A practical guide to model selection

Isabelle Guyon
ClopiNet, Berkeley, CA 94708, USA
email: isabelle@clopinet.com

1. Introduction

This chapter is dedicated to students and practitioners who are new to the filed of machine learning or data mining and want to:

- quickly get results on some applications of interest,

- understand the basic principles of the methods available to eventually be able to customize them, and

- gain enough vocabulary and concepts to be able to read papers addressing more advanced topics.

Slides accompanying this material are available at `http://clopinet.com/isabelle/Projects/MLSS08/`. We focus in the chapter on the problem of **model selection**. The rest of the topics covered in the class are developed in tutorials on **feature selection** (Guyon and Elisseeff, 2003; Guyon et al., 2006a) and **causality** (Guyon et al., 2007a; Guyon, 2008; Guyon et al., 2009a).

Nowadays, there are many machine learning or data mining packages providing highly optimized implementations of leading algorithms, including commercial platforms like SAS and SPSS and freeware packages like Weka[1], R[2], Lush[3] and several Matlab libraries like the Spider[4] and CLOP[5]. The MLOSS open source repository[6] indexes such valuable resources. In this context, the task of practitioners has shifted from that of identifying and implementing algorithms to that of learning how to use such packages and selecting the best model. The problem of model selection still remains largely the user's responsibility.

Best practices for model selection have emerged over the years, grounded in a wide variety of theories (regularization, Bayesian priors, MDL, structural risk minimization, bias/variance tradeoff, etc.; see Appendix A). Interestingly, all those theories converge towards the same principle stated already in the 14th century by William of Ockham: "Pluralitas non est ponenda sine neccesitate", which prescribes limiting model complexity to the minimum necessary to explain the data, or *shave off unnecessary parameters* (Ockham's razor). Indeed, as illustrated in Figure 1, attempts to improve prediction performance on training data by increasing model complexity may lead to data "over-fitting": the good

---

1. `http://www.cs.waikato.ac.nz/ml/weka/`
2. `http://www.r-project.org/`
3. `http://lush.sourceforge.net/`
4. `http://www.kyb.mpg.de/bs/people/spider/`
5. `http://clopinet.com/CLOP/`
6. `http://mloss.org/software/`

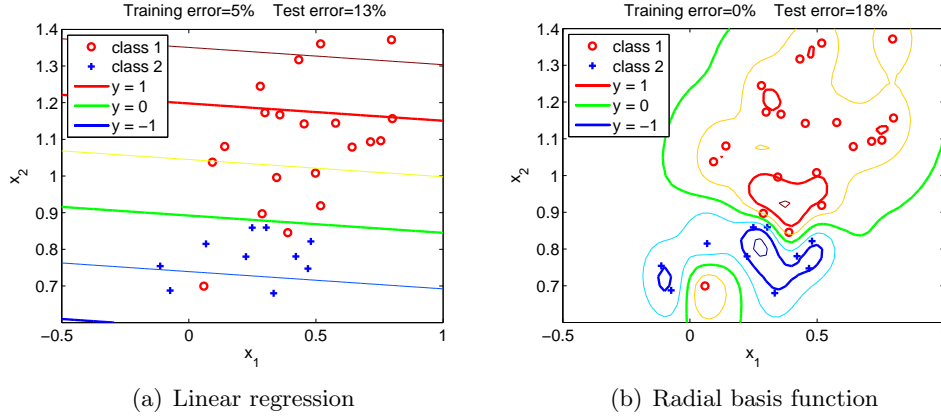| | Training error=5%   Test error=13% | | Training error=0%   Test error=18% |
| (a) Linear regression | | (b) Radial basis function | |

Figure 1: **Over-fitting.** The problem of over-fitting is schematically illustrated on a classification example with two input variables x_1 and x_1. The example includes 30 training examples (shown on the figure), drawn at random from a Gaussian mixture, and 300,000 test examples (not shown), drawn from the same distribution. The decision boundaries (shown in green) were computed with the CLOP package (Saffari and Guyon, 2006), using the following models (described in Section 3.1): (a) `kridge('degree=1')` (linear ridge regression), (b) `kridge({'degree=0', 'gamma=100'})` Gaussian kernel classifier with $\sigma = 0.1$. This second classifier has a lower training error but a larger test error. It overfits the training data.

training performance is deceiving, it does not extend to new test data; the model does not "generalize" to new unknown cases. Little more needs to be understood to perform good model selection. The remainder of this chapter equips the reader with means of evaluating *model complexity* and estimating relative *model performance*.

Section 2 defines more precisely the scope of the chapter and gives examples of model selection problems. Section 3 provides a high-level description of three algorithms, which have been performing consistently well in benchmarks, and which belong to every state-of-the-art toolkit. Section 4 equips the reader with basic model selection strategies. Section 5 addresses the problem of data representation and feature selection. The chapter is complemented with an appendix giving a tour of learning theory for beginners.

2. What is model selection?

Modeling encompasses a wide variety of methods in science and technology, with the purpose of explaining phenomena and making predictions. The focus of the chapter is on statistical models, which can be optimized with some training data to perform prediction tasks. Examples of application are found in pattern recognition, medical diagnosis, text classification, marketing, etc. For simplicity, the data are assumed to have been preprocessed adequately into a feature representation, but the principles are rather general and can easily be adapted to other situations. We further restrict ourselves to the problem of "supervised learning" described below.

2

2.1 Supervised learning

For the purpose of this chapter, data are formatted as a matrix $X = [x_i^k]$ of dimension (p, n), $i = 1, ...p$, $k = 1, ...n$, the p lines representing samples and the n columns representing features[7]. The samples (also called patterns, examples, or instances) may represent images, spectra, texts, patients, or customers, depending on the application. The features (also called attributes or variables) may represent colors of an image pixel, frequency coefficients, demographic attributes, risk factors, survey responses, etc. We are particularly interested in cases in which an additional data column (of dimension $(p, 1)$) called *target*, denoted as $\mathbf{y} = [y^k]$, $k = 1, ...n$, encodes an outcome to be predicted. In classification problems, the target is a category (*e.g.,* one of twenty six letters in a handwriting recognition problem, topics in a text classification problem, disease in a medical diagnosis problem) and, in regression problems, the target is a continuous value (life expectancy, expected revenue, etc.). The task of the model is to take input values $\mathbf{x} = [x_1, x_2, ..., x_n]$ (a line of the data matrix, the pattern index being omitted) and predict the corresponding outcome y. Available are *m training example* pairs $\{\mathbf{x}^k, y^k\}$, $k = 1, ...m$, to adjust the parameters of the model ("train" the model). Such trainable models are called "learning machines". The last $(p-m)$ example pairs are reserved as *test examples* to evaluate the prediction performance of the model on new (unseen) examples. This problem is referred to "supervised learning", to indicate that the learning machine is provided with a teaching signal y^k with every training example $\mathbf{x}^k$. In contrast, in "unsupervised learning" problems, no target y is available and the task of the model is to uncover some latent structure in data ("natural" classes like in clustering; directions in space explaining most of the variance in data like in Principle Component Analysis (PCA), etc.).

We treat only the problem of supervised learning in this chapter. In this context, a model can be thought of as a function f approximating data $y \simeq f(\mathbf{x})$ and supervised learning encompasses problems of **classification**, **posterior density estimation** $(P(\mathbf{y}|X))$, **interpolation**, and **regression**. Learning may be formalized as an optimization problem, that of minimizing an *expected risk* (also called "generalization error") that is the mathematical expectation of a loss function $\mathcal{L}(y, f(\mathbf{x}))$:

$$R[f] = \int \mathcal{L}(y, f(\mathbf{x}) \; dP(\mathbf{x}, y) \; . \tag{1}$$

The summation runs over an infinite number of examples drawn from an unknown distribution $P(\mathbf{x}, y)$ (identically and independently drawn samples). Obviously, the expected risk can never be known exactly, it can only be approximated using available data samples. Defining good surrogate cost functions for training, model selection, and performance evaluation is one of the problems addressed in this chapter. **Training data must be used for adjusting all parameters, hyper-parameters and perform model selection**. The test set must remained untouched during the process of model selection. It serves to evaluate the final model. See section 4.

---

7. Boldface characters indicate vectors and uppercase characters indicate matrices. We do not have a special notation for random variables. When variables are associated with a distribution, they are considered random variables. Super-indices are used for patterns and sub-indices for features. This allows us, without ambiguity, to omit one of the two indices. We adopt the Matlab convention to separate elements by commas in row vectors and semi-columns in column vectors.

2.2 Model selection

We regroup under "model selection" number of problems, including: (i) **selecting the best features**, (ii) **selecting the best preprocessing** (data normalization, mathematical transformations of feature space) (iii) **selecting the best learning machine** (*e.g.,* neural network, linear model, kernel method, classification or regression tree, Bayes net, nearest neighbor, etc.), (iv) **selecting the best set of hyper-parameters** (number of layers and hidden units in a neural network, kernel type and smoothing parameter in kernel methods, number of neighbors in the nearest neighbor method, etc.).

These various problems are formalized in a unified way, using a hyper-parameter notation. We denote by $\boldsymbol{\theta} = [\theta_1, \theta_2, ...\theta_h]$ the vector of hyper-parameters, which may include: indicators of presence or absence of some features (feature selection), indicators of preprocessing choices, indicators of learning machine choices, and learning machine hyper-parameters. For each learning machine[8], we denote by $\boldsymbol{\alpha}$ its parameters, subject to training. So, in our notation, $f(\mathbf{x}; \boldsymbol{\alpha}, \boldsymbol{\theta})$ represents a compound model including eventually all the components (i-iv) described above.

The separation between parameters and hyper-parameters may seem arbitrary. After all, why not carrying out a global optimization of both hyper-parameters and parameters, bypassing the problem of model selection? There are both practical and fundamental reasons for making this distinction. In practice, as noted in the introduction, we often want to try and to compare the latest and greatest algorithms available in machine learning or data mining packages, which are not integrated in a global optimization program. An easy way to perform such explorations is to fix $\boldsymbol{\theta}$ and use off-the-shelf algorithms to adjust $\boldsymbol{\alpha}$. But more fundamentally, as explained in Section 4 and in Appendix A.2, introducing a hierarchy of parameters is exploited by multi-level inference algorithms to alleviate the problem of over-fitting.

3. Three useful algorithms

It is difficult to learn how to perform model selection without having a basic understanding of the models to select from. This section covers three representative machine learning algorithms, which have been highly successful in benchmarks (Guyon et al., 2006b, 2007b, 2008, 2009b): kernel methods, ensembles of decision trees, and the Naïve Bayes algorithm. Within a model family, the algorithms are mostly interchangeable (have similar computational burden and performance). Across model families, the algorithms present different practical and computational advantages, depending on the type data and differences in performance may be indicative of particularities of the data structure. Figure 2 illustrates the variety of types of decision boundaries obtained with these methods.

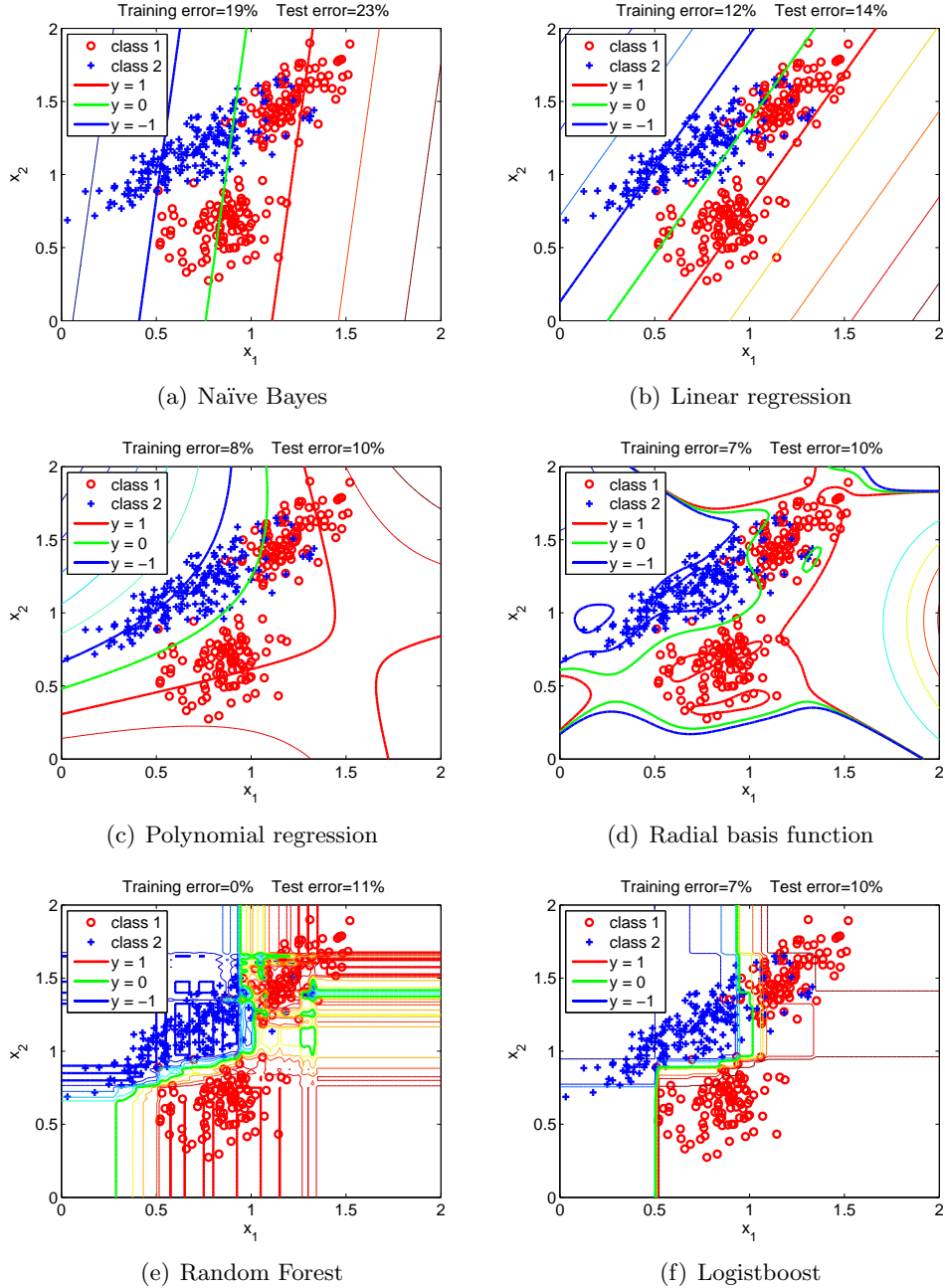8. The other modules (feature selection, preprocessing) may also have adjustable parameters.

