to deal with the bias of the proposed risk estimation, without varying the value of V, contrary to VFCV where only V fixes simultaneously and in a tricky manner, the bias and variance of the risk estimation. In [2] it is shown that choosing C_V to overpenalise (*i.e.* pen_{VF} is larger than pen_{id} even in expectation) can improve prediction performance when the signal to noise ratio is small. The choice is a difficult one however, and according to empirical results on synthetic datasets, it depends on the sample size, noise level and smoothness of the regression function.

5 A Complexity-Based Selection of C_V

In practice, as we shall later see, for a fixed training set and V, the approximate penalty as given by pen_{VF} often poorly approximates the ideal penalty and it cannot be improved by varying the penalisation constant C_V. To study the cause of the problem we analyse the pen_{VF} criterion relative to the ideal penalty. Consider the first term in the sum of pen_{VF}, $L_S\left(s^{(-j)}(Q)\right)$, and write it in terms of the loss on the training and test set:

$$\begin{aligned} & \frac{1}{n}\sum_{i\in S^{(-j)}} \ell_i(s^{(-j)}(Q)) + \frac{1}{n}\sum_{i\in S^{(j)}} \ell_i(s^{(-j)}(Q)) \\ = \quad & \frac{V-1}{V} L_S^{(-j)}(s^{(-j)}(Q) + \frac{1}{V} L_S^{(j)}(s^{(-j)}(Q), \end{aligned}$$

in which $\ell_i(\cdot)$ is the loss for the ith example. The link between the lines can be seen by noting that

$$L_S^{(-j)}(s^{(-j)}(Q) = \frac{V}{(V-1)n} \sum_{i\in S^{(-j)}} \ell_i(s^{(-j)}(Q)).$$

When we put the above form of $L_S\left(s^{(-j)}(Q)\right)$ into pen_{VF} we obtain

$$\frac{C_V}{V}\left[\frac{1}{V}\sum_{j=1}^{V}\left(L_S^{(j)}\left(s^{(-j)}(Q)\right) - L_S^{(-j)}\left(s^{(-j)}(Q)\right)\right)\right],$$

and the term inside the square brackets is the empirical expectation of the error on the test set minus the error on the training set. One can say that this an approximation of the ideal penalty using $(V-1)n/V$ examples since the loss term on the right side is computed over $S \setminus B_j$. A variety of error bounds have the penalty proportional to a complexity measure and inversely proportional to the number of examples to some power of a learning rate $\beta(Q)$ (see [25] for example). In other words, for a learner s with parameters Q we consider the following form of the penalty:

$$\text{pen}_V(Q) = \frac{C_V}{V}\frac{D(Q)}{(n(V-1)/V)^{\beta(Q)}} \qquad (2)$$

where $D(Q)$ is the complexity of $s(Q)$ and $\beta(Q)$ is a learning rate, and we have replaced the square bracketed term in pen_{VF} with $D(Q)/(n(V-1)/V)^{\beta(Q)}$. A learning rate of 0 implies a large penalty and that we have overfitted the data and hence, for a fixed V and sample size we do not learn anything (in other words one predicts on a test set randomly). As the sample size increases one continues to overfit and hence the penalty term is $C_V D(Q)/V$ regardless of the sample size. In contrast when $\beta(Q) = 1$ the penalty is small, and rapidly decreases with the sample size, and also with V. A limit of 1 for $\beta(Q)$ is natural for the learning rate since this is the bound often used in complexity bounds. The ideal penalty has the form

$D(Q)/n^{\beta(Q)}$ for n examples and hence we would like to choose C_V above so that

$$\frac{C_V}{V}\frac{D(Q)}{(n(V-1)/V)^{\beta(Q)}} = \frac{D(Q)}{n^{\beta(Q)}}, \tag{3}$$

and solving gives $C_V = (V-1)^{\beta(Q)}/V^{\beta(Q)-1}$. A learning rate of 0 which occurs with complex models (e.g. large decision trees) implies $C_V = V$ and similarly for small models where $\beta(Q) = 1$ we have $C_V = V - 1$ as suggested asymptotically above. In this latter case we recover exactly the value of C_V suggested in [2]. On the whole, $\text{pen}_V(Q)$ is an estimation of the ideal penalty using pen_{VF} and the model complexity $D(Q)$. It remains to consider how one computes the learning rate. We equate Eq. (2) with pen_{VF} and then taking logs results in $\log(\text{pen}_{VF}(Q)/C_V)$ equal to

$$-\log(V) - \beta(Q)\log(n(V-1)/V) + \log(D(Q)).$$

One finds the gradient of $\log(\text{pen}_{VF}(Q)/C_V) + \log(V)$ versus $\log(n(V-1)/V)$ for a selection of different V values whilst fixing Q and n, in order to find the learning rate $\beta(Q)$.