Figure 2: **Examples of classification boundaries.** The examples include 300 training examples (shown on the figure), drawn at random from a Gaussian mixture, and 300,000 test examples (not shown), drawn from the same distribution. The decision boundaries (shown in green) were computed with the CLOP package (Saffari and Guyon, 2006), using the following models: (a) `naive`, a naïve Bayes Gaussian classifier, (b) `kridge('degree=1')`, linear ridge regression, (c) `kridge('degree=2')`, second degree polynomial regression, (d) `kridge({'degree=0', 'gamma=1'})`, Gaussian kernel classifier with $\sigma = 1$, (e) `rf('units=50')`, Random Forest of 50 non-pruned trees, (f) `logistboost('units=50')`, Logitboost, boosted tree logistic regression, with 50 shallow trees.

5

3.1 Kernel methods: kernel ridge regression

Kernel methods regroup algorithms in which the predictive function is of the form[9]:

$$f(\mathbf{x}) = \sum_{k=1}^{t} \alpha_k \ K(\mathbf{x}, \mathbf{x}^k) \quad , \tag{2}$$

where the sum runs over training examples and $K(\mathbf{x}, \mathbf{x}^k)$ is a so-called kernel function, which can be thought of as a similarity measure between $\mathbf{x}$ and $\mathbf{x}^k$. The two most widely used kernels are:

- the Gaussian kernel $K(\mathbf{x}, \mathbf{x}^k) = \exp\left(-\|\mathbf{x} - \mathbf{x}^k\|^2/\sigma^2\right)$, and

- the polynomial kernel $K(\mathbf{x}, \mathbf{x}^k) = (1 + \mathbf{x} \cdot \mathbf{x}^k)^q$,

of parameters σ (kernel width) and q (degree). Both kernel types yield *universal approximators*: for sufficiently large degree or small kernel width, any training set can be learned without error, using the function of Equation 2. The Gaussian and polynomial kernels can be combined in a generalized kernel with two hyper-parameters:

$$K(\mathbf{x}, \mathbf{x}^k) = (1 + \mathbf{x} \cdot \mathbf{x}^k)^q \ \exp\left(-\|\mathbf{x} - \mathbf{x}^k\|^2/\sigma^2\right) \ .$$

A number of algorithms rely on the property of some kernels to be factorizable. In particular, all semi-definite positive kernels (like the three kernels above mentioned) admit a factorization of the type:

$$K(\mathbf{x}, \mathbf{x}^k) = \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{x}^k) \ ,$$

where the "·" notation denotes a dot product or scalar product and $\boldsymbol{\phi}(.)$ is a continuous or discrete function of $\mathbf{x}$. In the discrete case, $\boldsymbol{\phi}(.)$ can be thought of as a feature vector (of finite or infinite dimension). For concreteness, let us consider the case of the polynomial kernel of degree 2 and input vectors $\mathbf{x} = [x_1, x_2]$ of only two dimensions. Then:

$$
\begin{aligned}
K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x} \cdot \mathbf{x}')^2 \ &= \ (1 + x_1 \ x_1' + x_2 \ x_2')^2 \\
&= \ 1 + 2x_1 x_1' + 2x_2 x_2' + 2x_1 x_2 x_1' x_2' + (x_1 x_1')^2 + (x_2 x_2')^2 \\
&= \ [1, 2x_1, 2x_2, 2x_1 x_2, x_1^2, x_2^2] \cdot [1, 2x_1', 2x_2', 2x_1' x_2', x_1'^2, x_2'^2] \ .
\end{aligned}
$$

We see that the kernel is in that case the dot product of two feature vectors composed of products of the original features.

Algorithms taking advantage of such factorization (known as "kernel trick") implicitly optimize a prediction function in $\boldsymbol{\phi}$-space, linear in its parameters w_i:

$$f(\mathbf{x}) = \sum_i w_i \phi_i(\mathbf{x}) = \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x})$$

---

9. For regression problems, $f(\mathbf{x})$ is used for predictions. For two-class classification problems, classification is carried out according to the sign of $f(\mathbf{x})$. Generalizations to the multi-class problem are handled in various ways, (see *e.g.,* Mitchell, 1997).

The weight vector $\mathbf{w}$, which may be of infinite dimension, is related as follows to the α^k parameters:

$$\mathbf{w} = \sum_{k=1}^{t} \alpha^k \boldsymbol{\phi}(\mathbf{x}^k) \ .$$

A more formal treatment of kernel methods is given in Appendix A.6. Note that in the case of the linear kernel $K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x} \cdot \mathbf{x}')$, the function $f(\mathbf{x})$ is simply a linear function:

$$f(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i + w_0 \ .$$

One popular optimization method in this framework is the Support Vector Machine algorithm (SVM) (Boser et al., 1992), which provides a sparse solution: the sum in Equation 2 runs only over a small subset of the training examples, called "support vectors". Others include kernel logistic regression, kernel ridge regression, and LSSVM. See (Schoelkopf and Smola, 2002) for a comprehensive treatment.

We present here the **kernel ridge regression algorithm**, which does not provide a sparse solution, but combines the advantages of involving only a simple matrix inversion (gradient descent versions also exist and are needed to process large sample sizes, see "weight decay" in Appendix A.2), allowing to optimize the *shrinkage* or *regularization* parameter (ridge) with a single pass through the training data (see Appendix A.6 for a definition of *regularization*), being quite insensitive to noise in data, and being free of patent protection. The algorithm works both for regression and classifications problems (it treats classification as a regression problem; the predicted values of $f(\mathbf{x})$ need to be thresholded to obtain classification decisions). The ridge regression algorithm optimizes the square loss $\mathcal{L}(\boldsymbol{\phi}(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2$ and uses a penalty term (regularizer) proportional to $\|\mathbf{w}\|^2$ to bias the solution towards weight vectors of small Euclidean norm in $\boldsymbol{\phi}$-space. Principled motivations for using such a bias are found in Appendix A.

The regularized risk functional (training cost function) being optimized is defined as:

$$R_{reg}[f] = \sum_{k=1}^{t} (f(\mathbf{x}^k) - y^k)^2 + \gamma \|\mathbf{w}\|^2 \ . \tag{3}$$

The solution minimizing $R_{reg}[f]$ is:

$$\boxed{\boldsymbol{\alpha} = K_\gamma^{-1} \mathbf{y}}$$

where $\boldsymbol{\alpha} = [\alpha^1; \alpha^2; ... \alpha^t]$ and $\mathbf{y} = [y^1; y^2; ... y^t]$ are column vectors and K_γ is a (t, t) regularized "kernel" matrix, defined as $K_\gamma = K + \gamma I$, where $K = [K(\mathbf{x}^k, \mathbf{x}^h)]$, $k = 1 : t, h = 1 : t$, is the (t, t) *kernel matrix* and I is the (t, t) identity matrix. It is computationally efficient to perform an eigen value decomposition

$$K = UDV \ ,$$

where U and V are orthogonal matrices and D is a diagonal matrix. Then,

$$K_\gamma^{-1} = V^T (D + \gamma I)^{-1} U^T \ .$$

The inversion of a diagonal matrix being trivial, once the eigen decomposition of K is computed, several values of γ can be tried at minimal additional computational cost. Further, evaluating the solutions with the leave-one-out estimator (see Section 4) is quite inexpensive because there is a closed form solution (the *virtual leave-one-out estimator*):

$$R_{loo}[f] = \frac{1}{t} \sum_{k=1}^{t} (\epsilon_{loo}^k)^2 \ ,$$

with

$$\epsilon_{loo}^k = (f(\mathbf{x}^k) - y^k)/\left(1 - [KK_\gamma^{-1}]_{kk}\right) \ ,$$

where the notation $[\cdot]_{kk}$ designates the kth diagonal element.

The solution for the linear kernel may be computed either in "direct space" ($\mathbf{w}$-space) or in "dual space" ($\boldsymbol{\alpha}$-space). Depending on the dimension of the data matrix X, one of the two is more efficient computationally[10]. Define $X1 = [X, \mathbf{1}]$ by adding an extra columns of ones to matrix X. Then invert one of the two matrices $G_\gamma = X1^T X1 + \gamma I$ of dimension $(n+1, n+1)$ or $K_\gamma = X1 \ X1^T + \gamma I$ of dimension (p, p), using the same type of eigen decomposition mentioned above. Then the solution is given by $f(\mathbf{x}) = [\mathbf{x}, 1]X1^T K_\gamma^{-1}\mathbf{y} = [\mathbf{x}, 1]G_\gamma^{-1}X1^T\mathbf{y}$.

Details on this algorithm and its derivation can be found in Appendix A. It is implemented in the CLOP package (Saffari and Guyon, 2006). Gavin Cawley won several challenges using a variant of this method called weighted least-square support vector machines (Cawley, 2006). The algorithm optimizes the bias value (rather than just adding a feature with value 1) and uses a different ridge value for every feature to automatically scale the features according to relevance.

3.2 Ensemble of tree classifiers: Logitboost

Classification and regression trees are a family of models widely used in data mining and statistics. They perform recursive partitioning of the data along the axes of the variables/features and make a piecewise constant estimate in each domain defined by the partition. Like kernel methods, they are *universal approximators*. An example of classification tree (decision tree) is shown in Figure 3. The problem is to decide whether or not to play golf. There are three features (*outlook*, *humidity*, and *windy*) and the target variable is the decision play/don't play. The nodes indicate the fraction of training examples in the two classes, which remain after each partial decision. The terminal nodes are "pure": they include only examples of one class. New examples, not used for training, can be classified by "putting them down the tree" and performing the decision, when a terminal node is reached, according the the class of the training examples falling into that node. There are various algorithms to train decision trees, generally growing the tree from the root node and performing a greedy optimization by selecting features to partition training data in a way that improves "node purity" (see *e.g.,* Breiman et al., 1984).

Decision trees provide decision functions, which are easily understood in terms of rules. For instance, in the example of Figure 3, the rules are:

---

10. However, to compute the leave-one-out estimate, K_γ must be inverted in any case.
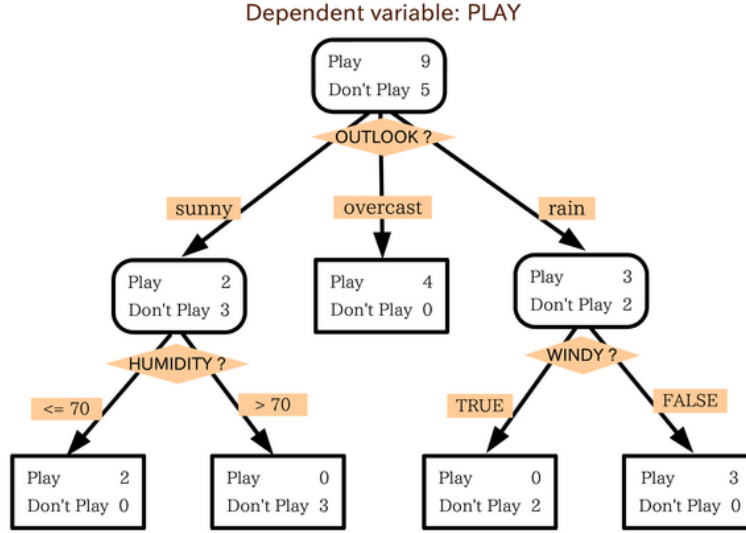
Figure 3: **Example of decision tree.** *Figure reprinted from Wikipedia; author: Toshihiro Kita*

if OUTLOOK=sunny and HUMIDITY≤70 then DECISION=play
if OUTLOOK=sunny and HUMIDITY>70 then DECISION=don't play
if OUTLOOK=overcast then DECISION=don't play
if OUTLOOK=rain and WINDY=yes then DECISION=don't play
if OUTLOOK=rain and WINDY=no then DECISION=play

Large ensembles of shallow trees (up to three layers) have obtained very competitive results in benchmarks (at the expense of loosing in data understanding capability). The two most prominent ensemble methods are bagging (Breiman, 1996) and boosting (Freund and Schapire, 1996). Bagging is a parallel ensemble method (all trees are built independently from training data subsets), while boosting is a serial ensemble method (trees complementing each other are progressively added to decrease the residual error). Random Forests (RF) (Breiman, 2001) are a variant of bagging methods in which both features and examples are subsampled. Boosting methods come in various flavors including Adaboost, Gentleboost, and Logitboost. The original algorithm builds successive models (called "weak learners") by resampling data in a way that emphasizes examples harder to learn. Newer versions use a weighting scheme instead of resampling (Friedman, 2000).

We describe one of the variants: the Logitboost algorithm (Friedman et al., 2000). It is used to train discriminant functions $F(\mathbf{x})$ capable of estimating posterior probabilities $P(y = 1|\mathbf{x})$ for classification problems, ensuring that the predictions are confined between 0 and 1. The logistic model serves that purpose:

$$F(\mathbf{x}) = 0.5 \left(\log P(y = 1|\mathbf{x}) - \log P(y = -1|\mathbf{x}) \right)$$

or, after inversion:

$$p(\mathbf{x}) = P(y = 1|\mathbf{x}) = \frac{e^{F(\mathbf{x})}}{e^{-F(\mathbf{x})} + e^{F(\mathbf{x})}} .$$

The method uses the logistic loss, which is the negative Binomial log likelihood:

$$\mathcal{L}(F(\mathbf{x}), y) = \log(1 + e^{-2yF(\mathbf{x})}) = -[y^* \log p(\mathbf{x}) + (1 - y^*) \log(1 - p(\mathbf{x}))] ,$$

9

with $y \in \{-1, +1\}$ and $y^* = (y+1)/2 \in \{0, 1\}$.

The outline of the algorithm, is as follows:

1. Start with $F(\mathbf{x}^k) = 0$ and $p(\mathbf{x}^k) = 1/2$, $k = 1, .., t$ (t number of training examples).

2. Repeat for $\ell = 1, ..., L$ (L number of weak learners):

   a. Compute the weights and working response
   $w^k = p(\mathbf{x}^k)(1 - p(\mathbf{x}^k))$,
   $z^k = \frac{y^{*k} - p(\mathbf{x}^k)}{p(\mathbf{x}^k)(1 - p(\mathbf{x}^k))}$.

   b. Fit the function $f_\ell(\mathbf{x})$, using the tree-based learner, by a weighted least-squares regression of z^k to $\mathbf{x}^k$ using weights w^k.

   c. Update $F(\mathbf{x}^k) \leftarrow F(\mathbf{x}^k) + \frac{\nu}{2} f_\ell(\mathbf{x}^k)$, $0 < \nu \leq 1$, and $p(\mathbf{x}^k) = \frac{e^{F(\mathbf{x}^k)}}{e^{-F(\mathbf{x}^k)} + e^{F(\mathbf{x}^k)}}$.

3. Output the discriminant function $F(\mathbf{x}) = \frac{\nu}{2} \sum_{\ell=1}^{L} f_\ell(\mathbf{x})$ and classify according to its sign.

The algorithm has two hyper-parameters that may be adjusted by cross-validation (see Section 4): the *shrinkage* or *regularization* parameter ν and the number of weak learners L. Roman Lutz won several challenges using this method (Lutz, 2006), which also fared well in the 2009 KDD cup (Guyon et al., 2009b).

3.3 Naïve Bayes classifier

The naïve Bayes classifier is a simple yet powerful method, which makes independence assumptions between variables. The resulting classifier is biased if this assumption is violated, but very robust against over-fitting. We describe the algorithm for discrete valued inputs for a two-class classifier. We refer the reader to tutorials and textbooks for the case of continuous variables, multiple classes, or extensions to the case of regression (Mitchell, 1997; Duda et al., 2001; Hastie et al., 2000).

Bayes classifiers follow the rule: classify pattern $\mathbf{x}$ in class 1 if $P(y = 1|\mathbf{x}) > P(y = 0|\mathbf{x})$ and in the other class otherwise (we use 0/1 target values for notational simplicity). According to Bayes rule $P(y = c|\mathbf{x}) = P(\mathbf{x}|y = c)P(y = c)/P(\mathbf{x})$ with $c = \in \{0, 1\}$. Because $P(\mathbf{x})$ does not affect the classification decision, an equivalent classification rule is obtained with: classify pattern $\mathbf{x}$ in class 1 if $P(\mathbf{x}|y = 1)P(y = 1) > P(\mathbf{x}|y = 0)P(y = 0)$ and in the other class otherwise. Hence, we can build a discriminant function:

$$f(\mathbf{x}) = \log[P(\mathbf{x}|y = 1)P(y = 1)] - \log[P(\mathbf{x}|y = 0)P(y = 0)] . \tag{4}$$

Making independence assumptions between the variables x_i (given y), we have, for $c = \in \{0, 1\}$:

$$P(\mathbf{x}|y = c) = \prod_i P(x_i|y = c) .$$

Hence, our discriminant function of Equation 4 becomes:

$$\boxed{f(\mathbf{x}) = \sum_{i=1}^{n} \log[P(x_i|y = 1)/P(x_i|y = 0)] + b}$$

10

with $b = log[P(y = 1)/P(y = 0)]$.

The prior distributions $P(y = 1)$ and $P(y = 0)$ may be estimated by p_1 and p_0, the fractions of examples in the positive and the negative class respectively. The likelihoods $P(x_i = a|y = c)$ may be estimated by p_{aci} the fraction of times variable x_i takes value a when the corresponding target value is $y = c$. For the case special of 0/1 binary input variables, the discriminant function of Equation 3.3 can be brought back to a linear function:

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i x_i + w_0$$

with $w_i = \log(p_{11i}/p_{10i}) - \log(p_{01i}/p_{00i})$ and $w_0 = \sum_i \log(p_{01i}/p_{02i}) + log(p_1/p_0)$.

The naïve Bayes classifier is implemented in CLOP for binary and continuous inputs (Saffari and Guyon, 2006). Marc Boullé obtained excellent results in several challenges using enhancements to this method (Boullé, 2007), featuring feature construction and selection, and ensemble learning.

3.4 Filters for model selection

Before we spend the rest of the chapter explaining methods for choosing a model based on evaluations of model performance, we would like to point out that *model choice may often be guided first by heuristic rules*, referred to as "model selection filters". There are benefits to applying such model filtering or screening process: by reducing upfront the number of models considered (or by prioritizing them) one reduces both the risk of over-fitting (statistical complexity) and computational complexity. The following guidelines are often helpful:

1. **Naïve Bayes** is the method which is least prone to over-fitting and least computationally expensive. In addition, the weights of the classifier are indicators of univariate feature relevance, which aids data understanding. For those reasons, we recommend to always include it as baseline method. If the number of features is very large compared to the number of examples (several orders of magnitude) and there are not enough training examples to estimate performance by cross-validation (less than 30 examples), it is reasonable to just use Naïve Bayes (or an equivalent method making independence assumptions between variables). Because of its simplifying assumptions, Naïve Bayes may underfit data (poor prediction accuracy even on training data).

2. **Linear ridge regression** and **ensembles of stumps** (decision trees of depth 1) have also very low statistical complexity but may provide a better fit. These methods give good solutions to most pattern recognition problems. Use either one, basing your choice on the following computational considerations. If either the number of examples t is small or the number of features n is small (less than a few thousand), use *linear ridge regression* carrying out matrix inversion as explained in Section 3.1 (complexity cubic in $\min(n, t)$). If the number of examples is large (more than a few thousands) apply gradient descent to compute the linear ridge regression solution (see "weight decay" in Appendix A.2) or use *ensembles of decision stumps* (computational complexity linear in the number of features, examples, and "weak learners").

3. **Non-linear ridge regression** and **ensembles of deep trees** should be considered only if sufficient amounts of training data are available to perform *cross-validation* (see Section 4). Kernel methods handle well large feature spaces (up to hundreds of thousands) for samples sizes up to thousands. Ensembles of decision trees handle well very large number of examples (up to millions) but require feature subsampling (like in Random Forests) if the feature space is large.

4. Basic model selection strategies

We have been so far informally talking about **performance evaluation** and **over-fitting**. In this section, we formalize these notions and exploit them to devise practical model selection techniques. Performing model selection requires exploring hyper-parameter space with a search strategy (since evaluating the models for all hyper-parameter configurations may be infeasible), and using an evaluation function to compare the relative merit of the models. A good approximation of the **expected risk** (Equation 1) would make an ideal evaluation function. However, since we need to evaluate only relative merit, any monotonic function of an approximation of the expected risk would do.

In this section, we first formalize the notion of **statistical complexity** and introduce the **bias/variance tradeoff** and the concept of **guaranteed risk**. We then present a number of **evaluation functions** and **search techniques**. We finish with some recommendations.

4.1 Statistical complexity

As mentioned in the introduction, the notion of *model complexity* and the necessity to *limit it to a minimum as long as the data are explained*, was intuitively understood a long time ago and well stated in Ockham's razor principle.

In classical statistics, the idea is formalized in the bias/variance tradeoff, which was introduced to the machine learning community by (Gem). A predictive model f can be though of as an estimator of $E(y|\mathbf{x})$, the expected value of y given $\mathbf{x}$. As such, it has a bias and a variance. Much of classical statistics is devoted to the quest of unbiased estimators with small variance. However, in machine learning, **biased models** are sometimes preferred because they **may yield better generalization error**. We introduce the notation $f(\mathbf{x}; D)$ to denote a model trained on a dataset D of a given size t and the notation $E_D[\cdot]$ to denote the expected value taken over all datasets of size t drawn according to $P(\mathbf{x}, y)$. In can be shown that:

$$E_D[(f(\mathbf{x}; D) - E[y|\mathbf{x}])^2] = (E_D[f(\mathbf{x}; D)] - E(y|\mathbf{x}))^2 + E_D[(f(\mathbf{x}; D) - E_D[f(\mathbf{x}; D)])^2] \quad (5)$$

The first term of the right hand side of the equation is the square of the **bias** and the second term the **variance** of the estimator. Hence, if on average over all datasets of the same size $f(\mathbf{x}; D)$ is different from $E(y|\mathbf{x})$, the model is biased. But, even an unbiased estimator may yield a poor fit if it has a large variance. There is often an optimum compromise between bias and variance (bias/variance tradeoff), the least biased models having a large variance and vice versa. For instance, putting a bound on the degree q of a polynomial model introduces some bias, but reduces its variance.

Bias and variance can be exactly computed for some models (see in Appendix A.5 the calculation for the ridge regression model). But the bias/variance tradeoff remains a qualita-

tive guideline for many models for which bias and variance cannot be computed analytically (like neural networks or decision trees). Nonetheless, it serves as a justification for bagging methods (Breiman, 1996). If it were possible to use as predictive model $E_D[f(\mathbf{x}; D)]$ instead of $f(\mathbf{x}; D)$, the variance term in Equation 5 would vanish. The proposition of bagging is to resample with replacement the available training dataset of size t to create a large number of datasets D of the same size, train models $f(\mathbf{x}; D)$, then average them, in order to approximate $E_D[f(\mathbf{x}; D)]$. Bagging works best if the model class of $f(\mathbf{x}; D)$ is unbiased, because otherwise the bias term starts dominating the error in Equation 5.

Generally, as the training set size increases, the bias remains, but the variance vanishes to zero as $t \to \infty$. Hence, for a given value of t, **the variance is a measure of statistical complexity of the model**.

Statistical learning theory quantifies statistical complexity in a different way (Vapnik, 1998). The idea is that the *empirical risk* or *training error* $R_{tr}[f] = (1/t) \sum_{k=1}^{t} \mathcal{L}(f(\mathbf{x}^k), y^k)$ is a poor estimator of the *expected risk* $R[f] = \int \mathcal{L}(f(\mathbf{x}), y) \, dP(\mathbf{x}, y)$. For some models there can be large deviations between $R_{tr}[f]$ and $R[f]$. We want to guarantee that $R[f]$ will not exceed a certain value $R_{gua}[f, \delta]$, the *guaranteed risk*. With probability $(1 - \delta)$:

$$R[f] \leq R_{gua}[f, \delta] .$$

Such guaranteed risk is often decomposed into *training error* plus *penalty term* $\epsilon(C/t, \delta)$:

$$R_{gua} = R_{tr} + \epsilon(C/t, \delta) ,$$

where C is the "capacity" of the model. The penalty term is a bound on the error made by estimating $R[f]$ with $R_{tr}[f]$, and it monotonically increases with C/t. The capacity characterizes how easy it is to learn a variety of functions. For classification, for instance, Vapnik and Chervonenkis defined the capacity or "VC dimension" as the number points in general position, which can can be shattered (separated in all possible ways) (Vapnik and Chervonenkis, 1971). For a given number of training examples, while the training error is expected to decrease with increasing capacity, the generalization error might increase (see the schematic illustration of Figure 4-a). Therefore, **the capacity is a measure of statistical complexity of the model**.

Some learning algorithms (like SVMs or boosting) optimize a *guaranteed risk* rather than the *expected risk*, and therefore provide some guarantee of good *generalization* (see Appendix A.6). Algorithms derived in this way have an embedded model selection mechanism: they optimize a two part cost function, the first term being the training error (*empirical risk*) and the second term a *penalty* for more complex models, which vanishes as $t \to \infty$. Other very related penalty-based methods include the Bayesian MAP method (Appendix A.3), MDL (Appendix A.4), and regularization (Appendix A.6).

4.2 Evaluation functions

In the previous section, we have warned that the *empirical risk* (training error) may be a poor estimator of the *expected risk* (generalization error). We have mentioned that several algorithms minimize a penalty-based cost function like the *guaranteed risk* to alleviate the over-fitting problem. This can be considered an internal *embedded* model selection method. Are we done with model selection? Actually not because such *regularized* algorithms often

generate additional hyper-parameters (like the weighting coefficient of the penalty term in the two-part cost function). In addition, practitioners usually like to try a few different algorithms (*e.g.,* naïve Bayes, kernel methods, ensembles of decision trees). Hence, they still need to evaluate the relative generalization performances of several models. This section reviews various **evaluation functions** serving that purpose.

4.2.1 FINAL TESTING

Before addressing the problem of selecting an evaluation function for model selection, we review in this section the problem of the final testing of the model selected. Of p available examples in a study, m can be used for training parameters and performing model selection and $T = (p - m)$ should be reserved for testing. Test data should remain untouched throughout the study and should be used only for the final performance evaluation of a *single* model selected with the remainder of the data. Otherwise, it can be argued that it was to some extent used for training (selecting a model based on performance is a form of training). It is not always possible to reserve a large test set and have a sufficient size training set to train the model(s). However, if the model performance must be known within a certain confidence interval, then a minimum number of test examples must be reserved.