6 Experimental Setup

We study the behaviour of VFCV and V-fold penalisation on a collection of benchmark datasets. The scikit-learn library in Python [30] is used to generate the output of the RBF SVR and CART algorithms.

In total, 10 datasets from the UCI machine learning repository [16] and DELVE [28] are used. Each dataset is split into 100 training and test realisations/sets after being processed so that the examples and labels have zero mean and unit standard deviation. Details are provided in Table 1. When comparing model selection algorithms, a statistically significant improvement of one method over another is such that the mean error is greater and by using a paired t-test. For the t-test we take the sample of errors over all realisations for 2 methods, then compute a p value and reject the null hypothesis, that the means are equal, if $p < 0.1$. In all experiments we use the mean absolute error, i.e. for a prediction function $f : \mathcal{X} \to \mathcal{Y}$, the error is $\frac{1}{n}\sum_{i=1}^{n} \|f(x_i) - y_i\|_1$.

Dataset	Learn	Test	d	Abrv.
abalone	835	3342	8	ab
add10	2937	6855	10	ad
comp-activ	2457	5735	22	ca
concrete	309	721	8	cc
parkinsons-motor	2937	2938	20	pm
parkinsons-total	2937	2938	20	pt
pumadyn-32nh	3276	4916	32	pd
slice-loc	26750	26750	385	sl
winequality-red	1066	533	11	wr
winequality-white	3265	1633	11	ww

Table 1: Information on the benchmark datasets used. There are 100 learn/test splits for each dataset.

6.1 Model Selection

In all of the following experiments we use a grid to approximate the set of hyperparameters. The SVR penalty is chosen as $C \in \{2^{-10}, 2^{-8}, \ldots, 2^{12}\}$, the kernel width as $\gamma \in \{2^{-10}, 2^{-8}, \ldots, 2^{2}\}$, and $\epsilon \in \{2^{-4}, 2^{-3}\}$. More sophisticated ways of searching in the hyperparameter space actually exist, such as the iterative process derived from the so-called active sets method and used in [20, 12, 29] to walk along the entire path of the SVR: γ is fixed and *all* values of C are considered. Others heuristics involve e.g. genetic algorithms [22], local search methods [27]. However, some degeneracies can occur and so, a search on a grid should be more stable. Moreover, the grid-search has also the advantage of being easily parallelised, because each value of (γ, C, ϵ) is independent from the others. In the case of CART regression, we pick the bound on the tree size t from $\{2^1 - 1, \text{round}(2^{1.5} - 1), \ldots, 2^7 - 1\}$.

An important characterisation of the model picked during the selection phase is its complexity. In all of the model selection techniques we choose a set of parameters over $n(V-1)/V$ examples however the final predictor is training using all n examples. Ideally, we would like the complexity to be identical in both model selection and whilst training using all examples, since the penalty is a function of complexity (Equation 2). For the SVR the norm of the weight vector is the measure of complexity used in error bounds (see [35]). For this reason, we compute the mean norm of the SVR weight vector $\|\mathbf{w}\|$, for each value of C, γ, ϵ used in

model selection and store $\|\mathbf{w}^*\|$, the norm corresponding to the lowest error, as well as the corresponding values γ^*, ϵ^*. When we train using all n examples, we again compute $\|\mathbf{w}\|$'s corresponding to each value of C, for γ^*, ϵ^*, and choose C with corresponding norm closest to $\|\mathbf{w}^*\|$. In the case of CART trees, we can be slightly more direct: for the optimal bound on the tree size, t^*, we compute the real mean tree size found during model selection and round to the nearest integer $\hat{t}$. The value of $\hat{t}$ is then used to train over all n examples.