In (Guyon et al., 1998) we discuss what size test set gives good generalization error estimates for the classification problem, using guaranteed confidence intervals. We end up giving a rule-of-thumb, which can guide experimental design for classification problems, derived here in a simpler approximate manner. Assume that we have (from a previous study or from preliminary experiments) an estimate of the error rate that we anticipate our best model could reach on test data. It approximates the probability of error E of our final model. Further assume that the errors made on test examples will be *i.i.d.* (independently and identically distributed). In this case, the errors follow a Bernouilli process with a probability E of failure and $(1 - E)$ of success. The test error rate computed on T examples will have an expected value of E and its standard deviation will be:

$$\sigma = \sqrt{E(1 - E)/T} \ .$$

Assume that we want the error bar not to exceed 10% of E, we must solve $\sigma = 0.1E$ to get an estimate of the minimum number of test examples required. If we further assume than E is small compared to one $(1 - E \simeq 1)$, we obtain the rule-of-thumb:

$$\boxed{T = \frac{100}{E}} \ .$$

According to this formula, the number of test examples should be inversely proportional to E. This makes sense: in order to get a good error rate estimate, it is necessary to see at least a few errors. If the error rate is very low, it takes more test examples to see a few errors. If the classes have an unbalanced number of examples, the focus should be on getting at least a minimum number of examples of the positive class.

4.2.2 VALIDATION

We are now coming back to the problem of selecting an evaluation function for model selection. Selecting models by **validation** means splitting the m available training examples into
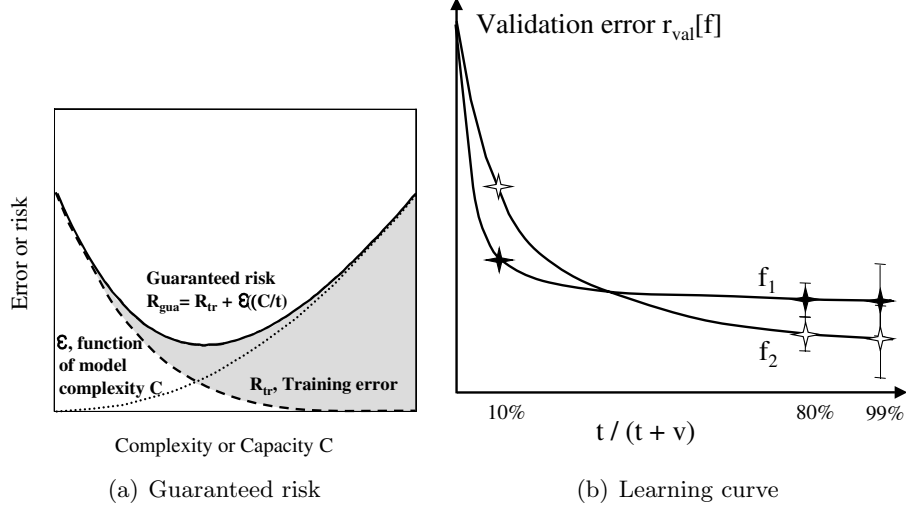
Figure 4: **Learning theory problems. Performance prediction** is the problem of estimating the generalization error $R[f]$. **Model selection** is the problem of adjusting the capacity or complexity of the models to the available amount of training data to avoid either under-fitting or over-fitting. Solving the performance prediction problem would also solve the model selection problem, but model selection is an easier problem. If we find an ordering index $r[f]$ such that for all pairs of functions $r[f_1] < r[f_2] \Rightarrow R[f_1] < R[f_2]$, then the index allows us to correctly carry out model selection. (a) **Guaranteed risk.** We schematically represent risk functionals for a fixed number of training examples t. Theoretical performance bounds (*guaranteed risk*, solid line) have been proposed as ranking indices. They often take the form $r[f] = R_{tr}[f] + \epsilon(C/t)$, where $R_{tr}[f]$ is the training error (dashed line) and $\epsilon(C/t)$ is the error bar (dotted line). The error bar is a function the ratio of the model class capacity (or complexity) C to the number of training examples t. While $R_{tr}[f]$ usually decreases when the capacity increases, $r[f]$ goes through an optimum and the hope is that this also corresponds to an optimum of $R[f]$. (b) **Validation error.** We schematically represent the validation error $r_{val}[f, v] = (1/v) \sum_{k=t}^{m} \mathcal{L}(f(\mathbf{x}^k), y)$ as a function of the fraction of examples used for training t/m, where $m = (t + v)$ is the total amount of training data available (*learning curves*). Training using all $m =$ training examples yields a model f whose performance $R[f]$ we are interested in. To evaluate the model, we use only t examples for training and test on v remaining validation examples. There is a tradeoff between using more examples for training or validation: In the example, two models f_1 and f_2 are compared and $R[f_2] < R[f_1]$. A bad model selection decision is made for a too small t (*e.g.*, $t/m = 10\%$) because the results are biased (the learning curves later cross). A bad model selection decision might be made for a too small v (*e.g.*, $t/m = 99\%$) because of the variance of $r_{va}[f]$ (large error bars). There may be a best bias/variance tradeoff (*e.g.*, $t/m = 80\%$).

t examples for adjusting the model parameters $\boldsymbol{\alpha}$ and $v = (m - t)$ examples for evaluating the relative generalization performance of the models. Assuming, without loss of generality, that the first t examples are used for training and the v last ones are used for validation, the resulting evaluation function is:

$$r_{val}[f, v] = \frac{1}{v} \sum_{k=t}^{m} \mathcal{L}(f(\mathbf{x}^k), y) .$$

Validation is the preferred method of performance estimation, for computational reasons and because error bar calculations are often possible. It should be used if m is large enough that:

- there is a value of v providing us with desired error bar guarantees on $r_{val}[f, v]$ (for classification problems, use *e.g.,* the calculator http://www.measuringusability.com/wald.htm or the rough estimator of Equation 4.2.1, which approximates the Binomial law with the Normal law),

- training on $t = (m - v)$ examples does not significantly degrade performance.

The tradeoff is illustrated in Figure 4. Using less data for training biases the performance estimate while using less data for validation increases the variance. Finding a theoretical optimum value for t/m is not easy because it depends on the shape of the learning curves of the models being compared (do they cross or not, and where?) and the validation error bars (which increase with the number of models being compared). In practice, because the statistical complexity of the first level of inference (learning the parameters of the models) is generally much higher than the statistical complexity of the second level of inference (training a few hyper-parameters or comparing a few algorithms), more examples are reserved for training than for validation. Typical choices of practitioners are $t/m \in [70\%, 90\%]$.

It is a good idea to vary t/m and plot learning curves and error bars on those curves. If one model has a curve dominated by all other models, select this model. Otherwise, if the curves cross, use your judgement to decide whether the error bars allow you to extrapolate the learning curves to $t/m = 1$. If so, you may choose the model whose extrapolated performance at $t/m = 1$ is best. Otherwise, attempt to reduce the variance by collecting more data or using cross-validation, as explained in the next section.

4.2.3 CROSS-VALIDATION

Often, there are not enough training data to guarantee good error bars without significantly degrading classifier performance. One may then resort to performing multiple data splits and averaging the results. Let us call D_v a subset of the training data of size v and $r_{val}[f, D_v]$ the corresponding validation estimator, when training on the remaining $t = (m - v)$ examples and testing on D_v. For K such subsets D_v, the **cross-validation estimator** is:

$$R_{CV}[f, v] = \frac{1}{K} \sum_{D_v} r_{val}[f, D_v] . \tag{6}$$

There are many ways in which the data may be split. The most widely used method is **K-fold cross-validation**. It consists in partitioning training data into $K \simeq m/v$ disjoint

subsets D_v of roughly equal sizes v (up to rounding errors). In stratified cross-validation, the class proportions are respected in all subsets. The variance of the results may be reduced by performing Q times K-fold cross-validation and averaging the results of the Q runs.

Finding an optimum proportion between training and validation data is even more difficult for cross-validation than for simple validation, because there is no unbiased estimator of the error bars (Bengio and Grandvalet, 2004). We recommend the following heuristic: plot learning curves $R_{CV}[f, v]$ as a function of t/m ($t/m \in [10\%, 90\%]$). For K-fold cross-validation (or Q times K-fold cross-validation), you may use $K = 2, 3, 5, 10$ and swap the roles of the training and validation sets to get points for which $t < v$. Use as error bar the approximate standard error:

$$\sigma_{CV} = \frac{\text{std}(r_{val}[f, D_v])}{\sqrt{K}} \ . \tag{7}$$

While averaging the results of several K-fold reduces the variance, we do not know exactly by how much. We suggest to divide $\text{var}(r_{val}[f, D_v])$ by K and NOT by $\text{num}(D_v) = Q \times K$ because in K-fold cross-validation the results on the K folds are reasonably independent, while in Q times K-fold the independence assumption is severely violated. If one model has a curve dominated by all other models, select this model. Otherwise, if the curves cross, use your judgement to decide whether the error bars allow you to extrapolate the learning curves to $t/m = 1$. If so, you may choose the model whose extrapolated performance at $t/m = 1$ is best. Otherwise, collect more data to reduce the error bars or proceed to Section 4.4. We illustrate this method in Figure 5.

Before closing this section, we want to make a few remarks:

1. The larger the number of models being compared, the higher the risk of **over-fitting the cross-validation error** when selecting a model. A simple correction may be applied to the error bars by multiplying them by $\sqrt{M}$, M being the number of models being compared. For a justification, see Section 4.4.

2. The cross-validation method provides an estimate of the generalization error of a model trained on m examples by training K models on $t < m$ models. In the end, we might want to **train a final model on all m examples**, or vote on the results of all K model (ensemble averaging).

3. The **leave-one-out estimator** $R_{loo}[f]$ is a cross-validation estimator using $v = 1$ and averaging over all possible choices of D_1:

$$R_{loo}[f] = \frac{1}{m} \sum_{D_1} r_{val}[f, S_1] \ .$$

The leave-one-out estimator is almost unbiased, but is has a high variance. Its principle advantage is that for some models there exist exact or approximate algebraic formulas allowing us to compute it from the training error, without having to train m models (virtual leave-one-out). An example was given in Section 3.1 for the case of the ridge regression algorithm.

4. The **out-of-bag estimator** is used for bagging ensemble methods (Breiman, 1996). Recall that for bagging, several models are built from datasets of size m sampled with replacement from the training data. Each training set contains approximately $t = 2/3$ of the training examples. The remaining $v = (m - t)$ examples (called out-of-bag samples), may be used to form a subset D_v and compute $r_{val}[f, D_v]$. The out-of-bag estimator is obtained by averaging $r_{val}[f, D_v]$ over all trials.

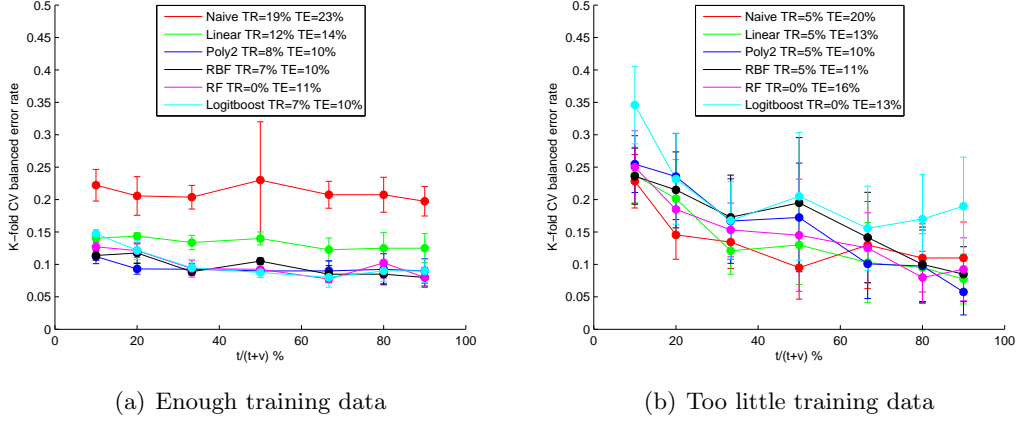(a) Enough training data   (b) Too little training data

Figure 5: **Cross-validation learning curves.** Learning curves are shown for two toy examples. We compare the six classifiers of Figure 2. Of m training examples, t are used for training and $v = (m - t)$ are used for validation. (a) **Example of Figure 2**: We performed K-fold cross-validation (see text). The four bottom learning curves are clearly dominated by the two top ones. Any of these classifiers should perform similarly well. Indeed, they all have almost the same test error rate $\simeq 10\%$. (b) **Example of Figure 1**: The number of examples is ten times smaller than in the previous example. We performed 10 times K-fold cross-validation (see text) in an effort to reduce variance. The learning curves cross and the number of training examples is too small to reliably extrapolate performances to $t = m$ and predict which classifier should generalize best on independent test data.

4.3 Search strategies

The problem of searching hyper-parameter space is often overlooked because most practitioners use exhaustive search for discrete parameters or grid search for continuous parameters (*i.e.,* discretize them). Grid search combined with 10-fold cross-validation has become almost a de-facto standard for model selection among practitioners. However, this strategy may be suboptimal:

- for computational reasons, because a coarse discretization might be required to cut down computations, leading to sub-optimal results,

- for statistical reasons, because an intensive search is prone to over-fitting (error bars increase with the number of models tried).

The most advanced methods for training $f(\mathbf{x}; \boldsymbol{\alpha}, \boldsymbol{\theta})$ perform a joint optimization of the two levels of inference (level 1: learning $\boldsymbol{\alpha}$; level 2: learning $\boldsymbol{\theta}$) called **bi-level optimization** (Bennett et al., 2006). Eventually two different objective functions are used for the two levels (*e.g.,* a penalty-based cost function for learning $\boldsymbol{\alpha}$ and a cross-validation estimator for the hyper-parameters $\boldsymbol{\theta}$). However, not all algorithms lend themselves to bi-level optimization programming. The next best solution is to treat the optimization of $\boldsymbol{\alpha}$ as an independent "black box" sub-routine returning a function $f(\mathbf{x}; \boldsymbol{\theta})$, which is called by a

18

higher level program optimizing $\boldsymbol{\theta}$. Linear programming (*e.g.,* the simplex method) may be used to optimize continuous hyper-parameters (Cawley, 2006).

Good results have also been recently obtained with regularized stochastic search methods such as Particle Swarm Optimization (PSO) (Escalante et al., 2009). The method, which is available in the CLOP package (Saffari and Guyon, 2006), allows full model selection (selection of preprocessing, feature selection method, learning machine, and post-processing).

4.4 Nested subset methods

Although a *nested subset method* is just particular case of search strategy, we devote a special section to nested subset methods because of their practical importance. We focus in this section on finite families of heterogenous models. Nested subset methods are also applied to hyper-parameter selection, under the name of "structural risk minimization" (see Appendix A.2).

Let us call $\mathcal{F} = \{f_1, f_2, ..., f_i, ..., f_M\}$ a family of models to choose from. For instance, if we consider the models used in Figure 2: $\mathcal{F}$ ={*Naive, Linear, Poly2, RBF, RF, Logitboost*}. Training and evaluation of the models may be performed using validation or cross-validation. We call $R_{val}[i]$ the evaluation function chosen (*e.g.,* the 10-fold cross-validation error) and $\sigma_{val}[i]$ the corresponding error bars (standard error). We will assume for simplicity that the error bars are similar for all models and we refer to the average error bar as σ_{val}.

Rather than just relying on $R_{val}[i]$ to select the best model, we would like to take into account the error bars and bias the decision towards simpler models, if there is no statistically significant performance advantage.

The idea is to form nested subsets of models S_i such that:

$$S_1 \subset S_2 \subset ... \subset S_i \subset ... \subset S_M = \mathcal{F} \ .$$

In our example, the following structure may be built:

$$
\begin{aligned}
S_1 &= \{Naive\}, \\
S_2 &= \{Naive, Linear\}, \\
S_3 &= \{Naive, Linear, Poly2\}, \\
S_4 &= \{Naive, Linear, Poly2, RBF\}, \\
S_5 &= \{Naive, Linear, Poly2, RBF, RF\}, \\
S_6 &= \{Naive, Linear, Poly2, RBF, RF, Logitboost\}.
\end{aligned}
$$

The nested subsets define model classes, which are guaranteed to increase in model complexity, regardless of the order chosen to add the models. The order in which the models are introduced impose a prior giving a preference to the models appearing sooner in the list.

We compute a second level of inference training error in subset S_i:

$$R_{val}[S_i] = \min_{j=1:i} R_{val}[j] \ .$$

To take into account error bars, we apply Tchebychev's inequality to $R[i]$, the expected risk (generalization error) of model i:

$$Proba\left[(R[i] - R_{val}[i]) \geq \frac{\sigma_{val}}{2\eta}\right] \leq \eta \ , \tag{8}$$

where $\eta \in [0, 1]$.

In order to define a *guaranteed risk* (a valid upper bound) for $R[S_i]$ (the expected risk of the model minimizing $R_{val}[i]$ in subsetS_i), we must consider the problem of large deviations Vapnik (1998) and bound the supremum of $(R[j] - R_{val}[j])$, $j = 1 \in [1, i]$:

$$Proba\left[(R[S_i] - R_{val}[S_i]) \geq \kappa\right] \leq Proba\left[\sup_{j=1:i}(R[j] - R_{val}[j]) \geq \kappa\right] \ .$$

Since:

$$Proba\left[\sup_{j=1:i}(R[j] - R_{val}[j]) \geq \kappa\right] \leq \sum_{j=1}^{i} Proba\left[(R[j] - R_{val}[j]) \geq \kappa\right] \ ,$$

taking into accounts the inequality 8, we obtain:

$$Proba\left[(R[S_i] - R_{val}[S_i]) \geq \frac{\sigma_{val}}{2\eta}\right] \leq i \times \eta \ ,$$

or, substituting $\delta = i \times \eta$:

$$Proba\left[(R[S_i] - R_{val}[S_i]) \geq \frac{\sigma_{val}}{2\delta} \times \sqrt{i}\right] \leq \delta \ .$$

We therefore define the following guaranteed risk:

$$\boxed{R_{gua}[S_i] = R_{val}[S_i] + \frac{\sigma_{val}}{2\delta} \times \sqrt{i}} \ , \tag{9}$$

which holds with confidence $(1 - \delta)$, $\delta \in [0, 1]$. We use this guaranteed risk as a penalized cost function to choose the best model instead of just using $R_{val}[i]$.

Examples are shown in Figure 6. We plot the balanced error rate (average of the error rate of the first class and that of the second class). The examples of Figure 6-a and Figure 6-b correspond to the data of Figure 2. They illustrate a case in which the error bars are small. The model selection process is affected by the choice of model order at confidence level $(1 - \delta) = 95\%$: If we consider all 720 possible model orders, $Poly2$, RBF, RF, and *Logitboost* are each selected about 25% of the time, showing that the performances of these four models are undistinguishable. The algorithm therefore finds good performing models for almost any ordering. *Linear* is only selected 12 times (2%) and *Naive* never. Figure 6-b shows an order in which *Linear* is selected, illustrating a bad bias introduced by this particular model order.

The examples of Figure 6-c and Figure 6-d correspond to the data of Figure 1. They illustrate a case in which the error bars are large. At confidence level $(1 - \delta) = 95\%$, if we consider all 720 possible model orders, $Naive$, $Linear$, $Poly2$, RBF, and RF are all chosen about 20% of the time and *Logitboost* never. The algorithm considers that, at that confidence level, the performances of these five models are undistinguishable. If we are willing to lower our confidence to $(1 - \delta) = 50\%$, Figure 6-b shows that the algorithm chooses then $Poly2$, which is the model having best generalization error. However, it cannot be excluded that this is just luck, given the high level of uncertainty.
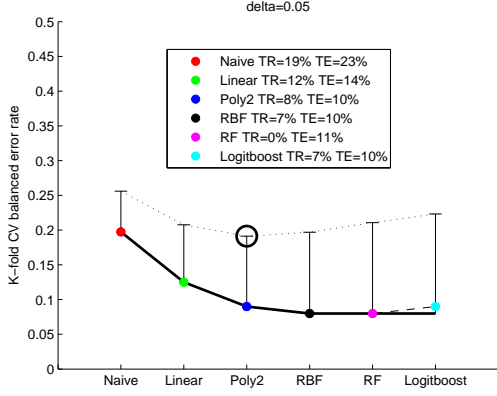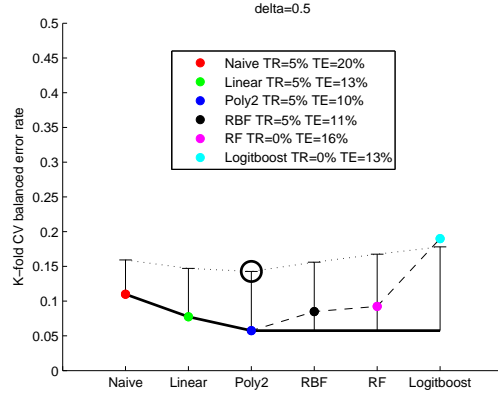
(a) Fig. 2 data. Proposed model order.
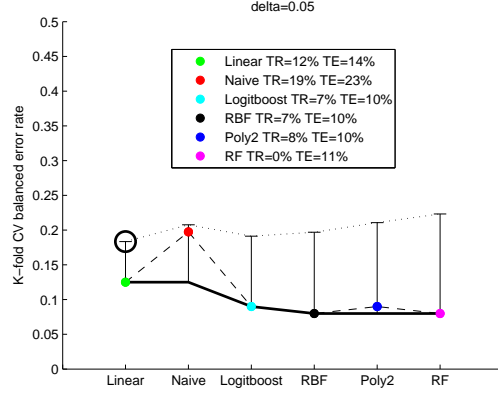
(b) Fig. 2 data. Random model order.
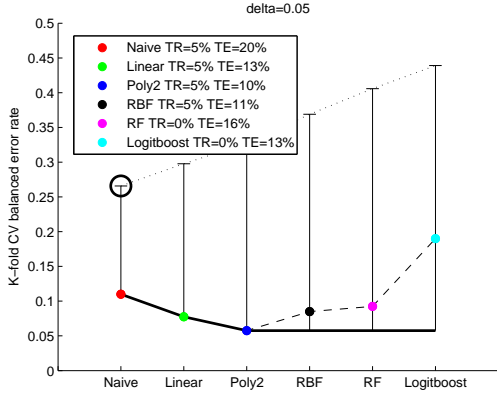
(c) Fig. 1 data. Proposed model order.

(d) Fig. 1 data. Same order, larger δ.

Figure 6: **Nested subset method**. The models ordered on the x-axis from left to right are indexed by variable i, $i = 1 : M$, $M = 6$. The dashed line joins the cross-validation (CV) results: $R_{CV}[i]$. The solid line represents the best CV result in nested subsets of models: $\min_{j=1:i} R_{CV}[i]$. The dotted line represents the best CV results plus an error bar $(1/\sqrt{2\delta})\sigma_{CV}\sqrt{i}$ (a guaranteed risk with confidence $(1 - \delta)$). The selected solution is circled. The training error of a model trained on all m examples and the corresponding test error (computed on a large independent test set) are indicated in the legend.

4.5 Heterogenous ensembles

We already mentioned ensemble methods in several parts of the chapter as a means of reducing variance (bagging) or reducing both bias and variance (boosting). Ensemble methods can also be used as a way of circumventing model selection. In particular, a method of heterogeneous ensembles proposed by (Caruana and Niculescu-Mizil, 2004; Caruana et al., 2006) has been very successful in benchmarks. The method called "Ensemble Selection" is an overproduce-and-select ensemble building method that is designed to generate high performing ensembles from large, heterogeneous libraries of models.

1. Generate a large library of base classifiers (thousands of models), using *e.g.,* the algorithms of Section 3, for many combinations of hyper-parameter values.

2. Form cross-validation splits of the training data.

3. Train and validate each model on every data split. The performance of the model is estimated by the average validation error (called cross-validation error). Keep every prediction made to be able to construct ensemble predictions.

4. Proceed to build ensembles with greedy forward selection. An ensemble model predicts according to the average prediction of its members (or a majority vote for classification algorithms). At each step, add the model improving the cross-validation error most. Models can be added multiple times.

For computational reasons, instead of re-training models on all the data to build the final ensemble, the members trained on the various folds may be included directly. Performing forward selection in a large library of models may overfit the cross-validation estimator. (Caruana et al., 2006) proposed several means of alleviating over-fitting, including initializing the ensemble with the models performing best individually and bagging the ensembles of classifiers (training several ensembles using data subsets and averaging the resulting ensembles).

Many other methods to build heterogeneous ensembles have been proposed, including the simple weighted majority algorithm, which uses the CV performance to weight the predictors. Another approach is the Monte Carlo Markov Chain (MCMC) method, which performs a stochastic search in model space, visiting longer the most promising parts of the space. For a given candidate step in model space, a new model is accepted if it performs better than the current model or, if it performs worse, it is accepted with a certain probability, exponentially decreasing with its estimated generalization error (*e.g.,* the CV error). Every model visited is added to the ensemble.

5. Feature selection

An important part of machine learning is to use a good data representation, but choosing an appropriate data representation is very domain dependent. Common preprocessing steps include filter bank transformations, Fourier transform, taking the log or square root, making sums or products of features, normalizing the features (columns of the matrix) or normalizing the samples (lines of the matrix). In benchmarks, it has often been found that generating a large number of low-level features yields better result than hand-crafting a

1. Choose an algorithm $\mathcal{A}$ to create nested feature subsets.

2. Choose a learning machine $\mathcal{M}$ to evaluate the feature subsets.

3. Split the m available training samples into K training and validation subsets pairs $\{D_t^j, D_v^j\}$ of dimension t and v, $t + v = m$, $j = 1 : K$.

4. For $j = 1 : K$

   a. Using $\mathcal{A}$ and only the D_t^j examples, create nested subsets of the n available features: $S_1^j \subset S_2^j \subset ... \subset S_i^j \subset ...S_n^j$.

   b. For $i = 1 : n$, train $\mathcal{M}$ on subset S_i^j using D_t^j, and test it using D_v^j. Call $r_{val}[i, j]$ the resulting estimation of performance.

5. Compute the CV scores of the nested feature subsets: $R_{CV}[i] = \sum_{j=1}^{K} r_{val}[i, j]$.

6. Select the best number of features: $N = \text{argmin}_i(R_{CV}[i])$.

7. Using all m training examples, create nested subsets of the n available features: $S_1 \subset S_2 \subset ... \subset S_i \subset ...S_n$.

8. Select S_N.

Figure 7: **Cross-validation for nested subsets of features**.

few features incorporating a lot of expert knowledge. The feature set can then be pruned by feature selection. This section briefly outlines basic feature selection strategies. More details are found in (Guyon and Elisseeff, 2003; Guyon et al., 2006a, 2007a; Guyon, 2008).

Since the paper of (Kohavi, 1995) the nomenclature of "filter" and "wrapper" is widely used in feature selection. The former refers to methods using ad-hoc relevance criteria to select feature subsets. The latter refers to a search in feature subset space guided by the performance of a predictor or *learning machine*. A further distinction is made between "pure wrapper methods" in which the predictor is used as a black box, with no required knowledge of the learning algorithm, and "embedded methods" in which the search for an optimal feature subset is guided by the learning algorithm.

5.1 Nested subset methods

We already emphasized the importance of nested subset methods for model selection. They are so prominent for feature selection that we start with them. Many methods exist to rank features with univariate criteria or create nested subsets of features by forward selection or backward elimination. Once a ranking is obtained, the problem is to set a cutoff threshold on the number of features.