6.2 Primary Setup

In order to test the model selection techniques we take random training subsamples of either 50, 100 or 200 examples of the learning sets to observe model selection on a limited number of examples. Furthermore, we test using $2, 4, \ldots, 12$ folds. Model selection is performed using each subsample and then SVR or CART is trained using the optimal parameters and the entire subsample. This is repeated for each realisation and results are averaged over the entire set of realisations. As well as recording the error obtained using model selection over the realisations, we also store the difference between the "ideal" and approximated penalty. This former quantity is computed simply as the difference between the V-fold penalty and the penalty as computed using the test set. All results for V-fold penalisation are evaluated with $C_V = (V-1)\alpha$ with $\alpha \in \{0.6, 1.2, \ldots, 1.6\}$ being the multiplier for the penalisation. We denote the types of model selection methods as: VFCV, V- fold penalisation (`PenVF`) and V-fold penalisation using a learning rate (`PenVF+`).

For the `PenVF+` method one needs to compute learning rates for each model (set of parameters). We use the same training sets as above and vary V from the set $\{2, 3, \ldots, 12\}$. The quantities $\log(\text{pen}_V(Q)) + \log(V)$ versus $\log(n(V-1)/V)$ are computed and the gradient, found using linear regression, provides $\beta(Q)$ which in turn is used to calculate $C_V = (V-1)^{\beta(Q)}/V^{\beta(Q)-1}$. As this estimation of $\beta(Q)$ can be unstable especially with small training sets we clip its value to lie within the valid range $[0, 1]$.

7 Experimental Results

7.1 Comparison of Penalisation and VFCV with $V = 2$

We start by studying the SVR results in Table 2 which shows errors for all datasets when $V = 2$, $\alpha = 1.0$. We consider $V = 2$ in this case since it provides the greatest distinction between the model selection methods. For `PenVF+` many of the results are comparable to VFCV when we also consider the standard deviations. As the same time, `PenVF+` does not always improve upon `PenVF`. In contrast `PenVF` can perform significantly worse than both VFCV and `PenVF+`, for example with `abalone`, `winequality-red` and `winequality-white`. Also of note is that the difference in error between VFCV and `PenVF` does not improve with $m = 200$ with `abalone` for example: it is $0.08, 0.089, 0.089$ with $m = 50, 100, 200$.

Also shown at the bottom of Table 2 is the equivalent CART results. It is evident that error rates are generally worse than the SVR with the exception of `pumadyn-32nh`. Also, we see that penalisation provides a larger advantage relative to VFCV in this case. One explanation is that CART is more sensitive to its hyperparameters. We observe that `PenVF+` is equivalent or improves over VFCV in nearly every case, and there are $5, 7, 7$ wins for $m = 50, 100, 200$ respectively. Again we see that `PenVF` performs poorly with `abalone`, `winequality-red` and `winequality-white`.

7.2 Paired t-test Comparison with VFCV

Table 3 shows the results of the paired t-tests to compare `PenVF` with $\alpha = 1.0$ and `PenVF+` with VFCV. Consider first the SVR results. Here we see that as one might expect, there are few statistically significant differences between 10-fold CV and `PenVF`. Indeed, results indicate that as V increases the penalisation methods and VFCV become more similar. In particular we see that `PenVF+` is identical to VFCV in all but one or two cases, with the main exception being 5 draws with $m = 200$ and $V = 2$, in which there are 2 improvements and 3 losses (`abalone`, `pumadyn-32nh` and `winequality-red`). `PenVF` fares worse against VFCV: we see more wins but at the same time more