In Section 4.4, we proposed a penalized cross-validation score as evaluation function. For feature selection, the number of subsets being considered being usually very large, a different method must used not to over-penalize large subsets. The optimum number of features N is assessed using a cross-validation method, which includes a separate feature ranking in each fold. Then, a final ranking is performed using the entire training set and the first N features are selected. This method is less biased than using the ranking produced with the entire training set and selecting the best subset directly by cross-validation. The algorithm is outlined in Figure 7.

Given a large number of features (several thousands) it is often useful, for computational reason and to alleviate storage requirements, to use first a "filter" to reduce the number of features. Most practitioners use univariate feature ranking filters, which evaluate the individual predictive power of the features. Examples of relevance ranking scores include the Pearson correlation coefficient between feature and target, the mutual information, and the Area under the ROC curve (AUC), considering a feature as a univariate classifier. The choice of relevance score depends on the nature of the features and the target (binary, categorical, continuous) and the presence or not of non-linearities. A large choice of filters and their domains of application is given in (Guyon et al., 2006a, Chapter 3).

The features are then ranked according to the relevance score and one of two methods are commonly used to set an ad-hoc cutoff threshold:

1. **Eye-balling**: Plotting the relevance score as a function of rank and deciding the number of features worth keeping.

2. **False discovery rate (FDR)**: Statistically estimating the fraction of features falsely thought of being relevant to the target (see *e.g.,* Guyon et al., 2006a, Chapter 2).

The second method is more principled and it has the advantage of producing a normalized score. However, the threshold on the FDR being arbitrary, there is no clear advantage over the first method. Furthermore, the FDR does not correlate well with prediction accuracy because prediction accuracy is more affected by false rejection of features than false acceptance (Guyon et al., 2008).

For these reasons, the cutoff threshold is often set using the cross-validation performance of learning machines built with nested subsets of features of increasing size, starting from the highest ranking ones. For that purpose, use the algorithm of Figure 7. This simple "wrapper" method performs surprisingly well in practice. It has two disadvantages: (1) the candidate feature subsets include often redundant features; (2) complementarity of features is not taken into account in forming the candidate feature subsets. But these are not severe pitfalls because redundancy is not harmful to prediction performance and seeking complementary features is prone to over-fitting.

If feature redundancy is a concern because the application requires obtaining a compact feature subset, then two options may be considered: Nested feature subsets may be obtained by starting from an empty set and progressively adding features (**forward selection**), or starting with the subset of all features, progressively eliminating features (**backward elimination**). Strategies alternating forward and backward steps also exit, but we limit ourselves to these two simpler methods, both of which generate nested feature subsets.

Such methods come in two flavors:

1. **Pure wrapper method** (requiring no knowledge of the learning algorithm $\mathcal{M}$): At each step i of the forward selection (or backward elimination) process, try all candidate features (to be added or removed) and evaluate all corresponding candidate feature subsets using $\mathcal{M}$ (*e.g.,* with a cross-validation estimator R_{CV}). The candidate subset S_i with best performance (call it *e.g.,* $R_{CV}[i]$) is retained and the procedure is iterated.

When all features are exhausted, $N = \mathrm{argmin}_i(R_{CV}[i])$ is chosen and the subset S_N selected.

2. **Embedded method** (using the learning machine $\mathcal{M}$ to guide the search): Use a learning machine $\mathcal{M}$ capable of producing a feature ranking (*i.e., $\mathcal{A} = \mathcal{M}$*) and apply the algorithm of Figure 7.

The first method has the advantage that is can be used with any machine learning algorithm, treated as a black box. But, for n features, it requires evaluating $n(n-1)/2$ models by cross-validation. In contrast, the second method evaluates only n models. Examples of the second method include a backward elimination algorithm for Support Vector machines called RFE-SVM (Guyon et al., 2002) and a forward selection algorithm based on Gram-Schmidt orthogonalization (Stoppiglia et al., 2003).

The computation of **scaling factors** provides an alternative to forward and backward selection methods. Some models possess feature scaling factors, which are adjusted as part of learning (see Guyon et al., 2006a, Chapter 5). Those include Bayesian Automatic Relevance Determination (ARD) algorithms (MacKay, 1992; Rasmussen and Williams, 2006) and kernel method adjusting scaling factors by gradient descent. For linear models, the absolute value of the weights of the model is sometimes simply used (Chang and Lin, 2008). The scaling factors may serve as relevance ranking scores to create nested feature subsets.

5.2 Other wrapper and embedded methods

Any search strategy in feature subset space, including forward-backward methods and stochastic search (see Guyon et al., 2006a, Chapter 4, for a review), may be used in conjunction with any evaluation method (including penalized cost functions and cross-validation), to form a wide variety of **wrapper methods** of feature subset selection (Kohavi and John, 1997). The more extensive the search, the higher the risk of over-fitting. Extensively searching feature space is not recommended unless there is a large amount of available training data (compared to the number of features to select from, $n \gg m$).

A number of completely integrated **embedded methods**, producing simultaneously a feature subset and a trained learning machine, have been proposed. Those include L_0 and L_1 regularization methods: L_0 SVM (Weston et al., 2003), L_1 SVM (Zhu et al., 2003) and Lasso (Tibshirani, 1994). For these methods, selecting the best number of features is replaced by selecting a regularization parameter, which sets indirectly the number of features. This regularization parameter is usually selected by cross-validation. Decision trees and Random Forests can also be considered embedded feature selection methods.

6. Conclusion

State-of-the-art learning machines include powerful embedded model-selection methods called "regularization". Hence, even universal approximators capable of learning any function will not necessarily overfit. Nonetheless, there almost always remain a few hyper-parameters, which practitioners adjust by cross-validation. This chapter gave practical guidelines on how to properly use cross-validation to select models, hyper-parameters, and feature subsets, without risking to overfit the cross-validation error. The most important guideline remains to try simple thing first and not to try too many things.

Appendix A. Learning theory convergence: the example of ridge regression.

In this appendix, we give a quick tour of learning theoretic implementations of Ockham's razor. We use throughout this brief tutorial the same example, known under different names in different fields: **weight decay**, **L2-norm regularization**, **Gaussian Processes**, or **ridge regression**. This example illustrates "theory convergence". But this convergence should not lead the reader to think that all theories are equivalent. In fact, because they rely on quite different premises and branch into various directions, they have given birth to rather different approaches to model selection. In what follows, we adopt the same notations as in the rest of the chapter: Training data $D = \{(\mathbf{x}^k, y^k), k = 1, ...t\}$ are drawn randomly and independently from an unknown distribution $P(\mathbf{x}, y)$. We call f a model belonging to a model class $\mathcal{F}$. The goal is to minimize an expected risk (also called generalization error) $R[f] = \int \mathcal{L}(f(\mathbf{x}), y) \, dP(\mathbf{x}, y)$, where $\mathcal{L}(f(\mathbf{x}), y)$ is a loss function measuring the discrepancy between the prediction $f(\mathbf{x})$ and the target value y.

A.1 Weight decay and the power of amnesia

The human brain is made out of billions of cells or Neurons, which are highly interconnected by synapses. Exposure to enriched environments with extra sensory and social stimulation enhances the connectivity of the synapses, but children and adolescents can lose them up to 20 million per day. Nature seems aware of Ockham's razor: eliminate useless synapses!

Inspired by such biological observations, researchers proposed to replace the regular update rules for the neural connections in artificial neural networks $\mathbf{w} \leftarrow \mathbf{w} + \eta U(D, \mathbf{w})$ by $\mathbf{w} \leftarrow (1 - \lambda)\mathbf{w} + \eta \, U(D, \mathbf{w})$, where λ is a decay parameter and $U(D, \mathbf{w})$ is the regular update rule, *e.g.,* given by the backpropagation algorithm. Performance improvements are obtained with this heuristic, but, what kind of theoretical justification can be given?

A.2 Structural Risk Minimization (SRM)

In statistical learning theory, the expected risk $R[f]$ or generalization error is bounded by a guaranteed risk $R_{gua}[f, \delta]$ with probability $(1 - \delta)$:

$$R[f] \leq R_{gua}[f, \delta]$$

A tentative choice for a model selection ranking index could be $r[f] = R_{gua}[f]$. We cannot be certain that for all pairs of functions f_1 and f_2 $R_{gua}[f_1] < R_{gua}[f_2] \Rightarrow R[f_1] < R[f_2]$. But this choice implements Ockham's razor bias towards "simpler" models (here models with smaller capacity). Unfortunately, for many model classes $\mathcal{F}$ the guaranteed risk depends on an elusive notion of capacity $C(\mathcal{F})$, impossible to compute (see Section 4). The idea of **Structural Risk Minimization** is to organize the model class in nested subsets of functions: $S_1 \subset S_2 \subset ... \subset \mathcal{F}$. This guarantees that the capacity of the model (sub-)class increases from subset to subset: $C(S_1) < C(S_2) < ... < C(\mathcal{F})$. This allows us to convert the learning problem into a constrained optimization problem monitoring the capacity:

$$\min_f R_{tr}[f] \,, \quad \text{such that } f \in S_i$$

and introduce a second level of inference: find the optimum value of i.

What have we won by introducing a second level of inference? The model class $\mathcal{F}$ might be a family of universal approximators with infinite VC dimension $C(\mathcal{F})$. Hence any finite training set drawn from any distribution could be learned without error ($R_{tr}[f] = 0$), the price to be paid being the huge (infinite) error bar $\epsilon(C(\mathcal{F}), \delta)$, hence, potentially a very bad generalization error (unbounded). In contrast, if we restrict the model class to S_i, we may get worse $R_{tr}[f]$, but a better guaranteed risk, hence possibly a better generalization. We now need to optimize i, for instance by cross-validation.

SRM AND WEIGHT DECAY

We now connect the method of Strutural Risk Minimization to "weight decay". Assume that our models are parameterized by a weight vector $\mathbf{w}$. We consider the structure: $S_i = \{\mathbf{w} | \|\mathbf{w}\|^2 \leq A\}$, where A is a positive constant. We are then brought back to solving the optimization problem:

$$\min_f R_{tr}[\mathbf{w}] \ , \quad \text{such that } \|\mathbf{w}\|^2 \leq A$$

which can be replaced, via the use of a positive Lagrange multiplier γ, by

$$\boxed{\min_{\mathbf{w}} R_{tr}[\mathbf{w}] + \frac{\gamma}{t}\|\mathbf{w}\|^2, \ \gamma > 0 \ .} \tag{10}$$

We call $\Omega[f] = \|\mathbf{w}\|^2$ "regularizer"[11] and define a regularized risk functional $R_{reg}[\mathbf{w}] = R_{tr}[\mathbf{w}] + (\gamma/t)\|\mathbf{w}\|^2$. Now we can easily recover the weight decay learning rule, by assuming that learning is performed with gradient updates:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial R_{reg}[\mathbf{w}]}{\partial \mathbf{w}}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial R_{tr}[\mathbf{w}]}{\partial \mathbf{w}} - 2\eta \frac{\gamma}{t} \mathbf{w}$$

$$\mathbf{w} \leftarrow (1 - \lambda)\mathbf{w} + \eta \ U(D, \mathbf{w})$$

where $U(D, \mathbf{w}) = -\frac{\partial R_{tr}[\mathbf{w}]}{\partial \mathbf{w}}$ is the regular weight update based on the minimization of the empirical risk and $\lambda = 2\eta\gamma/t$ is the decay parameter (see Section A.1).

Many other structures can be defined to monitor over-fitting, for example, is $\mathcal{F}$ is the class of polynomial models, S_i can be all the class of polynomials of degree less than i. Structures can also be defined for feature selection where S_i is the subset of models with less than i features (see Section 5.1) and for heterogeneous learning machines (see Section 4.4).

A.3 Bayesian priors

We now turn to a Bayesian interpretation of weight decay. In the Bayesian setting, it is assumed that a model f was drawn from the model class $\mathcal{F}$ according to a given probability distribution $P(f)$ called the **prior** and then the model was used to generate the data. For instance, in the case of a regression problem with a Gaussian noise model, it is assumed that input data are drawn according to an unknown distribution $P(\mathbf{x})$ and output data are drawn

---

11. See Section A.6 for a justification of this nomenclature.

according to $P(y|\mathbf{x}, f) \rightsquigarrow \mathcal{N}(f(\mathbf{x}), \sigma_1^2)$ where the variance σ_1^2 is assumed to be independent of $\mathbf{x}$. Hence, for a dataset $D = \{(\mathbf{x}^k, y^k), k = 1, ...t\}$, $P(D|f) = \Pi_k P(y^k|\mathbf{x}^k, f)P(\mathbf{x}^k)$ and therefore $P(D|f)$ is proportional of $\Pi_k \exp -(y^k - f(\mathbf{x}^k))^2/(2\sigma_1^2)$.