	$m = 50$			$m = 100$			$m = 200$		
	VFCV	PenVF+	PenVF	VFCV	PenVF+	PenVF	VFCV	PenVF+	PenVF
	SVR								
ab	.544 (.031)	.549 (.033)	.624 (.036)	.514 (.017)	.519 (.027)	.603 (.026)	.497 (.013)	.501 (.019)	.584 (.012)
ad	.471 (.037)	.471 (.031)	.473 (.036)	.409 (.020)	**.401** (.022)	**.404** (.023)	.330 (.021)	**.320** (.016)	**.320** (.013)
ca	.236 (.052)	.227 (.055)	**.220** (.060)	.172 (.051)	.165 (.048)	.165 (.048)	.130 (.026)	.127 (.021)	.128 (.021)
cc	.472 (.046)	.478 (.050)	.485 (.050)	.413 (.025)	.415 (.031)	.430 (.036)	.366 (.021)	**.350** (.021)	.366 (.020)
pm	.272 (.030)	.274 (.030)	.282 (.032)	.237 (.012)	.240 (.017)	.254 (.024)	.217 (.009)	.215 (.010)	.219 (.010)
pt	.116 (.024)	.116 (.025)	.114 (.024)	.092 (.012)	.092 (.013)	.090 (.014)	.080 (.009)	.078 (.009)	.078 (.010)
pd	.814 (.032)	.813 (.037)	**.806** (.026)	.796 (.021)	.797 (.028)	.793 (.015)	.778 (.014)	.783 (.018)	.779 (.015)
sl	.423 (.037)	.427 (.043)	.415 (.037)	.341 (.023)	.346 (.022)	**.334** (.020)	.290 (.012)	.292 (.018)	**.279** (.009)
wr	.712 (.064)	.709 (.062)	.745 (.057)	.660 (.034)	.667 (.041)	.710 (.048)	.637 (.021)	.643 (.030)	.704 (.037)
ww	.728 (.028)	.734 (.039)	.752 (.034)	.704 (.025)	.710 (.035)	.731 (.034)	.680 (.025)	.679 (.023)	.707 (.031)
	CART								
ab	.699 (.057)	.696 (.065)	.713 (.065)	.665 (.044)	.667 (.049)	.691 (.052)	.633 (.028)	.632 (.040)	.661 (.036)
ad	.750 (.063)	**.725** (.066)	**.720** (.072)	.637 (.055)	**.616** (.048)	**.624** (.040)	.567 (.036)	**.556** (.027)	.571 (.030)
ca	.330 (.116)	**.305** (.100)	**.303** (.099)	.220 (.052)	.215 (.051)	.215 (.050)	.186 (.031)	.180 (.023)	.182 (.023)
cc	.681 (.084)	**.642** (.068)	**.624** (.056)	.569 (.069)	**.535** (.051)	**.526** (.045)	.459 (.037)	**.444** (.030)	**.438** (.029)
pm	.348 (.055)	**.331** (.038)	**.323** (.036)	.282 (.037)	**.268** (.028)	**.262** (.027)	.210 (.025)	**.204** (.022)	**.202** (.021)
pt	.268 (.160)	.259 (.161)	.257 (.161)	.213 (.077)	**.198** (.044)	**.198** (.045)	.161 (.020)	.158 (.020)	.157 (.020)
pd	.812 (.036)	.821 (.054)	1.039 (.089)	.796 (.014)	.794 (.048)	.947 (.104)	.751 (.051)	**.694** (.057)	.808 (.077)
sl	.679 (.128)	**.604** (.105)	**.594** (.102)	.484 (.088)	**.453** (.071)	**.444** (.069)	.362 (.048)	**.344** (.032)	**.340** (.030)
wr	.812 (.067)	.797 (.065)	.799 (.072)	.776 (.059)	**.756** (.061)	.765 (.066)	.738 (.045)	**.721** (.044)	.730 (.048)
ww	.789 (.052)	.786 (.065)	.807 (.072)	.771 (.040)	**.762** (.041)	.782 (.059)	.760 (.035)	**.744** (.038)	.768 (.053)

Table 2: Error rates (with standard deviations in parentheses) for cross validation with the SVR (top) and CART (bottom) and penalisation for $V = 2$, $\alpha = 1.0$. Statistically significant improvements over VFCF are in bold. With the SVR, `PenVF+` is generally comparable to VFCV and `PenVF` is more variable. With CART, the penalisation methods both improve over VFCV a number of times.

losses. Our later analysis will shed light on why this is the case. We also compared the "ideal" model, in which the test set is used during model selection, with VFCV and found that one can generally gain improvements except in the case of `slice-loc`, which has a large number of features. In every case the ideal model selector can improve over 2-fold CV.

The CART results at the bottom of Table 3 show that for $V > 4$ penalisation is identical to VFCV. Clearly the bias with low values of V in conjunction with VFCV is more prominent in this case. We have already examined the $V = 2$ case and with $V = 4$ we see improvements in one case for `PenVF+` for $m = 50, 200$. With the ideal errors we see that with more of the datasets compared to SVR, performance cannot be improved by using the test realisation and furthermore as m increases improvements over VFCV are increasingly difficult except for the 2-fold case.

7.3 Optimal Penalisation Constant is Dataset Dependent

To discover the effect of overpenalisation, observe Figure 2 which shows the errors on 2 datasets as α varies when $V = 10$. The effect of α is clearly dataset dependent: on `abalone` observe that the error tends to decrease with the SVR with more penalisation, and hence a slight amount of overpenalisation ($\alpha = 1.2$) is recommended. In contrast, overpenalisation increases the error with `slice-loc`. In total, 5 of the datasets benefited from overpenalisation and 3 improved with underpenalisation. With `pumadyn` and `parkinsons-total`, a value of $\alpha = 1.0$ as predicted by the theory gave optimal results. Results were similar with CART except that 7 of the datasets benefit from underpenalisation. Notice that as sample size m increases the choice of α becomes less important since estimates of error for each model are more reliable and the penalty terms become small. We noticed that with

	2	4	6	8	10	12	2	4	6	8	10	12	2	4	6	8	10	12
	SVR																	
PenVF+	0	0	0	1	0	0	10	10	10	9	10	10	0	0	0	0	0	0
PenVF	5	4	3	1	1	0	3	5	7	9	9	10	2	1	0	0	0	0
Ideal	0	0	0	0	0	0	0	1	1	1	1	1	10	9	9	9	9	9
PenVF+	0	1	0	0	0	0	9	9	10	10	10	10	1	0	0	0	0	0
PenVF	5	5	3	3	2	1	3	3	6	6	8	9	2	2	1	1	0	0
Ideal	0	0	0	0	0	0	0	1	1	1	1	1	10	9	9	9	9	9
PenVF+	3	1	0	1	0	0	5	9	10	9	10	10	2	0	0	0	0	0
PenVF	3	4	3	3	2	1	5	5	6	6	7	8	2	1	1	1	1	1
Ideal	0	0	0	0	0	0	0	1	1	1	1	1	10	9	9	9	9	9
	CART																	
PenVF+	0	0	0	0	0	0	5	9	10	10	10	10	5	1	0	0	0	0
PenVF	2	1	0	0	0	0	3	6	10	10	10	10	5	3	0	0	0	0
Ideal	0	0	0	0	0	0	1	2	2	2	2	2	9	8	8	8	8	8
PenVF+	0	0	0	0	0	0	3	10	10	10	10	10	7	0	0	0	0	0
PenVF	2	3	0	0	0	0	3	5	10	10	10	10	5	2	0	0	0	0
Ideal	0	0	0	0	0	0	1	2	4	3	4	3	9	8	6	7	6	7
PenVF+	0	0	0	0	0	0	3	9	10	10	10	10	7	1	0	0	0	0
PenVF	2	2	0	0	0	0	5	8	10	10	10	10	3	0	0	0	0	0
Ideal	0	0	0	0	0	0	0	4	4	4	5	4	10	6	6	6	5	6

Table 3: Number of statistically significant losses (left block), draws (middle block) and wins (right block) against standard errors for CV and across different numbers of folds using $\alpha = 1.0$. The sample size m is 50 (top), 100 (middle), and 200 (bottom). As V increases the penalisation methods become more similar to VFCV. In particular for CART, when $V > 4$ penalisation is identical to VFCV.

CART, VFCV consistently underestimates the tree size whereas `PenVF` chooses larger sizes in general and this can be an advantage or disadvantage depending on the variation of error with t. Also observed is that as expected VFCV provides a pessimistic error compared to the ideal case and `PenVF` is generally more accurate than VFCV, however, as in model selection since we pick the model with the lowest error, this does not always translate into the best predictor.

7.4 On the Estimation of the Ideal Penalty

Next we study the approximated penalty and how it differs from the "ideal" penalty in the case of CART with $V = 2$, see Figure 3. Notice that the curves for `PenVF` and `PenVF+` are shorter than the ones for the "ideal" penalty since only half the examples are used for training, limiting the tree size. The `PenVF` method diverges from the ideal case when we grow large trees, but is close to the ideal case for small trees. This change occurs with relatively small trees: size 4 with $m = 50$ and size 11 with $m = 100$. This pattern was observed with most of the datasets. When we looked at a greater number of folds, `PenVF` was close to the ideal penalty. In contrast, `PenVF+` does not diverge as the tree size increases, however it seems to slightly overestimate the penalty.