From the Bayesian point of view, solving the regression problem requires calculating

$$P(y|\mathbf{x}, D) = \int P(y|\mathbf{x}, D, f)P(f|\mathbf{x}, D)df$$

which can be simplified to

$$P(y|\mathbf{x}, D) = \int P(y|\mathbf{x}, f)P(f|D)df$$

assuming that all the dependency between y and D is captured by f and that the choice of f is independent of the point $\mathbf{x}$ where it is evaluated. If we are more interested in expected values than in distributions, we can rather compute $E(y|\mathbf{x}, D) = \int E(y|\mathbf{x}, f)P(f|D)df$. Noting that $E(y|\mathbf{x}, f) = f(\mathbf{x})$, the Bayesian regression solution is to compute:

$$E(y|\mathbf{x}, D) = \int f(\mathbf{x})P(f|D)df \; , \tag{11}$$

where, $P(f|D)$ is estimated using the Bayes formula

$$P(f|D) = P(D|f)P(f)/P(D) \; . \tag{12}$$

The following nomenclature is used: $P(f|D)$ is called the **posterior**, $P(D|f)$ is called the **likelihood**, and $P(f)$ is called the **prior**. The calculation of the integral may be impossible or tedious. In the Maximum A Posteriori (MAP) framework it is further assumed that $P(f|D)$ is peaked around it mean and that by choosing $f_{MAP} = \mathrm{argmax}_f P(f|D)$ one can approximate $E(y|\mathbf{x}, D)$ by $f_{MAP}(\mathbf{x})$. Hence, noting that $P(D)$ is independent of f, the MAP problem is to solve $f_{MAP} = \mathrm{argmax}P(D|f)P(f)$, or, equivalently

$$f_{MAP} = \mathrm{argmin}_f[-\ln P(D|f) - \ln P(f)] \; . \tag{13}$$

BAYESIAN PRIORS AND SRM

To go back to our weight decay example, assume the we have a model f parameterized by a weight vector $\mathbf{w}$ and such that every weight w_i is drawn independently from a Gaussian distribution $\mathcal{N}(0, \sigma_2^2)$[12]. In other words, we have a **Gaussian prior** and the data generative process takes the name of **Gaussian process**. From the Gaussian prior, $P(f)$ is proportional to $\Pi_i \exp(-w_i^2/(2\sigma_2^2)) = \exp(-\|\mathbf{w}\|^2/(2\sigma_2^2))$ and from the Gaussian noise, $P(D|f)$ is proportional of $\Pi_k \exp(-(y^k - f(\mathbf{x}^k))^2/(2\sigma_1^2))$. Hence, using Equation 13, the MAP solution becomes:

$$f_{MAP} = \mathrm{argmin}_f \sum_{k=1}^{t}(y^k - f(\mathbf{x}^k))^2/(2\sigma_1^2) + \|\mathbf{w}\|^2/(2\sigma_2^2) \; . \tag{14}$$

---

12. The weights can each have a different variance, leading to the so-called ARD prior and the hypothesis of independence between the weights can be relaxed, but we start with the simplest possible example.

We easily see that this problem is equivalent problem 10, in the particular case where $R_{tr}[\mathbf{w}] = (1/t)\sum_{k=1}^{t}(y^k - f(\mathbf{x}^k))^2$, *i.e.,* when we use the square loss, and by using $\gamma = \sigma_1^2/\sigma_2^2$. It is the ridge regression problem of minimizing R_{reg} in Equation 3.

More generally the choice of a noise model corresponds to the choice of a loss function and the choice of a prior corresponds to the choice of a structure or regularizer. Note that Bayesian priors do not necessarily enforce simplicity and therefore Ockham's razor is not built into the Bayesian/MAP framework.

A.4 Minimum Description Length

Weight decay can also be given an information theoretic justification, stemming from the Minimum Description Length or MDL principle (Grünwald, 2005). Imagine that we want to transmit the data D over a channel as fast as possible and for that purpose we want to encode it in a compact way. We proceed with a two-part code: we model the data and transmit the coded model and the residuals, *i.e.,* the errors made by predicting the data with the model. In the case of regression, we assume that the inputs $\mathbf{x}$ are given to both ends of the transmission channel and we just need to transmit y. As per Shannon's theorem, the length of the shortest code to encode a piece of information X is $-\log_2 P(X)$ bits or $-\ln P(X)$ Nepers. So in Neper, since length of the shortest code to transmit the model is $-\ln P(f)$ and the residuals are given by $-\ln P(D|f)$, we are brought back to the problem of Equation 13.

While the MDL justification yields the same optimization problem as MAP, it sheds a different light on over-fitting avoidance. First, it does not assume that the data were drawn using a double random process (first draw f according to a given distribution then draw the data D). Second it introduces a notion of complexity of the model class linked to the length of the shortest coded needed to encode the model. In our weight decay example, if we assume that the w_i coefficients are encoded with finite precision decimal values, models with smaller $\|\mathbf{w}\|^2$ can be coded with a shorter code (perhaps at the expense of larger residuals). This is a further justification of weight decay, implementing Ockham's razor principle.

A.5 Weight decay, ridge regression, and the bias/variance tradeoff

Weight decay is a means of monitoring the bias/variance tradeoff, which we introduced in Section 4. The bias and variance of a linear model $f(\mathbf{x}) = \mathbf{x}\mathbf{w}^T$ trained with the square loss $\mathcal{L}(f(\mathbf{x}^k), y) = (f(\mathbf{x}^k) - y)^2$ and the square norm regularizer $\|\mathbf{w}\|^2$ have been worked out (Fox, 1997). We introduce a matrix notation: $X = [\mathbf{x}^1, \mathbf{x}^2, ...\mathbf{x}^k]$ is the (t, n) matrix whose lines are the training vectors of n features and $\mathbf{y} = [y^1; y^2; ...y^k]$ is the column vector of the t target values. In this particular case, optimization problem 10 can be written as:

$$\min_{\mathbf{w}} \|X\mathbf{w}^T - \mathbf{y}\|^2 + \gamma\|\mathbf{w}\|^2 \ .$$

It can be shown that the solution to this problem is given by:

$$\mathbf{w}_\gamma^T = X^T(XX^T + \gamma I)^{-1}\mathbf{y} = (X^TX + \gamma I)^{-1}X^T\mathbf{y} \ .$$

This is the so-called "ridge regression" estimator (see Section 3.1). When the t samples are collinear or nearly collinear, the matrix XX^T of dimension (t, t) has some off-diagonal

elements, which are very large, making it ill conditioned (difficult to invert). A similar bad conditioning happens to $X^T X$ of dimension (n, n) if some features are collinear or nearly collinear. Adding a positive constant to the diagonal improves the conditioning (turns a singular or nearly singular matrix into a regular matrix). In the limit of small γ, we recover the least-square estimator:

$$\lim_{\gamma \to 0} \mathbf{w}_\gamma = \mathbf{w}_{LSQ}$$

Assuming that the data were generated with $f(\mathbf{x}) = \mathbf{x}\mathbf{w}_{true}^T$ plus some Gaussian uncorrelated noise of zero mean, $\mathbf{w}_{LSQ}$ is an unbiased estimator of $\mathbf{w}_{true}$: $E_D[\mathbf{w}_{LSQ}] = \mathbf{w}_{true}$.

The ridge regression estimator can be re-written as:

$$\mathbf{w}_\gamma = U\mathbf{w}_{LSQ}, \text{with } U = (I + \gamma(X^T X)^{-1})^{-1} .$$

It can be verified that $\|\mathbf{w}_\gamma\|^2 \leq \|\mathbf{w}_{LSQ}\|^2$. Assume fixed X data (all randomization comes from $\mathbf{y}$), then $X^T X$ and U are fixed. In this case, the operator $E_D[.]$ averages only over the randomness of $\mathbf{y}$:

$$E_D[\mathbf{w}_\gamma] = U E_D[\mathbf{w}_{LSQ}] = U\mathbf{w}_{true} .$$

Hence

$$bias(\mathbf{w}_\gamma) = E_D[\mathbf{w}_\gamma] - \mathbf{w}_{true} = (U - I)\mathbf{w}_{true} .$$

The variance of the ridge estimator can also be derived (Fox, 1997):

$$variance(\mathbf{w}_\gamma) = (\frac{\sigma^2}{t - 1} X X^T + \gamma I)^{-1} X^T X (X X^T + \gamma I)^{-1}$$

Because the departure of U from I increases with γ, $bias(\mathbf{w}_\gamma)$ increases with γ.

As γ increases, the inverted term $(X X^T + \gamma I)^{-1}$ is increasingly dominated by γI. Therefore, $variance(\mathbf{w}_\gamma)$ is a decreasing function of γ.

The mean square error (MSE) of the ridge estimator is the sum of its squared bias and its variance. There is an optimum value of γ (dependent on the unknown $\mathbf{w}_{true}$) such that $\mathbf{w}_\gamma$ has a better MSE than $\mathbf{w}_{LSQ}$.

A.6 Regularization

We now turn to yet another viewpoint, that of **ill-posed problems** and **regularization**. If the hypothesis space $\mathcal{F}$ is "too big", the solution to the learning problem is not unique. The problem is then called "ill-posed". For example, finding a linear model when the number of training examples is smaller than the number of dimensions is an ill-posed problem. We will examine this example more closely and relate regularization to "weight decay".

We formulate the learning problem as that of solving a system of linear equations, which, using our previous matrix notations, can be written as:

$$X\mathbf{w}^T = \mathbf{y} . \tag{15}$$

Matrix X being usually rectangular, it is not invertible, but, by multiplying the equation right and left by X^T, we obtain a square matrix $X^T X$. The problem is then to solve

$$X^T X \mathbf{w}^T = X^T \mathbf{y} .$$

This matrix equation corresponds to a system of linear equations known as the **normal equations**. It is also obtained as the condition of optimality $\frac{dR_{tr}}{d\mathbf{w}} = \mathbf{0}$ for minimizing the empirical risk in the case of the square loss

$$R_{tr}[\mathbf{w}] = \frac{1}{t}\|X\mathbf{w}^T - \mathbf{y}\|^2 \ . \tag{16}$$

We now need to invert the square matrix $X^T X$, which unfortunately may be singular, for example if the problem is under-determined (fewer equations than unknown, $t < N$). One way to overcome the problem is to make the matrix $X^T X$ regular by adding to it a suitable regular matrix $\Gamma^T \Gamma$, often chosen as γI, $\gamma > 0$. In the limit $\gamma \to 0$ the matrix:

$$X^+ = \lim_{\gamma \to 0}(X^T X + \gamma I)^{-1} X^T$$

is known as the pseudo-inverse and the weight vector

$$\mathbf{w}^T = X^+ \mathbf{y}$$

is a solution to Equation 15 and to the minimization of the mean square error 16. It is the solution of minimum norm $\|\mathbf{w}\|$. One interesting property of the pseudo-inverse is that:

$$X^+ = \lim_{\gamma \to 0}(X^T X + \gamma I)^{-1} X^T = \lim_{\gamma \to 0} X^T (X X^T + \gamma I)^{-1}$$

which is sometimes convenient because inverting the matrix $X X^T + \gamma I$ of dimension (t, t) may be a lot faster than inverting $X^T(X X^T + \gamma I)$ of dimension (n, n) if $t << n$.

For non-vanishing values of γ, we obtain the "ridge regression" solution again

$$\mathbf{w}_\gamma^T = (X X^T + \gamma I)^{-1} X^T \mathbf{y} = X^T (X X^T + \gamma I)^{-1} X^T \mathbf{y} \ . \tag{17}$$

that is the solution of minimizing the regularized risk functional:

$$R_{reg} = \|X\mathbf{w}^T - \mathbf{y}\|^2 + \gamma\|\mathbf{w}\|^2 \ .$$

Tikhonov regularization (Tikhonov, 1943; Vapnik, 1998) is more general than a solution to solving ill conditioned systems of linear equations. It applies to all linear ill-posed problems, which can be represented by linear operator equations in some Hilbert space. In machine learning, regularized kernel methods are rooted in this theory, which we briefly summarize (see Schoelkopf and Smola, 2002, for further reading).

We first assume that the model class $\mathcal{F}$ considered is a Hilbert space of functions endowed with a dot product

$$< f, g >= \int f(\mathbf{x})g(\mathbf{x}) \ dP(\mathbf{x})$$

with the corresponding norm $\|f\|_{\mathcal{F}}^2 =< f, f >$. Consider an empirical risk $R_{tr}[f; D]$ of some element $f \in \mathcal{F}$ and some dataset $D = \{(\mathbf{x}^k, y^k), k = 1, ...t\}$. We define a regularized risk functional:

$$R_{reg}[f; D] = R_{tr}[f; D] + \Omega[f]$$

where Ω is usually a monotonically increasing function of $\|f\|_{\mathcal{F}}$, typically the multiplication by a positive constant.

The regularization method consists in replacing the minimization of $R_{tr}[f; D]$ by that of $R_{reg}[f; D]$. If both $R_{tr}[f; D]$ and $\Omega(\|f\|_{\mathcal{F}})$ are convex, there is a unique minimum of $R_{reg}[f; D]$ with respect to f. More generally, a regularization functional can be defined via a linear regularization operator Γ performing a transformation of f, such as extracting derivatives:

$$\Omega[f] = <\Gamma f, \Gamma f> = \|\Gamma\|_{\mathcal{F}}^2$$

REGULARIZATION AND KERNEL METHODS

There is a natural connection between regularization and kernel methods. It can be shown that to every regularization operator Γ defined over some function space $\mathcal{F}$ corresponds a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$ in which the solutions of minimizing $R_{reg}[f; D]$ are found.

By definition, a RKHS is a Hilbert space generated by a set of basis functions $K(x, \cdot)$ (*i.e.*, $f \in \mathcal{H}$ *iff* $f(\mathbf{x}) = \sum_k \alpha_k K(\mathbf{x}, \mathbf{x}^k)$), possessing the reproducing property $< f, K(\mathbf{x}, .) > = f(\mathbf{x})$. Following the "representer theorem" (Wahba and Nashed, 1974), a solution to the problem of minimizing the regularized risk $f^* = \text{argmin}_f R_{reg}[f; D]$, where $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ...(\mathbf{x}_n, y_n)\}$ is of the form $f^*(\mathbf{x}) = \sum_k \alpha_k K(\mathbf{x}, \mathbf{x}^k)$, where the sum runs over the training examples in D. Hence, the solution lies in the span of particular basis functions (kernels) centered on the training points. The smoothness properties of the kernel (and therefore of f) are directly related to the regularization operator Γ (typically a differential operator). Any Green's function $G(\mathbf{x}, .)$ of the operator $\Gamma^*\Gamma$ may be used as a kernel (Γ^* designates the adjoint operator, *i.e.*, satisfying $< \Gamma f, g > = < f, \Gamma^* g >$). A Green's function satisfies:

$$< \Gamma^*\Gamma G(x, .), f >_{\mathcal{F}} = < \Gamma G(x, .), \Gamma^* f >_{\mathcal{F}} = f(\mathbf{x}) .$$

In particular, Green's functions may be found by eigen decomposition of $\Gamma^*\Gamma$. Conversely, if optimization is carried out in a RKHS, one can derive a corresponding regularization operator.