The question remains about which cases `PenVF` improves over VFCV for low values of V for CART. Figure 4 demonstrates that the error estimation for `PenVF` is generally optimistic as model size increases. With `abalone` for example the optimal tree was of size 7 nodes, however `PenVF` chooses one of size 22. In contrast, on some datasets large trees did not overfit the test set and hence in these cases `PenVF` can perform better than VFCV. This also sheds some light on Figure 2 where we see that overpenalisation helps in some cases but not in others. `PenVF+` provides the best estimate of the error in general, however it also results in a larger tree size that the ideal case.

7.5 Extended Setup

In this final set of experiments we explore further the distinction between penalisation and VFCV with CART by considering $m = 500$ and using the same set of folds as in the original setup, see Table 4. The interesting figures in this case are those corresponding to 2 and 4 folds in which we see that `PenVF+` wins for 8 and 2 datasets respectively. In fact, since in the ideal case one can only win 8 times with $V = 2$ this certainly demonstrates the effectiveness of `PenVF+` and bias in VFCV in this case.

7.6 Key Points

Our experimental analysis has painted a detailed picture of penalisation versus cross validation for model selection. The bias in VFCV is evident with small values of V and small training sets, and we observed that as V and the training set sizes increase the model selection methods become more similar. With the SVR, `PenVF` makes a number of loses relative to VFCV and these losses are nearly all corrected with our modified penalisation `PenVF+`. Penalisation is more effective in general with CART: when $V > 4$ both `PenVF` and `PenVF+` are not statistically significantly different to VFCV, and for $V = 2$, `PenVF+` is at least as good as VFCV or improves over it in nearly every case, winning $5, 7, 7$ times for $m = 50, 100, 200$ examples. In contrast `PenVF` is more variable in comparison to VFCV and one reason for this is that it underestimates the penalty to a large degree with large models. On some datasets larger trees did not increase the error and hence in these cases `PenVF` performs well. In general `PenVF+` provides a much better approximation of the ideal penalty compared to `PenVF`. The most striking results were with CART and $V = 2$ in which we saw that `PenVF+` improves over VFCV in 8 out of 10 cases with $m = 500$ examples.

8 Conclusions

Model selection is a critical part of machine learning as it can dramatically affect generalisation performance. In practice, cross validation over a grid of parameter values is often used, and it has been shown to be very effective in a variety of cases. We studied V-fold penalisation which is a general purpose penalisation procedure that aims at improving on VFCV by correcting its bias and is proved in [2] to be asymptotically optimal in a histogram regression setting. V-fold penalisation is simple to implement and the penalised error can be computed using the same predictions as cross validation and hence at negligible additional computational cost. Furthermore, we propose an improvement of penalisation, called `PenVF+`, which takes into account learning rates in order to correct under-penalisation with large models.

We conducted an extensive empirical investigation into VFCV and V-fold penalisation over a collection of 10 well known benchmark datasets using an SVR with the RBF kernel and CART. With low values of V, penalisation can provide an advantage over VFCV but this advantage rapidly diminishes as V increases. Furthermore, in some cases penalisation fared worse than cross validation. When we compare the penalty with the "ideal" penalty, we observed that `PenVF` underestimates the penalty with large models and `PenVF+` improves penalty estimation in these cases. Hence, there is no fixed overpenalisation constant even for a particular dataset, but rather the penalisation should vary with model complexity as with `PenVF+`.

References

[1] David M. Allen. The relationship between variable selection and data augmentation and a method for prediction. *Technometrics*, 16:125–127, 1974.

[2] Sylvain Arlot. V-fold cross-validation improved: V-fold penalization, February 2008. arXiv:0802.0566v2.

[3] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79 (electronic), 2010.

[4] Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2003.

[5] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for

	2	4	6	8	10	12	2	4	6	8	10	12	2	4	6	8	10	12
PenVF+	0	0	0	0	0	0	2	8	10	10	10	10	8	2	0	0	0	0
PenVF	2	0	0	0	0	0	3	7	9	10	10	10	5	3	1	0	0	0
Ideal	0	0	0	0	0	0	2	3	4	4	4	4	8	7	6	6	6	6

Table 4: Wins, draws and losses for CART using $m = 500$. Note that `PenVF+` wins 8 times for $V = 2$ and 2 times for $V = 4$.

optimal margin classifiers. In *Proceedings of the 5th Annual Conference on Computational Learning Theory*, pages 144–152, New York, NY, USA, 1992. ACM Press.

[6] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and regression trees*. Wadsworth and Brooks, 1984.

[7] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and regression trees*. Wadsworth and Brooks, 1984.

[8] Leo Breiman and Philip Spector. Submodel Selection and Evaluation in Regression. The X-Random Case. *International Statistical Review*, 60(3):pp. 291–319, 1992.

[9] Prabir Burman. A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika*, 76(3):503–514, 1989.

[10] W.R. Burrows. Cart regression models for predicting uv radiation at the ground in the presence of cloud and other environmental factors. *Journal of Applied Meteorology*, 36(5):531–544, 1997.

[11] Ming-Wei Chang and Chih-Jen Lin. Leave-One-Out Bounds for Support Vector Regression Model Selection. *Neural Computation*, 17:1188–1222, May 2005.

[12] Vladimir Cherkassky and Yunqian Ma. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks.*, 17:113–126, January 2004.

[13] P.A. Chou, T. Lookabaugh, and R.M. Gray. Optimal pruning with applications to tree-structured source coding and modeling. *Information Theory, IEEE Transactions on*, 35(2):299–315, 1989.

[14] Harris Drucker, Chris J. C. Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. In *Advances in Neural Information Processing Systems 9*, pages 155–161, Cambridge, MA, 1997. MIT Press.

[15] B. Efron. Bootstrap methods: another look at the jackknife. *Ann. Statist.*, 7(1):1–26, 1979.

[16] A. Frank and A. Asuncion. UCI Machine Learning Repository, 2010.

[17] Seymour Geisser. The predictive sample reuse method with applications. *J. Amer. Statist. Assoc.*, 70:320–328, 1975.

[18] S. Gey and E. Nedelec. Model selection for cart regression trees. *Information Theory, IEEE Transactions on*, 51(2):658–670, 2005.

[19] I. Guyon, B. Boser, and V. Vapnik. Automatic Capacity Tuning of Very Large VC-dimension Classifiers. In *Advances in Neural Information Processing Systems*, volume 5, pages 147–155, 1993.

[20] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5:1391–1415, 2004.

[21] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. Springer Series in Statistics. Springer-Verlag, New York, 2001. Data mining, inference, and prediction.

[22] Wenwu He, Zhizhong Wang, and Hui Jiang. Model optimizing and feature selecting for support vector regression in time series forecasting. *Neurocomputing*, 72:600–611, December 2008.

[23] Vladimir Koltchinskii. Local Rademacher complexities and oracle inequalities in risk minimisation. *Ann. Statist.*, 34(6):2593–2656, 2006.

[24] James T. Kwok. Linear Dependency between epsilon and the Input Noise in epsilon-Support Vector Regression. In *ICANN*, volume 2130 of *Lecture Notes in Computer Science*, pages 405–410. Springer, 2001.

[25] P. Liang and N. Srebro. On the interaction between norm and dimensionality: Multiple regimes in learning. In *International Conference on Machine Learning (ICML)*, 2010.

[26] Colin L. Mallows. Some comments on C_p. *Technometrics*, 15:661–675, 1973.

[27] Michinari Momma and Kristin P. Bennett. A pattern search method for model selection of support vector regression. In *In Proceedings of the SIAM International Conference on Data Mining*. SIAM, 2002.

[28] Radford Neal. Assessing Relevance Determination Methods Using DELVE. In *Neural Networks and Machine Learning*, pages 97–129, 1998.

[29] Chong-Jin Ong, Shiyun Shao, and Jianbo Yang. An improved algorithm for the solution of the regularization path of support vector machine. *Trans. Neur. Netw.*, 21:451–462, March 2010.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Duchesnay E. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[31] J.D. Rodriguez, A. Perez, and J.A. Lozano. Sensitivity analysis of k-fold cross validation in prediction error estimation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(3):569–575, 2010.

[32] B. Schölkopf and A. Smola. *Learning with kernels*. MIT press, Cambridge (MA), 2002.

[33] Jun Shao. Linear model selection by cross-validation. *J. Amer. Statist. Assoc.*, 88(422):486–494, 1993.