For instance, going back to our earlier example, linear regression corresponds to optimizing a risk functional in the RKHS generated by the linear kernel $K(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'$. Indeed, the ridge regression solutions in the space $\mathcal{F}$ of functions $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$ are such that $f^* = \mathbf{w}^* \cdot \mathbf{x} = \sum_k \alpha_k K(\mathbf{x}, \mathbf{x}^k)$, with $\mathbf{w}^* = \sum_k \alpha_k \mathbf{x}^k$. This is easily seen from Equation 17: $\mathbf{x}\mathbf{w}^T = \mathbf{x}X^T(XX^T + \gamma I)^{-1}X^T\mathbf{y} = \mathbf{x}X^T\boldsymbol{\alpha}$, where $\boldsymbol{\alpha}$ is vector of t α_k coefficients and $\mathbf{x}X^T$ is a vector of t kernel values $K(\mathbf{x}, \mathbf{x}^k) = \mathbf{x} \cdot \mathbf{x}^k$. It can be verified that the regularizer $\Omega[f] = \|f\|_{\mathcal{F}}$ is indeed $\|\mathbf{w}^2\|$, the regularizer corresponding to "weight decay".

In Support Vector Machines (SVM) (**?**), and other kernel methods the kernels are usually semi definite positive and admit a dot product representation $K(x, x') = \sum_i \phi_i(\mathbf{x})\phi_i(\mathbf{x})$, where the index $i = 1, ...N$ runs over the N dimension of a feature space. N may be infinite for some kernels. For instance, the polynomial kernel $K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x} \cdot \mathbf{x}')^q$ (of degree q) admit an finite monomial expansion while the Gaussian kernel $K(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/\sigma^2)$ admits an infinite expansion. The functions of the model class have dual representations $f(\mathbf{x}) = \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}) = \sum_k \alpha_k K(\mathbf{x}, \mathbf{x}^k)$, with $\mathbf{w} = \sum_k \boldsymbol{\phi}(\mathbf{x}^k)$. In SVMs and kernel ridge regression the regularizer is $\|f\|_{\mathcal{F}}^2 = \|\mathbf{w}\|^2 = \boldsymbol{\alpha}^T K\boldsymbol{\alpha}$. A wide variety of kernel methods are obtained by combining various regularizers and loss functions (see Guyon et al.,

2006a, Chapter 1, for a review). For example, the loss function for SVMs is the hinge loss $\mathcal{L}(y, f(\mathbf{x})) = |1 - yf(\mathbf{x})|_+$, where $|a|_+ = 0$ if $a \leq 0$ and $|a|_+ = a$ otherwise.

CLOSING THE LOOP: GUARANTEED RISK MINIMIZATION

Support vector machines for classification maximize the margin in "feature space" between the training examples of either class around the decision boundary. If we impose that for marginal examples the $f(\mathbf{x}) = \pm 1$, the margin at the solution is $M^* = 1/\|\mathbf{w}^*\|$. It can be shown (Vapnik, 1998) that, in the linearly separable case,

$$R[\mathbf{w}*] \leq O(\rho^2 \frac{\|\mathbf{w}^*\|^2}{m})$$

and in the non-linearly separable case

$$R[\mathbf{w}*] \leq R_{tr}[\mathbf{w}*] + O(\rho \frac{\|\mathbf{w}^*\|}{\sqrt{m}})$$

where ρ is the radius of the smallest sphere enclosing the data. Hence, the choice of the regularizer $\|\mathbf{w}\|^2$ can be *a posteriori* justified. The success of SVMs has attracted renewed attention to the possibility of deriving algorithms from performance bounds.

A.7 Brief historical notes

The method of "ridge regression" has been re-invented many times and it is difficult to trace it back in history. The idea is essentially built into the least-square regression solution, via the normal equations published by Legendre in 1805. Assume we want to build a linear model $y = \mathbf{x}\mathbf{w}^T$, $\mathbf{x}$ being an input vector of dimension N and $\mathbf{w}$ a weight vector of the same dimension. Assume we have available pairs of training data $\{(\mathbf{x}^k, y^k), k = 1, ...t\}$, represented as an input matrix X of dimensions (t, n), and a target matrix $\mathbf{y}$ of dimensions $(t, 1)$. In modern matrix equation notation, the proposition of Legendre to solving $X\mathbf{w}^T = \mathbf{y}$ is to replace it by solving $X^T X \mathbf{w}^T = X^T \mathbf{y}$ (normal equations). The matrix $X^T X$ is now square and can potentially be inverted. If it is invertible, then we get the solution $\mathbf{w}^T = (X^T X)^{-1} X^T \mathbf{y}$. This solution minimizes $\|X\mathbf{w}^T - \mathbf{y}\|^2$ (*i.e.*, it is the least-square solution) and Gauss proved in 1929 that is it optimal in the sense that in a linear model where the errors have mean zero, are uncorrelated, and have equal variances, the best linear unbiased estimator of the coefficients is the least-square estimator. If $X^T X$ is not invertible (singular), we can make it invertible (regular) by adding a small positive constant to its diagonal elements. It is easy to understand why this should work. If you diagonalize a square matrix A as $A = U^T D U$, where U is an orthogonal matrix (*i.e.*, such that $U^T U = I$ and $U^{-1} = U^T$), then the inverse of A can be taken only if no element of D (the eigenvalues of A) is zero. If you replace A by $A + \gamma I$, where γ is a small positive value, the problem is fixed: $A^{-1} = U^T (D + \gamma I)^{-1} U$. Moore formalized this idea in 1920 in what is now known as the Moore-Penrose pseudo-inverse. The solution of $X\mathbf{w}^T = \mathbf{y}$ can be written as $\mathbf{w}^T = X^+ \mathbf{y}$, where $X^+ = \lim_{\gamma \to \infty}(X^T X + \gamma I)^{-1} X^T = \lim_{\gamma \to \infty} X^T (XX^T + \gamma I)^{-1}$. Regularization was further developed in the 1940's with the work of Tikhonov on inverse problems (Tikhonov, 1943). Hoerl in the 1960's, who coined the term "ridge regression" for the solution $\mathbf{w}^T = (X^T X + \gamma I)^{-1} X^T \mathbf{y}$, where γ is called the "ridge" (Hoerl, 1962).

This corresponds to replacing the least-square problem of minimizing $\|X\mathbf{w}^T - \mathbf{y}\|^2$ by that of minimizing $\|X\mathbf{w}^T - \mathbf{y}\|^2 + \gamma\|\mathbf{w}\|^2$. In the seventies, the idea was extended to kernel methods and regularization in reproducing kernel Hilbert spaces by (Wahba and Nashed, 1974). In the late eighties, it was extended to multi-layer neural network by Werbos under the name of "weight decay" (Werbos, 1988), and to radial-basis function networks by (Poggio and Girosi, 1990). At the beginning of the nineties, a Bayesian framework based on "Gaussian processes" was proposed by (MacKay, 1992). Simultaneously, the use of the L2-norm regularizer $\|\mathbf{w}\|^2$ with other loss functions than the square loss was popularized by the invention of Support Vector Machines (Boser et al., 1992). Then appeared the use of the L1-norm regularizer $\|\mathbf{w}\|$ with the square loss (Tibshirani, 1994) followed by a number of combinations of loss functions and regularizers, including regularized boosting, logistic regression, etc. (Friedman, 2000; Friedman et al., 2000).

References

Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *JMLR*, 5, 2004.

K. Bennett, J. Hu, X. Ji, G. Kunapuli, and J.-S. Pang. Model selection via bilevel optimization. In *Proc. IJCNN06*, pages 3588–3505, Vancouver, Canada, July 2006. INNS/IEEE.

B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, pages 144–152, 1992.

M. Boullé. Compression-based averaging of selective naive bayes classifiers. *JMLR*, 8: 1659–1685, July 2007.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

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

L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International, California, 1984.

R. Caruana, A. Munson, and A. Niculescu-Mizil. Getting the most out of ensemble selection. In *Proceedings of the 6th International Conference on Data Mining (ICDM '06)*, December 2006. Full-length version available as Cornell Technical Report 2006-2045.

R. Caruana and A. Niculescu-Mizil. Ensemble selection from libraries of models. In *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*, 2004. ISBN 1-58113-838-5.

G. Cawley. Leave-one-out cross-validation based model selection criteria for weighted ls-svms. In *IJCNN*, pages 1661–1668, 2006.

Y.W. Chang and C.J. Lin. Feature ranking using linear svm. In *JMLR W&CP*, volume 3, pages 53–64, WCCI2008 workshop on causality, Hong Kong, June 3-4 2008.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, USA, 2nd edition, 2001.

H. J. Escalante, M. Montes, and L. E. Sucar. Particle swarm model selection. *J. Mach. Learn. Res.*, 10:405–440, 2009. ISSN 1533-7928.

J. Fox. *Applied regression analysis and related models*. Sage publications, 1997.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.

J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression, a statistical view of boosting. *Annals of Statistics*, 28:337374, 2000.

P. Grünwald. A tutorial introduction to the minimum description length principle. In *Advances in Minimum Description Length: Theory and Applications*. MIT Press, 2005.

I. Guyon. Practical feature selection: from correlation to causality. In *Mining Massive Data Sets for Security*. IOS Press, 2008. URL `http://eprints.pascal-network.org/archive/00004038/01/PracticalFS.pdf`.

I. Guyon, C. Aliferis, G. Cooper, A. Elisseeff, J.-P. Pellet, P. Spirtes, and A. Statnikov. Design and analysis of the causation and prediction challenge. In *JMLR W&CP*, volume 3, pages 1–33, WCCI2008 workshop on causality, Hong Kong, June 3-4 2008. URL `http://jmlr.csail.mit.edu/papers/topic/causality.html`.

I. Guyon, C. Aliferis, and A. Elisseeff. *Causal Feature Selection*, pages 63–82. Chapman and Hall/CRC Press. Longer TR: `http://clopinet.com/isabelle/Papers/causalFS.pdf`, 2007a.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *JMLR*, 3:1157–1182, March 2003. URL `http://jmlr.csail.mit.edu/papers/volume3/guyon03a/guyon03a.pdf`.

I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, Editors. *Feature Extraction, Foundations and Applications*. Studies in Fuzziness and Soft Computing. With data, results and sample code for the NIPS 2003 feature selection challenge. Physica-Verlag, Springer, 2006a.

I. Guyon, D. Janzing, and B. Schölkopf. Causality: objectives and assessment. In *NIPS 2008 workshop on causality*, volume 7. JMLR W&CP, in press, 2009a.

I. Guyon, V. Lemaire, M. Boullé, Gideon Dror, and David Vogel. Analysis of the KDD cup 2009: Fast scoring on a large orange customer database. In *KDD cup 2009, in press*, volume 8. JMLR W&CP, 2009b.

I. Guyon, J. Makhoul, R. Schwartz, and V. Vapnik. What size test set gives good error rate estimates? *PAMI*, 20(1):52–64, 1998.

I. Guyon, A. Saffari, G. Dror, and J. Buhmann. Performance prediction challenge. In *IEEE/INNS conference IJCNN 2006*, Vancouver, Canada, July 16-21 2006b.

I. Guyon, A. Saffari, G. Dror, and G. Cawley. Agnostic learning vs. prior knowledge challenge. In *IEEE/INNS conference IJCNN 2007*, Orlando, Florida, August 12-17 2007b.

I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002. ISSN 0885-6125.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning, Data Mining, Inference and Prediction*. Springer Verlag, 2000.

A. E. Hoerl. Application of ridge analysis to regression problems. *Chemical Engineering Progress*, 58:54–59, 1962.

R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, pages 1137–1145, 1995.

R. Kohavi and G. John. Wrappers for feature selection. *Artificial Intelligence*, 97(1-2): 273–324, December 1997.

R. W. Lutz. Logitboost with trees applied to the WCCI 2006 performance prediction challenge datasets. In *Proc. IJCNN06*, pages 2966–2969, Vancouver, Canada, July 2006. INNS/IEEE.

D. MacKay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 4:448–472, 1992.

T. M. Mitchell. *Machine Learning*. McGraw-Hill Co., Inc., New York, 1997.

T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247(4945):978 – 982, February 1990.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

A. Saffari and I. Guyon. Quick start guide for CLOP. Technical report, Graz University of Technology and Clopinet, May 2006. URL `http://clopinet.com/CLOP/`.

B. Schoelkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge MA, 2002.

H. Stoppiglia, G. Dreyfus, R. Dubois, and Y. Oussar. Ranking a random feature for variable and feature selection. *JMLR*, 3:1399–1414, 2003.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

A.N. Tikhonov. On the stability of inverse problems. *Dokl. Akad. Nauk SSSR*, 39(5): 195–198, 1943.

V. Vapnik. *Statistical Learning Theory.* John Wiley and Sons, N.Y., 1998.

V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–180, 1971.

G. Wahba and M. Z. Nashed. Regularization and approximation of linear operator equations in reproducing kernel spaces. *Bull. Am. Math. Soc.*, 80(6):1213–1218, 1974.

P. Werbos. Backpropagation: Past and future. In *International Conference on Neural Networks*, pages 343–353. IEEE, IEEE press, 1988.

J. Weston, A. Elisseff, B. Schoelkopf, and M. Tipping. Use of the zero norm with linear models and kernel methods. *JMLR*, 3:1439–1461, 2003.

J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. L1 norm support vector machines. In *Advances in Neural Information Processing Systems*, 2003.