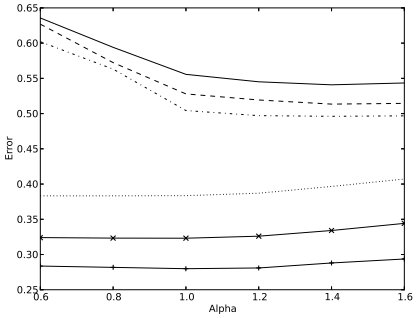
[34] A. Smola, N. Murata, B. schölkopf, and K. R. müller. Asymptotically Optimal Choice of ϵ-Loss for Support Vector Machines. In L. Niklasson, M. bodén, and T. Ziemke, editors, *Proceedings of the 8th International Conference on Artificial Neural Networks*, Perspectives in Neural Computing. Springer Verlag, 1998.

[35] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, August 2004.

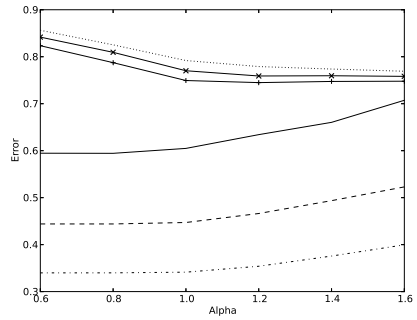
[36] V. N. Vapnik and A. Ya. Chervonenkis. Ordered risk minimization. *Automation and Remote Control*, 35:1226–1235, 1974.

[37] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag, New York, 1995.

[38] K.D. Wernecke, K. Possinger, G. Kalb, and J. Stein. Validating classification trees. *Biometrical journal*, 40(8):993–1005, 1998.

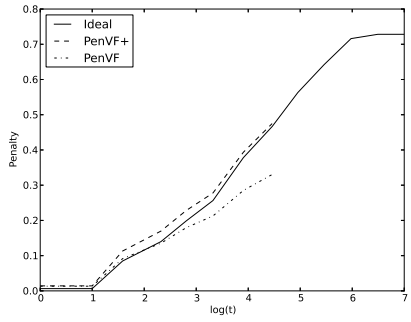


(a) SVR, `abalone` (top) and `slice-loc` (bottom)

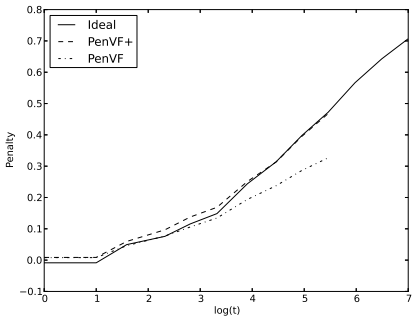


(b) CART, `winequality-white` (top) and `slice-loc` (bottom)

Figure 2: The variation in error with α for some sample datasets with $V = 10$. The plots are ordered from top to bottom: $m = 50, 100, 200$, for example for `winequality-white` and CART the respective curves are dotted, solid with crosses and solid with pluses. With `abalone` and the SVR, and `winequality-white` with CART overpenalisation improves results, however it makes them worse with `slice-loc` for both the SVR and CART.



(a) $m = 50$



(b) $m = 100$

Figure 3: The variation in penalty for CART in the "ideal" case relative to `PenVF` $\alpha = 1.0$, and PenVF+ with $V = 2$ and `abalone`. `PenVF+` estimates the ideal error well across a range of t's, whereas `PenVF` underestimates it for large t.

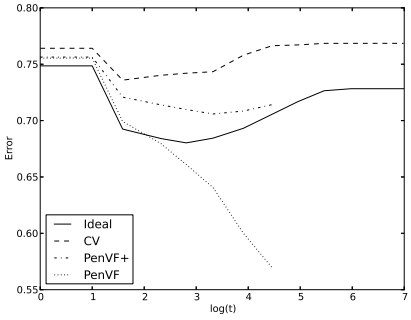


Figure 4: The error for CART in the "ideal" case relative to `PenVF` $\alpha = 1.0$, and PenVF+ with $V = 2$ and `abalone`, $m = 50$. `PenVF` provides a poor estimation of the error for large values of t. `PenVF+` gives better error estimates but results in the selection of larger trees than VFCV.